Adaptive and Passive Non-Visual Driver Assistance Technologies for the Blind Driver Challenge®

Paul C. D’Angio

DISSERTATION SUBMITTED TO THE FACULTY OF THE VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY
IN
MECHANICAL ENGINEERING

Dennis W. Hong, Chair
Alexander Leonessa, Co-Chair
Stephen C. Southward
Robert H. Sturges
Robin K. Panneton

May 30, 2012
Blacksburg, Virginia

Keywords: Driver Assistive Technologies, Model Predictive Control, Real-Time Neural Network Driver Modeling, Quazi-Newton Optimization, Non-Visual Human Computer Interfaces
Adaptive and Passive Non-Visual Driver Assistance Technologies for the Blind Driver Challenge®

Paul C. D'Angio

ABSTRACT

This work proposes a series of driver assistance technologies that enable blind persons to safely and independently operate an automobile on standard public roads. Such technology could additionally benefit sighted drivers by augmenting vision with suggestive cues during normal and low-visibility driving conditions. This work presents a non-visual human-computer interface system with passive and adaptive controlling software to realize this type of driver assistance technology. The research and development behind this work was made possible through the Blind Driver Challenge® initiative taken by the National Federation of the Blind.

The instructional technologies proposed in this work enable blind drivers to operate an automobile through the provision of steering wheel angle and speed cues to the driver in a non-visual method. This paradigm imposes four principal functionality requirements: Perception, Motion Planning, Reference Transformations, and Communication. The Reference Transformation and Communication requirements are the focus of this work and convert motion planning trajectories into a series of non-visual stimuli that can be communicated to the human driver.

This work proposes two separate algorithms to perform the necessary reference transformations described above. The first algorithm, called the Passive Non-Visual Interface Driver, converts the planned trajectory data into a form that can be understood and reliably interacted with by the blind driver. This passive algorithm performs the transformations through a method that is independent of the driver. The second algorithm, called the Adaptive Non-Visual Interface Driver, performs similar trajectory data conversions through methods that adapt to each particular driver. This algorithm uses Model Predictive Control supplemented with Artificial Neural Network driver models to generate non-visual stimuli that are predicted to induce optimal performance from the driver. The driver models are trained online and in real-time with a rapid training approach to continually adapt to changes in the driver’s dynamics over time.

The communication of calculated non-visual stimuli is subsequently performed through a Non-Visual Interface System proposed by this work. This system is comprised of two non-visual human computer interfaces that communicate driving information through haptic stimuli. The DriveGrip interface is pair of vibro-tactile gloves that communicate steering information through the driver’s hands and fingers. The SpeedStrip interface is a vibro-tactile cushion fitted on the driver’s seat that communicates speed information through the driver’s legs and back. The two interfaces work simultaneously to provide a continuous stream of directions to the driver as he or she navigates the vehicle.
Acknowledgements

This dissertation is dedicated to the members of the National Federation of the Blind and their incredible support for the research conducted in this work. The Blind Driver Challenge® initiative provided me with an avenue for interesting and challenging research that could be used to improve our society and serve as a beacon of hope to the blind community. I would like to thank Mark Riccobono and Anil Lewis for their outstanding cooperation, impeccable driving skills, and for all of the great experiences had throughout the Blind Driver Challenge®.

I would also like to thank my co-advisors, Dr. Dennis Hong and Dr. Alexander Leonessa, for all of their support and encouragement throughout my college career. Working with these professors has provided me with an incredible amount of experience in the research and development of robotics and has helped increase my passion for robotic systems. I thoroughly enjoyed working on the many different projects managed by each professor and learned many important engineering and life lessons from them. Dr. Hong and Dr. Leonessa continue to serve as role models and have become true friends of mine over the years. For all of this, I thank you both with the utmost gratitude.

Thank you to the other members of my dissertation committee: Dr. Stephen Southward, Dr. Robert Sturges, and Dr. Robin Panneton. You have all offered time in your busy schedules to help guide me along my research and complete my doctorate in an efficient manner. An additional thank you is given to Dr. Southward for his assistance in the significant amount of testing conducted at the Virginia International Raceway.

Much gratitude is given to my colleagues in the Blind Driver Challenge® team. Ryan Colby worked closely with me throughout the entire project and helped contribute to the development, testing, and demonstration of the technologies. I would also like to thank our “minions” that served or volunteered for the senior design team during the past few years. Your help was greatly appreciated and made all of this work possible. Thanks also go to TORC Robotics for their fantastic work on the Blind Driver Challenge® platform and for all of the great experiences had leading up to the Rolex-24 demonstration.

Many thanks go to my family and friends for their incredible amounts of support and tolerance throughout my research. My parents, Paul D’Angio Sr. and Sandra D’Angio, were always there for me in my successes and failures and have given me all the tools that I could ever need to excel in life. Samantha Smarte has stood by my side and has given me incredible amounts of support and inspiration throughout my graduate career.

My final thanks go to my colleagues in RoMeLa. I thoroughly enjoyed working with all of you and appreciate the help and friendship you have provided over the past years.
# Table of Contents

Section 1: Introduction.................................................................................................................. 1
  1.1: The Blind Driver Challenge® by the National Federation of the Blind ....................... 2
  1.2: The Virginia Tech Research Effort ................................................................................. 4
    1.2.1: History......................................................................................................................... 4
    1.2.2: The 2010-2011 Research Proposal ........................................................................... 7
  1.3: Problem Statement ............................................................................................................. 10
  1.4: Review of Literature ......................................................................................................... 13
    1.4.1: Driver Modeling .......................................................................................................... 13
    1.4.2: Motion Planning .......................................................................................................... 18
    1.4.3: Human-Computer Interface ........................................................................................ 23
    1.4.4: Model Predictive Control ............................................................................................. 25
  1.5: Contribution ....................................................................................................................... 28
    1.5.1: Proposed Non-Visual Interface Drivers ...................................................................... 28
    1.5.2: Proposed Non-Visual Interface System ....................................................................... 29
  1.6: Outline of Dissertation ..................................................................................................... 29

Section 2: The Blind Driver Challenge® Research Platform ............................................... 30
  2.1: The TORC ByWire XGV™ Standard Platform ................................................................. 30
    2.1.1: The Ford 2010 Escape Hybrid SUV ........................................................................... 31
    2.1.2: TORC ByWire XGV™ Conversion ............................................................................. 31
  2.2: Environmental Perception Sensors .................................................................................... 35
    2.2.1: Inertial Navigation Solution ......................................................................................... 35
    2.2.2: Obstacle Detection and Identification ......................................................................... 38
    2.2.3: Active Road Detection ................................................................................................ 42
  2.3: TORC Robotics Semi-Autonomous Systems .................................................................. 44
    2.3.1: Computer Hardware ................................................................................................... 44
    2.3.2: Software Components ................................................................................................. 47
  2.4: System Network and Communication .............................................................................. 55
    2.4.1: Network Switch and Wireless Access ......................................................................... 55
    2.4.2: The Operator Control Unit (OCU) ............................................................................. 56
    2.4.3: Network Diagram ......................................................................................................... 57
    2.4.4: JAUS Interoperability ................................................................................................. 58
Section 5: The Adaptive Non-Visual Interface Driver (ANVID) ........................................ 227
  5.1: Functional Requirements .............................................................................. 228
    5.1.1: Motion Profile Sawtooth Patterns ......................................................... 228
    5.1.2: Driver Time Delay ................................................................................. 228
    5.1.3: Driver Tracking Characteristics .............................................................. 229
  5.2: Functional Overview .................................................................................. 230
  5.3: Theoretical Definitions .............................................................................. 232
    5.3.1: Model Predictive Control (MPC) .......................................................... 232
    5.3.2: Driver Modeling ..................................................................................... 235
    5.3.3: Reference Optimization ......................................................................... 254
  5.4: Algorithm Definitions ................................................................................ 268
    5.4.1: ANVID() ............................................................................................... 269
    5.4.2: DriverModeling() ................................................................................. 269
    5.4.3: TrainModel() ........................................................................................ 271
5.4.4 : EvaluateModel() .......................................................... 272
5.4.5 : MEKA() ............................................................................. 273
5.4.6 : ShieldModel() ................................................................. 274
5.4.7 : ModelPredictiveController() .............................................. 275
5.4.8 : OptimizeReferenceSignals() ............................................. 276
5.4.9 : QNO() ............................................................................. 277
5.4.10 : CalculateNonVisualStimuli() .......................................... 279

5.5 : Software Implementation .................................................... 281

5.5.1 : Development Tools ............................................................ 281
5.5.2 : Algorithm Definitions ....................................................... 281
5.5.3 : Software Execution ............................................................ 290

5.6 : Performance Analysis .......................................................... 294

5.6.1 : Driver Modeling ................................................................. 294
5.6.2 : Model Predictive Controller ............................................... 299
5.6.3 : Comprehensive System Performance ................................... 303

Section 6 : Conclusions and Future Research .............................. 332

6.1 : The Non-Visual Interface System (NVIS) ............................ 333

6.1.1 : DriveGrip ........................................................................... 333
6.1.2 : SpeedStrip ......................................................................... 335
6.1.3 : Simultaneous Interface Usage ............................................ 336
6.1.4 : Associated Control Hardware ............................................ 337

6.2 : The Passive Non-Visual Interface Driver (PNVID) ............... 337

6.2.1 : Functional Requirement Satisfaction .................................. 338
6.2.2 : Software Implementation .................................................. 338
6.2.3 : Comprehensive System Performance ............................... 338
6.2.4 : Rolex 24 Demonstration .................................................... 339


6.3.1 : Driver Modeling ................................................................. 340
6.3.2 : Reference Optimization .................................................... 341
6.3.3 : Software Implementation .................................................. 341
6.3.4 : MPC and Driver Modeling Performance ........................... 342
6.3.5 : Comprehensive System Performance ............................... 342

6.4 : Future Research ................................................................. 343
6.4.1 : Improvement of the Non-Visual Interface System ..................................................343
6.4.2 : Improvement of the ANVID Software ........................................................................345
6.4.3 : Development of Informational Interfaces ....................................................................346
6.4.4 : Extensions of the Non-Visual Interface System ..........................................................346

Section 7 : References ...........................................................................................................347

Appendix A : Non-Visual Interface Performance Analysis Data Sets .................................359
  A.1 : DriveGrip Performance Analysis Full Data Set ..........................................................359
    A.1.1 : Step Reference Steering Wheel Angle Signals .......................................................359
    A.1.2 : Ramp Reference Steering Wheel Angle Signals ......................................................365
  A.2 : SpeedStrip Performance Analysis Data Set .................................................................371
    A.2.1 : Step Reference Speed Signals ................................................................................371
    A.2.2 : Ramp Reference Speed Signals ..............................................................................373
List of Figures

Figure 1-1: Instructional solution to the Blind Driver Challenge®. ............................................. 10
Figure 1-2: The relationship between motion planning, driver modeling, and human-
computer interface research areas in adaptive and passive transformation algorithms. 12
Figure 2-1: The TORC ByWire XGV™ research platform .......................................................... 30
Figure 2-2: TORC ByWire XGV™ Emergency Stop Disable/Pause and Manual Override
Systems ..................................................................................................................................... 34
Figure 2-3: Inertial Navigation System Sensors. NovAtel® SPAN-CPT™ INS (left) and
NovAtel® GPS-302L GPS Antenna (right) .................................................................................. 36
Figure 2-4: TORC Robotics Localization Board ........................................................................ 36
Figure 2-5: Mounting configuration for (A) NovAtel® SPAN-CPT™ (B) TORC Robotics
Localization Board (C) Vibration Isolating Mounts for equipment rack ................................ 37
Figure 2-6: Inertial Navigation System Connection Architecture .................................................. 38
Figure 2-7: Ibeo LUX Laser Scanner ........................................................................................... 39
Figure 2-8: Ibeo LUX Fusion System. Ibeo Sync Box (left) and Ibeo ECU (right) ................. 39
Figure 2-9: Mounting configuration for 2x forward-facing Ibeo LUX sensors ......................... 40
Figure 2-10: Mounting configuration for 1x rear-facing Ibeo LUX sensor ................................. 40
Figure 2-11: Obstacle Detection and Identification Horizontal Field of View ......................... 40
Figure 2-12: Ibeo LUX Fusion System Mounting Configuration. Ibeo ECU (A) and Ibeo
Sync Box (B) ............................................................................................................................. 41
Figure 2-13: Obstacle Detection/Identification System Connection Architecture ................. 41
Figure 2-14: Allied Vision Technologies Prosilica GC1290C Camera ...................................... 42
Figure 2-15: Mounting Configuration for 2x Allied Vision Technologies Prosilica
GC1290C ..................................................................................................................................... 43
Figure 2-16: Active Road Detection Horizontal Field of View .................................................. 43
Figure 2-17: Active Road Detection System Connection Architecture ..................................... 43
Figure 2-18: TORC Robotics AutonoNav™ Dual Computer System ........................................... 44
Figure 2-19: Mounting Configuration for TORC Robotics AutonoNav™ System (A) ......... 46
Figure 2-20: TORC Robotics AutonoNav™ Dual Computer System Connection
Architecture ................................................................................................................................. 46
Figure 2-21: An example of a standard motion within a Motion Profile .................................... 53
Figure 2-22: An example of a complete Motion Profile and the corresponding desired
trajectory it was derived from .................................................................................................... 53
Figure 2-23: Mounting configuration for the (A) D-Link DGS-1244T Ethernet Switch and
the (B) Ubiquiti PicoStation2HP Wireless Access Point ................................................................ 56
Figure 2-24: System Network Architecture of the Blind Driver Challenge® Platform. ... 57
Figure 2-25: Software/Hardware Module (top) and Strictly Software Module (bottom)
functionality .................................................................................................................................. 59
Figure 2-26: Graphical depiction of the JAUS system topology ............................................... 60
Figure 2-27: JAUS Message Composition .................................................................................... 62
Figure 1: Redesigned DriveGrip System Layout. Components include: Vibratory Motors (red), Standardized Connector (blue), Wiring Harness (Black), and Mounting Glove (gray).

Figure 2: DriveGrip System Headrest Mount on Driver Seat (Rear View).

Figure 3: DriveGrip System Headrest Mount with Breakout Box (Rear View).

Figure 4: DriveGrip Breakout Box Schematic.

Figure 5: DriveGrip Left/Right Glove Wiring Schematic.

Figure 6: DriveGrip Glove on the left hand of a driver.

Figure 7: DriveGrip vibration component with hot-glue shell for protection, shape and comfort.

Figure 8: DriveGrip main wire harness layout.

Figure 9: Completed DriveGrip Gloves – Top View.

Figure 10: DriveGrip Glove on the left hand of a driver.

Figure 11: DriveGrip Breakout Box Schematic.

Figure 12: DriveGrip System Headrest Mount with Breakout Box (Rear View).

Figure 13: DriveGrip System Headrest Mount on Driver Seat (Rear View).

Figure 14: Average driver response to a 60° step reference signal utilizing various DriveGrip operation configurations defined in Table 3-2.

Figure 15: Lateral deviation ranges caused by the strict error threshold tolerance (\(TI = 0.1°\)) at different vehicle speeds.

Figure 16: Lateral deviation ranges caused by the lenient error threshold tolerance (\(TI = 5°\)) at different vehicle speeds.

Figure 17: Average driver response to a 60° step reference signal utilizing the TFLT DOC defined in Table 3-2.

Figure 18: Average driver response to a 10°/s ramp reference signal utilizing various DriveGrip operation configurations defined in Table 3-2.

Figure 19: Average driver response to a 10°/s ramp reference signal utilizing the TFST DOC defined in Table 3-2.

Figure 20: Average driver response to an arbitrary reference signal utilizing the FFST and TFST DriveGrip operation configurations defined in Table 3-2.

Figure 21: Average driver response to an arbitrary reference signal utilizing the FFLT and TFLT DriveGrip operation configurations defined in Table 3-2.

Figure 22: Time-shifted Mean Absolute Error between an arbitrary steering reference signal and the driver’s response utilizing each of the DriveGrip operation configurations defined in Table 3-2.
Figure 3-23: Comparison of the average driver response to an arbitrary reference signal utilizing the FFST and FFLT operation configurations defined in Table 3-2. .......................... 97
Figure 3-24: Example of speed-dependent steering wheel angle error thresholds for a new Hybrid DriveGrip operation configuration. .......................................................... 98
Figure 3-25: Average Driver #1 steering response before and after seven hour DriveGrip usage period. .......................................................................................... 100
Figure 3-26: Average Driver #2 steering response before and after seven hour DriveGrip usage period. .......................................................................................... 100
Figure 3-27: SpeedStrip System Layout. Components include: Vibratory Motors (red), Standardized Connector (blue), Wiring Harness (Green), and Mounting Seat Cushion (gray). .......................................................................................... 103
Figure 3-28: HoMedics® Vibration Motor used in the SpeedStrip Interface .......... 104
Figure 3-29: Electrical Schematic of the SpeedStrip Non-Visual Interface.......... 105
Figure 3-30: Wiring harness stress relief for the SpeedStrip interface vibration components (top view). .......................................................................................... 106
Figure 3-31: Reconfigurable positioning of SpeedStrip vibration components with Velcro® strips.......................................................... 106
Figure 3-32: Completed SpeedStrip Non-Visual Interface mounted inside of the Blind Driver Challenge® vehicle.......................................................... 107
Figure 3-33: SpeedStrip and DriveGrip interface headrest mount ................. 108
Figure 3-34: Driver responses to a 5m/s step speed reference signal using the FE SOC defined in Table 3-7 .................................................................................. 110
Figure 3-35: Driver responses to a 5m/s step speed reference signal using the TE SOC defined in Table 3-7 .................................................................................. 111
Figure 3-36: Driver responses to a 0.3m/s² ramp speed reference signal using the FE SOC defined in Table 3-7 .................................................................................. 114
Figure 3-37: Driver responses to a 0.3m/s² ramp speed reference signal using the TE SOC defined in Table 3-7 .................................................................................. 114
Figure 3-38: Driver responses to an arbitrary reference signal using the FE and TE SOCs defined in Table 3-7 .................................................................................. 117
Figure 3-39: Time-shifted Mean Absolute Error between an arbitrary speed reference signal and the driver’s response utilizing each of the SpeedStrip operation configurations defined in Table 3-7. .......................................................... 117
Figure 3-40: Deceleration vs. Braking using the FE SOC defined in Table 3-7...... 119
Figure 3-41: Deceleration vs. Braking using the TE SOC defined in Table 3-7..... 119
Figure 3-42: Driver speed response before and after a three hour testing period using the FE SOC defined in Table 3-7. .......................................................... 121
Figure 3-43: Driver steering and speed responses during simultaneous DriveGrip and SpeedStrip interaction using the FFLT and TE operation configurations defined in Table 3-2 and Table 3-7. .......................................................... 122
Figure 3-44: Time-shifted Mean Absolute Error of driver steering and speed response during simultaneous DriveGrip and SpeedStrip interaction using the FFLT and TE operation configurations defined in Table 3-2 and Table 3-7. .................................................. 123
Figure 3-45: Example schematic of single non-visual interface vibration element. ..... 125
Figure 3-46: Non-Visual Interface high level layout design. ........................................... 126
Figure 3-47: National Instruments® USB-6501 OEM Digital Input/Output Board used in the Non-Visual Interface Controller. ................................................................. 127
Figure 3-48: Custom Darlington Array Interface PCB used in the Non-Visual Interface Controller......................................................................................................................... 128
Figure 3-49: Non-Visual Interface Controller Schematic Diagram.............................. 129
Figure 3-50: Completed design of the Non-Visual Interface Controller (NVIC). .......... 130
Figure 3-51: Internal view of the completed Non-Visual Interface Controller (NVIC) ... 131
Figure 3-52: Non-Visual Interface System Connection Architecture ............................ 133
Figure 4-1: Complete instructional solution to the Blind Driver Challenge® utilizing the PNVID software. The Research Platform refers to the Blind Driver Challenge® Research Platform and is described in detail in Section 2. The NVIS refers to the Non-Visual Interface System and is presented in detail in Section 3. .................................................. 134
Figure 4-2: Future desired curvature reference signal derived from the most recent motion profile and the current actual curvature. .......................................................... 136
Figure 4-3: Desired curvature sawtooth pattern created by rapid regeneration of motion profiles. The motion profile regeneration intervals are denoted by the red dashed lines and, in this example, occur at 4Hz. The dotted lines indicate discarded motion profile data. The heavy black line displays the resulting desired curvature signal. ............... 137
Figure 4-4: Actual desired curvature sawtooth pattern derived from TORC AutonoNav™ motion profile data during a driving experiment.......................... 137
Figure 4-5: Approximate linear relationship between steering wheel angle and vehicle curvature at varying speeds defined by (4.2). .......................................................... 138
Figure 4-6: Propagation of actual sawtooth patterns from Figure 4-4 into the desired steering wheel angle signal through (4.2)................................................................. 139
Figure 4-7: Propagation of sawtooth patterns from the desired steering wheel angle error (Figure 4-6) into the instantaneous steering wheel angle error signal through (3.1). The FFLT DOC error thresholds defined in Table 3-2 are overlaid to demonstrate the sawtooth pattern propagation into the non-visual stimuli created by the DriveGrip interface. ................................................................................................................................. 140
Figure 4-8: Example of sawtooth filtering transformations on the reference curvature signal. ............................................................................................................................................... 148
Figure 4-9: Sawtooth pattern behaviors and associated transformation actions. Each of the eight cases is labeled from C1 through C8.................................................. 149
Figure 4-10: Detailed example of Case 1 behaviors and transformation action. ........ 150
Figure 4-11: Detailed example of Case 2 behaviors and transformation action. ........ 151
Figure 4-12: Detailed example of Case 3 behaviors and transformation action. ........ 152
Figure 4-13: Detailed example of Case 4 behaviors and transformation action. ....... 153
Figure 4-14: LabVIEW Graphical Programming example: Generate Sine Point.vi
definition and Block Diagram................................................................. 163
Figure 4-15: Example of the transformed reference steering (TRS) signal generated by
the PNVID algorithm at a constant speed of 10m/s. The TRS is derived from the
untransformed reference steering (URS) signal and the current steering (CS) signal
through the algorithms defined in Section 4.2.5......................................... 172
Figure 4-16: Instantaneous steering wheel angle error calculated from the data shown in
Figure 4-15 using (3.1) and the FFLT DOC defined in Table 3-2.......................... 172
Figure 4-17: Time-shifted mean absolute error (MAE) comparison for determining the
phase shift between the unfiltered and filtered instantaneous steering wheel angle error
signal shown in Figure 4-16. ........................................................................ 173
Figure 4-18: Typical reference speed signal generated by the PNVID and the
associated driver response utilizing the TE SOC defined in Table 3-7.................... 174
Figure 4-19: Instantaneous speed error calculated from the data shown in Figure 4-18
using (3.1) and the TE SOC defined in Table 3-7. ........................................... 175
Figure 4-20: Typical driver response to a reference steering wheel angle signal
generated by the PNVID. ............................................................................ 176
Figure 4-21: Instantaneous steering wheel angle error derived from the reference
steering wheel angle signal and driver response signal shown in Figure 4-20......... 177
Figure 4-22: Road course utilized for comprehensive Blind Driver Challenge® system
testing. Course driving direction is clockwise. ................................................. 179
Figure 4-23: Example of a lateral lane deviation calculation using (4.5)-(4.7). .......... 181
Figure 4-24: Trajectory navigated by a blind driver on the VIR Patriot Course
characterized in Figure 4-22. The vehicle trajectory (blue) starts at 0,0 and makes a full
clockwise loop. Lane edges (black) and obstacles (red) are also shown. ............ 183
Figure 4-25: Lateral lane and speed deviations over the length of the entire VIR Patriot
course. .......................................................................................................... 183
Figure 4-26: Reference steering wheel angle generated by the PNVID and the
associated blind driver response over the entire VIR Patriot course. ................. 184
Figure 4-27: Instantaneous steering wheel angle error exhibited by the blind driver over
the entire VIR Patriot course. ...................................................................... 184
Figure 4-28: Reference speed generated by the PNVID and the associated blind driver
response over the entire VIR Patriot course...................................................... 185
Figure 4-29: Instantaneous speed error exhibited by the blind driver over the entire VIR
Patriot course. .............................................................................................. 185
Figure 4-30: Trajectory navigated by a blind driver on the low-curvature snake section of
the VIR Patriot Course. Timing is indicated by the green crosses...................... 186
Figure 4-31: Lateral lane and speed deviations over the low-curvature snake section of
the VIR Patriot course. ................................................................................ 187
Figure 4-32: Reference speed generated by the PNVID and the associated blind driver response over the low-curvature snake section of the VIR Patriot course. ........................................... 187
Figure 4-33: Instantaneous speed error exhibited by the blind driver over the low-curvature snake section of the VIR Patriot course. ................................................................. 187
Figure 4-34: Reference steering wheel angle generated by the PNVID and the associated blind driver response over the low-curvature snake course section. ........ 188
Figure 4-35: Instantaneous steering wheel angle error exhibited by the blind driver over the low-curvature snake section of the VIR Patriot course. ................................. 188
Figure 4-36: Trajectory navigated by a blind driver on the high-speed straightaway section of the VIR Patriot course. Timing is indicated by the green crosses and labeled accordingly. ........................................................................................................ 189
Figure 4-37: Lateral lane and speed deviations over the high-speed straightaway section of the VIR Patriot course. .................................................................................................................. 189
Figure 4-38: Reference speed generated by the PNVID and the associated blind driver response on the high-speed straightaway section of the VIR Patriot course. .......................... 190
Figure 4-39: Instantaneous speed error exhibited by the blind driver over the high-speed straightaway section of the VIR Patriot course. ......................................................... 190
Figure 4-40: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the high-speed straightaway course section. ..... 192
Figure 4-41: Instantaneous steering wheel angle error exhibited by the blind driver over the high-speed straightaway section of the VIR Patriot course. ......................................................... 192
Figure 4-42: Trajectory navigated by a blind driver on the medium-curvature snake section of the VIR Patriot course. Timing is indicated by the green crosses and labeled accordingly. ........................................................................................................ 194
Figure 4-43: Lateral lane and speed deviations over the medium-curvature snake section of the VIR Patriot course. .................................................................................................................. 195
Figure 4-44: Reference speed generated by the PNVID and the associated blind driver response on the medium-curvature snake section of the VIR Patriot course. ............ 195
Figure 4-45: Instantaneous speed error exhibited by the blind driver over the medium-curvature snake section of the VIR Patriot course. ......................................................... 196
Figure 4-46: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the medium-curvature snake course section. ..... 196
Figure 4-47: Instantaneous steering wheel angle error exhibited by the blind driver over the medium-curvature snake section of the VIR Patriot course................................. 196
Figure 4-48: Trajectory navigated by a blind driver on the dynamic obstacle field section of the VIR Patriot course. Timing is indicated by the green crosses. ...................... 197
Figure 4-49: Lateral lane and speed deviations over the dynamic obstacle field section of the VIR Patriot course. .................................................................................................................. 198
Figure 4-50: Reference speed generated by the PNVID and the associated blind driver response on the dynamic obstacle field section of the VIR Patriot course. ............ 198
Figure 4-51: Instantaneous speed error exhibited by the blind driver over the dynamic obstacle field section of the VIR Patriot course. ................................................................. 199
Figure 4-52: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the dynamic obstacle field course section. .......... 200
Figure 4-53: Instantaneous steering wheel angle error exhibited by the blind driver over the dynamic obstacle field section of the VIR Patriot course. ........................................ 201
Figure 4-54: Trajectory navigated by a blind driver on the gate section of the VIR Patriot course. Timing is indicated by the green crosses and labeled accordingly. .......... 202
Figure 4-55: Lateral lane and speed deviations over the gate section of the VIR Patriot course. ................................................................................................................................. 202
Figure 4-56: Reference speed generated by the PNVID and the associated blind driver response on the gate section of the VIR Patriot course. ................................. 203
Figure 4-57: Instantaneous speed error exhibited by the blind driver over the gate section of the VIR Patriot course. ........................................................................................................ 203
Figure 4-58: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the gate course section. ...................................... 204
Figure 4-59: Instantaneous steering wheel angle error exhibited by the blind driver over the gate section of the VIR Patriot course. ............................................................................. 204
Figure 4-60: Trajectory navigated by a blind driver on the high-curvature snake section of the VIR Patriot course. Timing is indicated by the green crosses. ................. 205
Figure 4-61: Lateral lane and speed deviations over the high-curvature snake section of the VIR Patriot course. ........................................................................................................ 205
Figure 4-62: Reference speed generated by the PNVID and the associated blind driver response on the high-curvature snake section of the VIR Patriot course. .............. 206
Figure 4-63: Instantaneous speed error exhibited by the blind driver over the high-curvature snake section of the VIR Patriot course. .......................................................................................... 206
Figure 4-64: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the high-curvature snake course section .......... 207
Figure 4-65: Instantaneous steering wheel angle error exhibited by the blind driver over the high-curvature snake section of the VIR Patriot course................................. 207
Figure 4-66: Trajectory navigated by a blind driver on the sweeping turn section of the VIR Patriot course. Timing is indicated by the green crosses. ....................... 209
Figure 4-67: Lateral lane and speed deviations over the sweeping turn section of the VIR Patriot course. ........................................................................................................ 209
Figure 4-68: Reference speed generated by the PNVID and the associated blind driver response on the sweeping turn section of the VIR Patriot course. ...................... 210
Figure 4-69: Instantaneous speed error exhibited by the blind driver over the sweeping turn section of the VIR Patriot course .......................................................... 210
Figure 4-70: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the sweeping turn course section ......................... 211
Figure 4-71: Instantaneous steering wheel angle error exhibited by the blind driver over the sweeping turn section of the VIR Patriot course. ................................. 211
Figure 4-72: Trajectory navigated by a blind driver on the static obstacle field section of the VIR Patriot course.  Timing is indicated by the green crosses. .......................... 213
Figure 4-73: Lateral lane and speed deviations over the static obstacle field section of the VIR Patriot course. ...................................................................................... 214
Figure 4-74: Reference speed generated by the PNVID and the associated blind driver response on the sweeping turn section of the VIR Patriot course. ........................... 214
Figure 4-75: Instantaneous speed error exhibited by the blind driver over the sweeping turn section of the VIR Patriot course. ................................................................. 214
Figure 4-76: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the static obstacle field course section.......................... 215
Figure 4-77: Instantaneous steering wheel angle error exhibited by the blind driver over the static obstacle field section of the VIR Patriot course. ............................... 215
Figure 4-78: Trajectory navigated by a blind driver at the finish line section of the VIR Patriot course. ................................................................................................. 216
Figure 4-79: Distance between the front of the Blind Driver Challenge® vehicle and the finish line stop point as a function of time at the finish line section of the VIR Patriot course. ...................................................................................... 217
Figure 4-80: Reference speed generated by the PNVID and the associated blind driver response at the finish line section of the VIR Patriot course. ............................... 218
Figure 4-81: Instantaneous speed error exhibited by the blind driver at the finish section of the VIR Patriot course. ................................................................................ .... 218
Figure 4-82: Lateral lane deviation throughout the finish line section of the VIR Patriot course. .................................................................................................................. 219
Figure 4-83: Reference steering wheel angle generated by the PNVID and the associated blind driver response at the finish line course section. ................................. 219
Figure 4-84: Instantaneous steering wheel angle error exhibited by the blind driver over the finish line section of the VIR Patriot course. .............................................. 219
Figure 4-85: Road course utilized at the Daytona International Speedway for the Rolex 24 Blind Driver Challenge® public demonstration. .............................................. 220
Figure 4-86: Deployment of dynamic obstacles exemplified during a practice session at the Virginia International Raceway Patriot course. .................................................... 221
Figure 4-87: Florida Congressman John Mica (right) hands the keys to the Blind Driver Challenge® vehicle to Mark Riccobono (middle) and Dr. Marc Maurer (left). .............. 225
Figure 4-88: Mark Riccobono driving the Blind Driver Challenge® vehicle on the 18° bank of the Daytona International Speedway shortly after the starting line. .......... 225
Figure 4-89: Mark Riccobono approaching the gate course section near the finish line. ................................................................................................................................. 226
Figure 4-90: Mark Riccobono (second from left) and Dr. Dennis Hong (right) celebrating the successful demonstration. ................................................................. 226
Figure 5-1: Complete instructional solution to the Blind Driver Challenge® utilizing the ANVID software. The Research Platform refers to the Blind Driver Challenge® Research Platform and is described in detail in Section 2. The NVIS refers to the Non-Visual Interface System and is presented in detail in Section 3. .................................................. 227
Figure 5-2: High-level functionality diagram of the ANVID software.......................... 231
Figure 5-3: General process flowchart for the Model Predictive Control method. ....... 233
Figure 5-4: Example of how MPC is used to control the human driver with fictional data. UREF is the untransformed reference steering signal and URESP is the driver’s response to UREF. TREF is the suboptimal transformed reference steering signal calculated by MPC and TRESP is the driver’s response to TREF. Note that the TRESP accurately recreates UREF. ............................................................... 234
Figure 5-5: Typical usage of a Time Series Prediction Neural Network (TSPNN) ...... 236
Figure 5-6: Full Blind Driver Challenge® system block diagram. ............................. 237
Figure 5-7: Simplified Blind Driver Challenge® system block diagram..................... 238
Figure 5-8: Decoupled simplified Blind Driver Challenge® system block diagram ..... 239
Figure 5-9: Input/Output definition for the driver steering and speed TSPNN prediction models. ........................................................................................................ 241
Figure 5-10: Input and output diagrams for both steering and speed TSPNN driver models. The inner loop (40Hz) iteration count is signified by (i) while the reference (8Hz) iteration count is signified by (j). ............................................................... 242
Figure 5-11: Multi-Layer Perceptron (MLP) network configuration for the steering and speed TSPNN driver models. The particular network is shown for the driver speed model; the driver steering model can be found by replacing S with θ. .................. 243
Figure 5-12: The hyperbolic tangent sigmoid activation function.............................. 244
Figure 5-13: Online training for multi-output ANNs using buffers. ......................... 250
Figure 5-14: Time-shifted supervised learning for TSPNN prediction models. .......... 251
Figure 5-15: Sample buffers utilized for ANVID driver model training. ................... 252
Figure 5-16: Standard three-layer MLP neural network. The output of each neuron is labeled as the capital of the respective neuron label............................... 259
Figure 5-17: Discontinuities in chain formations of the full reference signal. Square markers represent the time at which the QNO generated a new suboptimal reference action signal to replace the previous obsolete reference action signal.......................... 266
Figure 5-18: Accuracy of the steering TSPNN driver model with respect to chain predictions at 750ms intervals................................................................. 296
Figure 5-19: Accuracy of the steering TSPNN driver model with respect to overlapped chain predictions from the Model Predictive Control process........................ 297
Figure 5-20: Accuracy of the speed TSPNN driver model with respect to chain predictions at 750ms intervals....................................................................... 298
Figure 5-21: Accuracy of the speed TSPNN driver model with respect to overlapped chain predictions from the Model Predictive Control process...................... 299
Figure 5-22: Typical suboptimal reference steering wheel angle signal generated by the Model Predictive Controller component of the ANVID software. ................................................................. 302
Figure 5-23: Typical suboptimal reference speed signal generated by the Model Predictive Controller component of the ANVID software. ................................................................. 302
Figure 5-24: Road course utilized for comprehensive Blind Driver Challenge® system testing. .............................................................................................................................. 304
Figure 5-25: Trajectory (blue) navigated by a blind driver on the DIS simulation course characterized in Figure 5-24. The vehicle trajectory (blue) starts at 0,0 and ends at - 533, 167. Lane edges (solid black) and lane centerline (dotted black) are also shown. ................................................................................................................................. 306
Figure 5-26: Lateral lane and speed deviations over the length of the entire DIS simulation course. ................................................................................................................. 306
Figure 5-27: Reference steering wheel angle generated by the ANVID and the associated driver response over the entire DIS simulation course. ......................... 307
Figure 5-28: Instantaneous steering wheel angle error exhibited by the driver over the entire DIS simulation course. ................................................................. 307
Figure 5-29: Reference speed generated by the ANVID and the associated driver response over the entire DIS simulation course. ........................................... 308
Figure 5-30: Instantaneous speed error exhibited by the driver over the entire DIS simulation course. ............................................................................................ 308
Figure 5-31: Trajectory navigated by the driver on the starting straightaway section of the DIS simulation course. Timing is indicated by the green crosses. .......... 309
Figure 5-32: Lateral lane and speed deviations over the starting straightaway section of the DIS simulation course. ........................................................................ 310
Figure 5-33: Reference speed generated by the ANVID and the associated driver response over the starting straightaway section of the DIS simulation course. .......... 310
Figure 5-34: Instantaneous speed error exhibited by the driver over the starting straightaway section of the DIS simulation course. ........................................... 311
Figure 5-35: Reference steering wheel angle generated by the ANVID and the associated driver response over the starting straightaway course section. .......... 312
Figure 5-36: Instantaneous steering wheel angle error exhibited by the driver over the starting straightaway section of the DIS simulation course. ......................... 312
Figure 5-37: Trajectory navigated by the driver on the medium-curvature turn section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly. ................................................................. 313
Figure 5-38: Lateral lane and speed deviations over the medium-curvature turn section of the DIS simulation course. ........................................................................ 314
Figure 5-39: Reference speed generated by the ANVID and the associated driver response on the medium-curvature turn section of the DIS simulation course .......... 314
Figure 5-40: Instantaneous speed error exhibited by the driver over the medium-curvature turn section of the DIS simulation course. ........................................... 315
Figure 5-41: Reference steering wheel angle generated by the ANVID and the associated driver response on the medium-curvature turn course section. 316
Figure 5-42: Instantaneous steering wheel angle error exhibited by the driver over the medium-curvature turn section of the DIS simulation course. 316
Figure 5-43: Trajectory navigated by the driver on the hairpin turn #1 section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly. 317
Figure 5-44: Lateral lane and speed deviations over the hairpin turn #1 section of the DIS simulation course. 318
Figure 5-45: Reference speed generated by the ANVID and the associated driver response on the hairpin turn #1 section of the DIS simulation course. 318
Figure 5-46: Instantaneous speed error exhibited by the driver over the hairpin turn #1 section of the DIS simulation course. 318
Figure 5-47: Reference steering wheel angle generated by the ANVID and the associated driver response on the hairpin turn #1 course section. 319
Figure 5-48: Instantaneous steering wheel angle error exhibited by the driver over the hairpin turn #1 section of the DIS simulation course. 320
Figure 5-49: Trajectory navigated by the driver on the left-handed dogleg section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly. 320
Figure 5-50: Lateral lane and speed deviations over the left-handed dogleg section of the DIS simulation course. 321
Figure 5-51: Reference speed generated by the ANVID and the associated driver response on the left-handed dogleg section of the DIS simulation course. 321
Figure 5-52: Instantaneous speed error exhibited by the driver over the left-handed dogleg section of the DIS simulation course. 322
Figure 5-53: Reference steering wheel angle generated by the ANVID and the associated driver response on the left-handed dogleg course section. 323
Figure 5-54: Instantaneous steering wheel angle error exhibited by the driver over the left-handed dogleg section of the DIS simulation course. 323
Figure 5-55: Trajectory navigated by the driver on the hairpin turn #2 section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly. 324
Figure 5-56: Lateral lane and speed deviations over the hairpin turn #2 section of the DIS simulation course. 325
Figure 5-57: Reference speed generated by the ANVID and the associated driver response on the hairpin turn #2 section of the DIS simulation course. 325
Figure 5-58: Instantaneous speed error exhibited by the driver over the hairpin turn #2 section of the DIS simulation course. 325
Figure 5-59: Reference steering wheel angle generated by the ANVID and the associated driver response on the hairpin turn #2 course section. 326
Figure 5-60: Instantaneous steering wheel angle error exhibited by the driver over the hairpin turn #2 section of the DIS simulation course. ................................................................. 327
Figure 5-61: Trajectory navigated by the driver at the finish section of the DIS simulation course. .............................................................................................................................................. 328
Figure 5-62: Lateral lane and speed deviations at the finish line section of the DIS simulation course. .............................................................................................................................................. 328
Figure 5-63: Reference speed generated by the ANVID and the associated driver response throughout the finish section of the DIS simulation course. ................................................................. 329
Figure 5-64: Instantaneous speed error exhibited by the driver at the finish section of the DIS simulation course. .............................................................................................................................................. 329
Figure 5-65: Distance between the front of the Blind Driver Challenge® vehicle and the finish line stop point as a function of time at the finish section of the DIS simulation course. .............................................................................................................................................. 330
Figure 5-66: Reference steering wheel angle generated by the ANVID and the associated driver response at the finish course section. .............................................................................................................................................. 331
Figure A-1: Average driver response to a 30° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 359
Figure A-2: Average driver response to a 30° step reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 359
Figure A-3: Average driver response to a 30° step reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 360
Figure A-4: Average driver response to a 30° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 360
Figure A-5: Average driver response to a 60° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 361
Figure A-6: Average driver response to a 60° step reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 361
Figure A-7: Average driver response to a 60° step reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 362
Figure A-8: Average driver response to a 60° step reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 362
Figure A-9: Average driver response to a 180° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 363
Figure A-10: Average driver response to a 180° step reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 363
Figure A-11: Average driver response to a 180° step reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 364
Figure A-12: Average driver response to a 180° step reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2. .............................................................................................................................................. 364
Figure A-13: Average driver response to a 5°/s ramp reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .................................................. 365
Figure A-14: Average driver response to a 5°/s ramp reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.................................................. 365
Figure A-15: Average driver response to a 5°/s ramp reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2. .................................................. 366
Figure A-16: Average driver response to a 5°/s ramp reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2. .................................................. 366
Figure A-17: Average driver response to a 10°/s ramp reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .................................................. 367
Figure A-18: Average driver response to a 10°/s ramp reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2. .................................................. 367
Figure A-19: Average driver response to a 10°/s ramp reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2. .................................................. 368
Figure A-20: Average driver response to a 10°/s ramp reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2. .................................................. 368
Figure A-21: Average driver response to a 30°/s ramp reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2. .................................................. 369
Figure A-22: Average driver response to a 30°/s ramp reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2. .................................................. 369
Figure A-23: Average driver response to a 30°/s ramp reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2. .................................................. 370
Figure A-24: Average driver response to a 30°/s ramp reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2. .................................................. 370
Figure A-25: Average driver response to a 5m/s step reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7. .................................................. 371
Figure A-26: Average driver response to a 5m/s step reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7. .................................................. 371
Figure A-27: Average driver response to a 10m/s step reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7. .................................................. 372
Figure A-28: Average driver response to a 10m/s step reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7. .................................................. 372
Figure A-29: Average driver response to a 0.3m/s² ramp reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7. .................................................. 373
Figure A-30: Average driver response to a 0.3m/s² ramp reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7. .................................................. 373
Figure A-31: Average driver response to a 0.5m/s² ramp reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7. .................................................. 374
Figure A-32: Average driver response to a 0.5m/s² ramp reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7. .................................................. 374
List of Tables

Table 2-1: Vehicle Component Capabilities. All items listed can be sensed; items marked with an asterisk (*) can additionally be controlled .......................................................... 32
Table 2-2: Onboard Dual Computer System Specifications (Per Computer) .................. 45
Table 2-3: Report Global Pose JAUS Message. See [106] and [107] for complete message detail .......................................................... 48
Table 2-4: Report DUC Local Pose JAUS Message. See [106] for message detail ..... 48
Table 2-5: Report Wrench Effort JAUS Message. See [106] and [107] for complete message detail .......................................................... 50
Table 2-6: Report Velocity State JAUS Message. See [106] and [107] for complete message detail .......................................................... 51
Table 2-7: Set Motion Profile JAUS Message. See [106] for message detail ........... 54
Table 2-8: Operator Control Unit (OCU) Computer Specifications ...................... 56
Table 3-1: DriveGrip Vibro-Tactile Element Control Table for four finger and two finger vibration configurations .................................................. 72
Table 3-2: DriveGrip Operation Configurations (DOCs) ..................................... 84
Table 3-3: Transient characteristics of the driver responses plotted in Figure 3-14 .... 85
Table 3-4: Quantitative characteristics of the driver steering responses in Figure 3-18. 92
Table 3-5: Quantitative characteristics of the driver steering responses plotted in Figure 3-20 and Figure 3-21 .................................................. 97
Table 3-6: SpeedStrip Vibro-Tactile Element Control Table for four element and two element vibration configurations ........................................... 102
Table 3-7: SpeedStrip Operation Configurations (SOCs) .................................. 109
Table 3-8: Average transient characteristics for the FE and TE SOC induced responses shown in Figure 3-34 and Figure 3-35 ........................................... 111
Table 3-9: Average quantitative characteristics of the driver responses shown in Figure 3-36 and Figure 3-37 .................................................. 115
Table 3-10: Quantitative characteristics of the driver speed responses plotted in Figure 3-38 ............................................................................. 118
Table 3-11: Quantitative characteristics of the driver steering and speed responses plotted in Figure 3-44 .................................................. 123
Table 3-12: Non-Visual Interface Computer Specifications .................................. 132
Table 4-1: TORC Robotics ByWire XGV™ research platform steering properties ..... 138
Table 4-2: PNVID algorithm hierarchy. Each sub-algorithm is listed along with the corresponding section it is described within ........................................... 142
Table 4-3: Case 1 detection conditions and transformation action ....................... 150
Table 4-4: Case 2 detection conditions and transformation action ....................... 151
Table 4-5: Case 3 detection conditions and transformation action ....................... 152
Table 4-6: Case 4 detection conditions and transformation action ....................... 153
Table 4-7: PNVID Component custom VI hierarchy and section index ............... 164
Table 4-8: PNVID Component Configuration ................................................................. 170
Table 5-1: ANVID algorithm hierarchy. Each sub-algorithm is listed along with the corresponding section it is described within. ................................................................. 268
Table 5-2: ANVID software custom VI hierarchy and section index. ......................... 282
Table 5-3: Report Reference Steering experimental JAUS message. ......................... 286
Table 5-4: Report Reference Speed experimental JAUS message. ......................... 286
Table 5-5: MPC Component Configuration ................................................................. 291
Table 5-6: Driver Modeling Component Configuration .................................................. 293
Table 5-7: Average processing times of Driver Modeling algorithms. ....................... 295
Table 5-8: Average processing times of Model Predictive Control algorithms.......... 300
Table 5-9: QNO() iteration count statistics. ................................................................. 301
## Nomenclature

### Non-Visual Interfaces

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_R$</td>
<td>Instantaneous Reference Steering Wheel Angle</td>
</tr>
<tr>
<td>$\theta_A$</td>
<td>Instantaneous Actual Steering Wheel Angle</td>
</tr>
<tr>
<td>$\theta_E$</td>
<td>Instantaneous Steering Wheel Angle Error</td>
</tr>
<tr>
<td>$S_R$</td>
<td>Instantaneous Reference Speed</td>
</tr>
<tr>
<td>$S_A$</td>
<td>Instantaneous Actual Speed</td>
</tr>
<tr>
<td>$S_E$</td>
<td>Instantaneous Speed Error</td>
</tr>
<tr>
<td>$T_I$</td>
<td>Index Finger Steering Wheel Angle Error Threshold</td>
</tr>
<tr>
<td>$T_M$</td>
<td>Middle Finger Steering Wheel Angle Error Threshold</td>
</tr>
<tr>
<td>$T_R$</td>
<td>Ring Finger Steering Wheel Angle Error Threshold</td>
</tr>
<tr>
<td>$T_L$</td>
<td>Little Finger Steering Wheel Angle Error Threshold</td>
</tr>
<tr>
<td>$T_{L4}$</td>
<td>Knee Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{L3}$</td>
<td>Lower Thigh Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{L2}$</td>
<td>Upper Thigh Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{L1}$</td>
<td>Buttocks Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{B1}$</td>
<td>Lower Back Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{B2}$</td>
<td>Middle Back Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{B3}$</td>
<td>Upper Back Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{B4}$</td>
<td>Shoulder Speed Error Threshold</td>
</tr>
<tr>
<td>$T_{STOP}$</td>
<td>Full Stop Speed Threshold</td>
</tr>
</tbody>
</table>

### Vehicle

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta$</td>
<td>Steering Wheel Angle</td>
</tr>
<tr>
<td>$C$</td>
<td>Curvature</td>
</tr>
<tr>
<td>$v$</td>
<td>Velocity</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>Heading</td>
</tr>
<tr>
<td>$x$</td>
<td>Lateral Position in the Local Frame</td>
</tr>
<tr>
<td>$y$</td>
<td>Longitudinal Position in the Local Frame</td>
</tr>
<tr>
<td>$L$</td>
<td>Wheelbase</td>
</tr>
<tr>
<td>$R$</td>
<td>Steering Ratio</td>
</tr>
<tr>
<td>$k$</td>
<td>Understeer Coefficient</td>
</tr>
<tr>
<td>$P$</td>
<td>Perpendicular Lane Vector</td>
</tr>
<tr>
<td>$R$</td>
<td>Vehicle-Lane Vector</td>
</tr>
<tr>
<td>$D$</td>
<td>Lateral Lane Deviation</td>
</tr>
</tbody>
</table>
### Driver Modeling

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{\theta}_a )</td>
<td>Predicted Actual Steering Wheel Angle</td>
</tr>
<tr>
<td>( \hat{\mathbf{s}}_a )</td>
<td>Predicted Actual Speed</td>
</tr>
<tr>
<td>( y_n )</td>
<td>Neuron Output</td>
</tr>
<tr>
<td>( u_n )</td>
<td>Neuron Input</td>
</tr>
<tr>
<td>( \mathbf{u} )</td>
<td>Neuron Input Vector</td>
</tr>
<tr>
<td>( w )</td>
<td>Neuron Weight</td>
</tr>
<tr>
<td>( \mathbf{w} )</td>
<td>Neuron Weight Vector</td>
</tr>
<tr>
<td>( f(\cdot) )</td>
<td>Neuron Activation Function</td>
</tr>
<tr>
<td>( e_b )</td>
<td>Backpropagated Neuron Error</td>
</tr>
<tr>
<td>( d_n )</td>
<td>Desired Neuron Output</td>
</tr>
<tr>
<td>( \mathbf{J}_L )</td>
<td>Local Layer Jacobian (w.r.t. Layer Inputs)</td>
</tr>
<tr>
<td>( \mathbf{J}_L )</td>
<td>Global Layer Jacobian (w.r.t. Network Inputs)</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>Perturbation Vector</td>
</tr>
<tr>
<td>( \mathbf{\hat{w}} )</td>
<td>Estimated Neuron Weight Vector</td>
</tr>
<tr>
<td>( \mathbf{P} )</td>
<td>Inverse Covariance Matrix</td>
</tr>
<tr>
<td>( \mathbf{\hat{P}} )</td>
<td>Estimated Inverse Covariance Matrix</td>
</tr>
<tr>
<td>( K )</td>
<td>Kalman Gain</td>
</tr>
<tr>
<td>( q )</td>
<td>Weight Vector Jacobian</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>Extended Kalman Algorithm Forgetting Factor</td>
</tr>
</tbody>
</table>

### Model Predictive Control and Optimization

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c )</td>
<td>Commanded Action Signal</td>
</tr>
<tr>
<td>( r )</td>
<td>Reference Action Signal</td>
</tr>
<tr>
<td>( a )</td>
<td>Past Driver Action Signal</td>
</tr>
<tr>
<td>( \hat{a} )</td>
<td>Predicted Driver Action Signal</td>
</tr>
<tr>
<td>( \mathbf{M}(\cdot) )</td>
<td>Driver Model</td>
</tr>
<tr>
<td>( O(\cdot) )</td>
<td>Objective Function</td>
</tr>
<tr>
<td>( \Delta r )</td>
<td>Reference Optimization Increment</td>
</tr>
<tr>
<td>( s )</td>
<td>Optimization Step Size</td>
</tr>
<tr>
<td>( \mathbf{G} )</td>
<td>Gradient of Objective Function</td>
</tr>
<tr>
<td>( \mathbf{\hat{H}}^{-1} )</td>
<td>Estimated Hessian Matrix of Objective Function</td>
</tr>
<tr>
<td>( \mathbf{D} )</td>
<td>Descent Direction Vector</td>
</tr>
<tr>
<td>( c_1, c_2 )</td>
<td>Step Size Condition Parameters</td>
</tr>
</tbody>
</table>
Section 1: Introduction

Driver assistance technologies provide humans with a wide range of hardware and software that aid in the operation of an automobile in a more safe and efficient manner. Ever since the introduction of the mass-production automobile to the public in the early 1900s, researchers have been pursuing ways to assist the driver in everyday, dangerous, and even emergency driving situations. As the capabilities of these technologies increase, the human-vehicle system is streamlined; causing fewer accidents, increasing the efficiency of road traffic, and improving the overall vehicle operation experience itself.

The focus of this paper is to present robust and intuitive driver assistance technologies that enable fully blind individuals to safely and independently operate vehicles on standard public roads. This is achievable through the use of specialized hardware and software that can essentially “see” the surrounding environment for the driver and provide necessary information to them in the form of human-computer interface (HCI) commands. This technology can also benefit the sighted drivers of today in that it has the ability to augment their vision with suggestive cues during normal and low-visibility conditions, or even replace their vision in situations with no visibility.

The research and development of these technologies have all been made possible through the Blind Driver Challenge® (BDC): a collaborative research effort between the Robotics and Mechanisms Laboratory (RoMeLa) at Virginia Tech and the National Federation of the Blind (NFB). The proceeding text of this introductory chapter will provide a comprehensive definition of the Blind Driver Challenge® (Section 1.1) as well as summarize the history of Virginia Tech’s involvement in the effort since 2005 (Section 1.2).

A complete characterization of the problems behind developing technologies for the Blind Driver Challenge® is provided in Section 1.3 of this paper. The statement of the problem subsequently leads into a comprehensive review of literature within several fields of research that attempt to solve different aspects of the problems at hand (Section 1.4). After examination of the cutting-edge technologies presented in recent literature, Section 1.5 clearly states what the contributions of this paper are and how it will advance the state-of-the-art. This chapter finally concludes with a brief outline of the remaining chapters of this work in Section 1.6.
1.1: The Blind Driver Challenge® by the National Federation of the Blind

The Blind Driver Challenge® was initiated by the National Federation of the Blind on January 30th, 2004. The proposal came directly from the center of blind technology research: the NFB Jernigan Institute. The Jernigan Institute focuses on “understanding the real problems of blindness to develop innovative education, technologies, products and services that help the world’s blind to achieve independence [1].” According to the NFB, the lack of the ability to drive is one of the most influential factors in a blind person’s loss of independence. Without being able to drive, a blind person loses the capability to easily travel from one place to another. This means that a blind person cannot drive to the grocery store, a friend’s place, or even work. The inability to drive to work is a major factor in the high unemployment rates amongst the blind community.

The Blind Driver Challenge® was proposed by the NFB to entice universities and companies to provide a solution to the loss of independence inherited from the inability to drive. The challenge not only plans to solve the issue of blind driving, but to also stimulate development of other technologies as well as demonstrate the capacity of the blind to the public. According to the official BDC proposal by the NFB, the challenge has four main goals [2]:

1. To establish a path of technological advancement for nonvisual access technology, and close the gap between access technology and general technology.

2. To demonstrate that vision is not a requirement for success and that the application of innovative nonvisual solutions to difficult problems can create new opportunities for hundreds of thousands of people – blind and sighted.

3. To increase awareness among the university scientific community about the “real problems” facing the blind by providing expertise from the perspective of the blind within the context of a difficult engineering challenge.

4. To change the public perceptions about the blind by creating opportunities for the public to view blind people as individuals with capacity, ambition, and a drive for greater independence.

The first two goals of the Blind Driver Challenge® aim to stimulate research and development in many types of non-visual technologies that will benefit blind and sighted people alike. Although the technologies are meant for non-visual operation of an automobile, the fundamentals of these technologies can be applied to many other real-world problems. For example, if a blind person can use BDC technologies to drive a vehicle, the same technologies could be easily adapted to allow blind persons to walk around freely without the use of a cane or a guide. A very applicable problem where non-visual technologies could assist sighted persons in is low-visibility situations, such as driving or flying in the night, rain, fog or snow storms. While the sighted driver has
complete vision, they may still have trouble seeing the road or obstacles ahead due to darkness, obstructions, etc. Non-visual interfaces would not only be able to alert the driver that they are heading toward an obstacle or road edge, but also help them properly navigate back to safety.

While most goals of the challenge focus on non-visual technologies themselves, the third goal strives to improve the process in which these technologies are developed. In the past, many types of devices for the blind have been designed without much perspective from the blind themselves. It is not a simple task for the sighted to create products for the blind; closing one’s eyes is not the same as being blind. The blind learn many intricate ways to use their other senses to fill in the gap created by lack of sight. These skills are not easily understood by those who are not blind themselves, creating a lack of true functionality in devices made for the blind without actual input from the blind. This goal of the challenge simply seeks to facilitate healthy communication between the designer and users of blind technologies. With this type of high efficiency development, blind technologies will become more functional and useful to the market it is meant for.

The final goal of the Blind Driver Challenge® aims to demonstrate to the general public that blind people have capacity and are equally capable with sighted people. By proving that a blind person can independently operate a vehicle, the NFB hopes to not only improve the public opinion of the blind, but to also give blind people everywhere hope that they will have the same opportunities as sighted people do.

In summary, the National Federation of the Blind is planning to use this research endeavor to usher in a new era of non-visual technology development and improve the public perception of the blind. The technologies developed in the Blind Driver Challenge® will have the ability to benefit all drivers, blind and sighted. They will also act as building blocks that can be improved upon to develop new technologies that can assist people in many other ways: from how the blind read a graph to assisting sighted pilots navigate through low visibility situations.
1.2: The Virginia Tech Research Effort
Virginia Tech became the first entity to accept the Blind Driver Challenge® in 2005, and has remained the only contributing source of research to date. Since then, specialized teams composed of graduate and undergraduate students have continuously put forth tremendous effort to solve the complex task of putting a blind person safely behind the wheel of an automobile. This subsection documents the history of these efforts as well as the most recent 2010-2011 research goals that were achieved in this work.

1.2.1: History
The history of the Virginia Tech effort in the Blind Driver Challenge® can be conceptually split into three time periods: the development of a research vehicle base platform, the development of non-visual interfaces, and the implementation of a small-scale vehicle. The contributions from each of these time periods are summarized in the following sub-sections.

When Virginia Tech first accepted the Blind Driver Challenge®, the primary aim was to create a research vehicle base platform that could be used by future entities that would participate in the challenge [3]. The platform was essentially an extension of a previous Virginia Tech project, called the High-Speed Autonomous Vehicle [4], which was repurposed for blind driving. The vehicle consisted of a 2004 Cadillac SRX modified for drive-by-wire operation and sensor mounting. The drive-by-wire capabilities allowed for remote control of the vehicle steering/throttle/brake, and also included a pneumatic emergency braking system to serve as an emergency stop function. GPS and LIDAR systems were incorporated on the vehicle for waypoint navigation and real-time obstacle avoidance using an artificial potential field motion planning algorithm. All software in use on the vehicle was interoperable with the Joint Architecture for Unmanned Systems (JAUS) messaging protocol for higher efficiency and compatibility. The ultimate goal of these technologies were to produce a semi-autonomous vehicle that would perform all low-level driving tasks (steering, waypoint navigation, obstacle avoidance, etc.) while allowing a blind operator to perform the high-level tasks (specifying a destination, operation of AC and radio controls, etc.).

Although the original plan was to develop an information package based on the developed platform and deliver it to the National Federation of the Blind for use in the Blind Driver Challenge®, the team of students working on the project realized the potential of the platform’s technology for use in the 2007 DARPA Urban Challenge (DUC). Consequently, the team migrated all of their efforts towards the DUC (eventually placing third place) and all work related to the Blind Driver Challenge® was postponed.

After completing the 2007 DARPA Urban Challenge, the Blind Driver Challenge® was resumed through the Robotics and Mechanisms Laboratory (RoMeLa), led by Dr. Dennis Hong. Feedback from the National Federation of the Blind indicated that they were not interested in an autonomous vehicle, such as the one proposed by the previous 2005-2006 team, but wanted to actually be able to drive the vehicle itself. With this in mind, the new team took a much different approach in the development of blind driver technology: developing non-visual human-computer interfaces that provided driving information to the human, allowing them to independently operate the vehicle [5].

The RoMeLa team developed two separate non-visual interfaces: a vibro-tactile chair for speed information and a headphone set for auditory steering information [6]. The chair was outfitted with 6 vibration motors up the seatback and 4 rows of 2 vibration motors on the seat area. The position of vibrations on the chair indicated the need to accelerate or decelerate based on the current and desired speeds of the vehicle. The vibro-tactile chair was configured in such a way that many different vibratory patterns could be experimented with, such as using seat vibrations for deceleration and back vibrations for acceleration, or vice-versa.

The headphone-based non-visual interface took advantage of stereo sound patterns to help the driver steer the vehicle. Tones heard in the left or right ear indicated to turn in that respective direction. The tone frequency was adjusted in real time to give the driver an idea of the amount of turn required based on the current and desired steering angle. Similarly to the vibro-tactile chair, the system was highly configurable in terms of auditory patterns used to communicate information.

Although the non-visual interfaces developed by the RoMeLa team allowed a blind person to fully drive a vehicle, the interfaces themselves still required knowledge of a reference path for desired steering and speed information. For this reason, much of the technology from the previously developed vehicle base platform was still used; the autonomy capabilities of the platform provided all of the perception and motion planning information required by the non-visual interfaces. However, the physical research platform itself was unavailable for blind driver testing; therefore a comprehensive software-simulation of the platform was used for experimentation. Through these investigations, the team concluded that the non-visual interfaces independently work well, however combination of the interfaces caused driver mental overload and resulted in poor performance.
1.2.1.3: (2008-2010) Small Scale Vehicle Implementation

Over the next two years, RoMeLa concentrated its efforts on further developing the non-visual interfaces and implementing them on a new, small-scale vehicle platform [7] [8]. The new platform consisted of a dune buggy outfitted with a small sensor array for vehicle state identification and environmental perception. The sensor array included a front mounted LIDAR for obstacle detection, a Hall Effect sensor for speed identification, and a string potentiometer for steering wheel encoding. An onboard real-time/FPGA computer allowed simultaneous perception and non-visual interface control. Similarly to the original full-size vehicle platform developed in 2005, this implementation used potential field theory for motion planning.

Throughout the development of the small-scale vehicle, new and improved non-visual interfaces were created to meet the design requirements posed by the new vehicle and to improve blind driver performance. During this phase, several new types of non-visual interfaces were examined: a vibro-tactile vest, a click-wheel steering column, a refreshable tactile display matrix, vibro-tactile gloves, vibro-tactile leg straps and a vibro-tactile shoe. The most notable interface prototypes were the vibro-tactile vest, gloves, shoe, and refreshable display matrix.

Although the original idea was to implement the vibro-tactile chair developed during the 2007-2008 period, the team discovered that vibrations in the chair were completely obscured by strong engine vibrations from the small-scale vehicle. The vibro-tactile vest was subsequently used to alleviate this problem; vibrations on the front of the torso were much easier to feel even with engine vibrations present. The vest was only used to communicate the need to decelerate the vehicle. Later, vibro-tactile leg straps were added to the system to give comprehensive speed information: vibrations in the leg indicated the need to accelerate while the torso indicated to decelerate. The team ultimately combined these two speed-informative interfaces into one with the vibro-tactile shoe. The shoe consisted of vibration motors on the toe which indicated the magnitude of the desired speed change and additional vibration motors on the left/right sides of the foot which communicated if the vehicle needed to accelerate or decelerate.

The headphone-based auditory interface from the 2007-2008 period was modified to work with a click-wheel mounted on the vehicle steering column. The vehicle computer would communicate the necessary change in steering wheel position to safely maneuver the vehicle and communicate, through speech, how many “clicks” to turn the wheel and in what direction. Searching for a faster and less fatiguing non-visual interface, the team then developed a vibro-tactile set of gloves named DriveGrip. These gloves had a vibration motor on the index, middle, ring, and pinky finger of each hand. The location of the vibration indicated the necessary steering angle correction magnitude and direction: the left and right hands represented turn left and right,
respectively, while the range of pointer to pinky finger represented minimum and maximum magnitude.

The final non-visual interface proposed by the 2008-2010 teams was AirPix, an air-based refreshable tactile display matrix. The purpose of this interface was to display 2-dimensional mappings of driving and environmental information to the blind operator. Shapes, terrains, and even writing could be felt through AirPix by placing the hand over a grid of orifices releasing pressurized air through solenoid valves. The prototype included 135 orifices in a 4.9”x2.8” rectangular area, providing a density of 10 tactors/in². An in-depth analysis was performed on the AirPix interface, including mapping the sensitivity of the hand as well as discovering human reaction time to the display. However, initial findings listed unsatisfactory 2-5 second reaction times to successfully understand simple display information.

The final implementation of the small-size vehicle platform included the new vibro-tactile shoes and DriveGrip. At that point, AirPix was still in the very early prototype stage and was not capable of being mounted to the vehicle. With the cooperation of the National Federation of the Blind, the small-size vehicle implementation was tested extensively for feedback and improvement of the vehicle and non-visual interfaces. Similarly to the testing conducted in the 2007-2008 non-visual interface development stage, the primary conclusion made was that the interfaces independently work well, but cause severe mental overload and poor performance when used in combination.

1.2.2: The 2010-2011 Research Proposal

At the start of summer 2010, the RoMeLa BDC team was ready to take on the next step in putting a blind person behind the wheel: upgrading the BDC vehicle to a full size SUV. With full sponsorship from the National Federation of the Blind, the team planned to build two BDC SUV’s and provide a comprehensive blind driver demonstration to the public at the Daytona International Raceway. The two vehicles, named BRIAN (Blind Research Interfaces for Advanced Navigation) and ANDREA (Automobile for Non-visual Driving, Research, Education, and Advancement), would be outfitted with all the necessary non-visual interfaces and perception sensors required for blind driving.

The development of the new Blind Driver Challenge® vehicles, BRIAN and ANDREA, would be undertaken by a team of 2 full-time graduate students as well as 12 undergraduate students ranging from the sophomore to senior level. Under the advisory of Principle Investigator Dr. Dennis Hong, the team would continue development of new and past non-visual interfaces and integrate them with a full-size research platform developed through a subcontract with TORC Technologies.
1.2.2.1: Non-Visual Interface Development

The next generation vehicles planned on using three existing interfaces for communicating information to the blind: DriveGrip, the Tactile Chair, and AirPix. DriveGrip and the Tactile Chair would act as the primary steering and speed information provider, while AirPix would inform the driver of his surroundings for conscious driving decisions.

DriveGrip, having been designed by the previous year’s team, would make a full transition from a simple prototype to a robust and usable interface. Extensive improvements on the glove design would be made, including the addition of seamless vehicle integration for increased performance and efficiency. The software algorithms controlling the DriveGrip interface would also be re-invented and improved upon through extended testing with blind participants. With the combination of advanced glove design and new software, DriveGrip would become a dependable source of steering information to the driver.

The Tactile Chair would make a comeback as the newly named interface: SpeedStrip. Although SpeedStrip is functionally the same as the Tactile Chair, improvements on the vibration motor placement as well as redesign of control algorithm software would be made. This interface would also be seamlessly integrated with the vehicle to enhance the performance and comfort of the driver. Similarly to DriveGrip, substantial amounts of blind driver testing would be conducted to fine-tune SpeedStrip.

AirPix would also undergo a full transformation from the prototype phase to a complete interface that could be integrated inside the vehicle. AirPix would cooperate with the vehicle research platform’s perception suite to inform the driver of many types of information, including (but not limited to) lane position and obstacle identification/location. With this type of information literally at the blind driver’s fingertips, AirPix would provide a means of independence to the driver.

Although DriveGrip, SpeedStrip, and AirPix were the primary non-visual interfaces under development at the time, the proposal also included the possibility of new interface development. These new interfaces could be used to supplement and/or replace one or more of the existing interfaces. This left the floor open for new ideas that would come hand-in-hand with the fresh minds of that year’s new team members.
1.2.2.2: Vehicle Research Platform

The next generation vehicles, BRIAN and ANDREA, required a dependable, easy-to-integrate, and safe research platform. The RoMeLa BDC team decided that they would subcontract TORC Technologies to develop the new research platform. TORC Technologies is actually a company that was created by the same Virginia Tech students who participated in early stages of BDC as well as the DARPA Urban Challenge. By subcontracting TORC, the team would be able to use a company with previous experience in the Blind Driver Challenge® to develop the base research platform. TORC would be responsible for providing RoMeLa with a closed-package base research platform; meaning that construction, maintenance, and troubleshooting of the platform would be the sole responsibility of TORC. This allowed RoMeLa to focus more on the development and integration of the non-visual interfaces and associated control algorithms, which account for the real research purpose of the Blind Driver Challenge®.

The base research platform developed by TORC would ultimately be a complete drive-by-wire system implemented on a 2010 Ford Escape Hybrid SUV. This platform would allow full operation of the vehicle via remote control, full sensor access to the vehicle state, and provide a safety system using onboard and wireless emergency stops. The control and access of these features would be centralized on an onboard vehicle computer and exposed through JAUS (Joint Architecture for Unmanned Systems), a messaging protocol standardized by the US Department of Defense [9].

Once the base research platform was delivered, RoMeLa would work with TORC to add a sensor array, computer systems, electrical power systems, and other supporting hardware to the vehicle. The sensor array would incorporate many types of sensors, ranging from LIDAR to Stereo Vision Cameras. These sensors would be used to perceive the vehicle environment for the blind driver. Additional computer systems would be added to interface with the sensors, connect with the drive-by-wire system, and control the non-visual interfaces. Once the appropriate hardware was decided upon, TORC would install them on the base platform while maintaining the look of a stock 2011 Ford Escape Hybrid. At that point, the fully customized research platform would be delivered to RoMeLa for non-visual interface integration and ultimately Blind Driver Challenge® research and development.
1.3: Problem Statement

The primary goal of this paper’s research effort is to coordinate with the Blind Driver Challenge® and develop non-visual driver assistance technologies that can aid a human driver in controlling the heading and speed of their vehicle in scenarios with little or no visibility. The hardware and software encapsulated in this technology can be used by today’s drivers to replace or augment the use of vision for vehicle navigation in situations such as dense fog, darkness, or heavy rain. This non-visual technology can also be more specifically used and extended upon to one day enable blind individuals to independently operate an automobile (Section 1.1).

The technological requirements for an instructional solution to the Blind Driver Challenge® are illustrated in Figure 1-1. The process begins with the driving Environment that the vehicle operates within. The Environment includes roads that the driver must navigate and obstacles that the driver must avoid collisions with. To enable the driver to perform this task under blindness or low-visibility conditions, the system must first be able to use Perception technologies to sense the Environment for the driver. Next, Motion Planning algorithms must utilize the Perception data to generate safe trajectories through the Environment with minimal risk of collisions. Unfortunately, the trajectories calculated by Motion Planning cannot be directly understood by a human driver. Therefore, it is necessary for Reference Transformation algorithms to transform the planned trajectories into a form that can be understood and reliably interacted with by the human driver. The transformed information is then given to the driver through non-visual Communication methods so that he or she may finally respond to the transformed information with the correct Driver Actions. These actions change the vehicle’s position in the Environment and, in the time taken for the entire process to finish, the Environment may have changed. Therefore, the process repeats itself indefinitely so that the trajectories generated by Motion Planning remain current.
The technological requirements behind the instructional solution to the Blind Driver Challenge™ shown in Figure 1-1 can therefore be summarized as:

1. **Perception**: Sense the environment for the driver in full, partial, or no visibility driving scenarios with accuracy and robustness.
2. **Motion Planning**: Synthesize safe navigation information based on examination of the acquired perception data.
3. **Reference Transformation**: Transform the synthesized navigation information into non-visual stimuli that can be understood and reliably interacted with by a human driver.
4. **Communication**: Non-visualy communicate the transformed navigation information to the human driver in a dependable manner.

As indicated in Figure 1-1, autonomous vehicle technologies currently exist that can, to a certain degree, already solve the first and second problems of the Blind Driver Challenge®. One such example is the TORC Robotics ByWire XGV™ drive-by-wire vehicle supplemented with the AutonoNav™ autonomous navigation system [10]. This platform, when outfitted with the necessary sensors, is capable of perceiving the immediate driving environment and planning safe motions through the environment to reach a prescribed destination.

As the technology that solves the first two problems behind the Blind Driver Challenge® already exists, this work will utilize the capabilities of the TORC Robotics ByWire XGV™ and focus on solving the final two problems of the challenge: **Reference Transformation** and **Communication**. In order to successfully solve these remaining issues, transformation algorithms as well as human-computer interfaces (HCIs) must be developed. An additional consideration worth noting is that the transformation algorithms can be either adaptive or passive in nature. Passive algorithms strictly transform motion planning instructions into HCI signals and behave identically between different human drivers. Adaptive transformation algorithms differ in that they additionally create and use driver models to adapt to each particular driver’s tendencies and significantly increase the performance of the system.

With these considerations in mind, the third problem of the Blind Driver Challenge® can be described through the intersections of three main fields of research: Motion Planning, Human-Computer Interfaces, and Driver Modeling. Figure 1-2 portrays this relationship and how adaptive and passive transformation algorithms interplay with these diverse research fields. Fortunately, the fourth and final problem presented by the challenge is not as complex and can be isolated to the HCI research field alone.
Figure 1.2: The relationship between motion planning, driver modeling, and human-computer interface research areas in adaptive and passive transformation algorithms.
1.4: Review of Literature

As outlined in the Problem Statement of this work (Section 1.3), the contributions of this paper directly rely on the well-established research fields of driver modeling, motion planning, and human-computer interfaces. These topics have independently come under tremendous amounts of research due to their popularity in car performance testing, robot-environment navigation, and the participation of humans within a system loop. An additional and equally important field of research on Model Predictive Control (MPC) also exists and focuses on adapting to and interacting with nonlinear, complex dynamical systems. As a human being can be readily described as such a system, practices from MPC can be used to formulate a solution to the adaptive driver assistance problem. This section serves the purpose of summarizing and discussing the most relevant and notable contributions that have brought each of these important research fields to the cutting-edge of science.

1.4.1: Driver Modeling

Modeling human drivers has long been a popular research topic amongst the military, automobile companies and universities. Although the first models of humans as operators of vehicles appeared as pilot models for military research in airplanes and helicopters, interest in modeling humans as drivers soon spread with the rapid increase of automobile production in the 1950s and 1960s. Since then, many different attempts have been made to try and mathematically recreate human driving behavior, the most prominent of which are the Crossover Model, Optimal Control, Autoregressive Methods, and Artificial Neural Networks (ANNs). The progress made in each of these modeling methods is discussed in detail throughout the following subsections.

1.4.1.1: The Crossover Model

In 1968, McRuer and Weir presented a control theoretic crossover model for human drivers in a linearized compensatory task [11]. The single-loop model incorporated driver-vehicle dynamics in a lane-keeping scenario and characterized driver parameters by examining the open-loop frequency response around the gain crossover frequency found from experimental human driving data. However, this particular implementation used lane lateral position error as its only form of closed-loop feedback and consequently failed to work well for any driving scenario other than driving down a straight road. In [12] [13] [14] [15] [16], McRuer et al. improved upon their original crossover model through consideration of other types of driver feedback and ultimately the inclusion of multiple feedback terms in a dual-loop structure. This multi-loop structure consisted of an inner loop utilizing lateral lane position error feedback and an outer loop incorporating additional types of feedback (lateral reference position preview, heading angle, etc.) for improved lane tracking performance.
Hess continued work on a variation of the dual-loop human operator crossover model in 1978 [17], making a contribution of incorporating neuromuscular feedback derived from output rates on the human’s physical vehicle control interface to compensate for human adaptivity. In [18], Hess implemented the crossover model in a human pilot pursuit scenario and performed a detailed investigation on model inaccuracies from continuous adaptions of the pilot’s internal model of the vehicle dynamics. In order to account for time delays in the model response, Hess finally included tracking reference preview feedback within the dual loop structure [19]. Starting in 1989, Hess and Modjtahedzadeh worked together to adapt the human pilot crossover model for human driver steering behavior in curved lanes and lane changing [20] [21]. Here they used established multivariable feedback design techniques to outline model parameter tuning in the frequency domain, similarly to McRuer and Weir in [11]. However, Hess and Modjtahedzadeh’s contribution focused on the high and low frequency characteristics around the human gain crossover frequency and described their inherent relationship with the neuromuscular dynamics of the driver. In [22], the authors implemented their model on a fixed-based driving simulator developed by Reid et al. [23] with satisfactory human output reproduction and concluded their work with a presentation of how their model could be used for driver model-based handling quality analysis methodized within [24].

1.4.1.2: The Optimal Control Model

The notion of modeling a human operator as an optimal controller was first presented by Sheridan in 1966 [25]. However, in 1970, Baron, Kleinman and Levison presented an actual application of optimization theory to human operator modeling in linear control scenarios [26]. Based on the assumption that a human behaves optimally, they constructed a human state-space model consisting of a pure time delay, an optimized equalization network, and neuromuscular dynamics. The internal equalization network was optimized through an objective function based on the minimization of vehicle state error, control signal magnitude and control signal time-rate-of-change. Baron et al. also incorporated a Kalman estimator to simulate the human’s perception of system states from supplied inputs, while using a least mean-squared predictor to model human compensation for inherent time delays. Parameters internal to the dynamics model of the human and controlled system, such as time lag and neuromuscular constants, were taken from previously published experimental data. Results indicated accurate reproductions of human response through the model for 1-dimensional tracking in position, velocity and acceleration; however tuning of the optimization objective function in regard to the control variables proved difficult due to neuromuscular lag. In [27], Baron et al. successfully employed the optimal control human model on a Vertical Take-Off and Landing (VTOL) aircraft for vertical flight altitude control only. However, the authors concluded this work with a need for a systematic method of parameter estimation to more accurately model the human-machine system dynamics.
In 1980, MacAdam contributed an optimal control, linear time-invariant human model for tracking tasks based on reference preview [28]. This work used the same principals as Baron et al. [26]; however the optimization scheme estimated human control outputs through minimization of discrete preview output mean squared errors over the preview space. The optimization ultimately calculated a control output that would minimize the difference between future reference values and estimated future actual values for a predefined space ahead. MacAdam extended this work by later comparing his model to an actual driver in a simple lane changing task [29]. However, in this implementation MacAdam only used a single discrete preview point “for reasons of clarity and notational simplicity” which inadvertently undermined the potential of his model. Although he demonstrated very reasonable results in simulated versus measured driver steering angle, yaw rate and lateral acceleration for a lane change, the use of more preview points in the preview space would have seemed more valid.

Sharp, Casanova and Symonds built on the linear optimal discrete preview model using saturation functions to account for nonlinearity in tire dynamics during performance driving and applied it to racing vehicles [30]. In this work, Sharp et al. stress the importance for multi-point preview for successful human modeling. As within previously mentioned works, model parameters had to be manually set, and in this particular case human driver parameters were chosen based on intuition. However, Sharp et al. formed a comprehensive model of the vehicle dynamics, including aerodynamics and rotational inertial of the axels, etc. The authors tested their model with front and rear wheel drive vehicles on a complex racing course against human driver experimental data and obtained impressive results considering the complicated dynamics at racing speeds.

In 2001, Prokop presented a nonlinear optimal discrete preview model that experimented with a large array of nonlinear objective functions to plan future trajectories with an inner PID control loop to track said trajectories [31]. The nonlinear objective functions tested included maximization of travel distance, minimization of horizontal accelerations, minimization of braking, etc. It also incorporated the use of several constraints, ranging from the limitation of engine RPM to requiring the ability to stop the vehicle within half the distance of the preview sight for safety measures. A Sequential Quadratic Programming (SQP) algorithm was used to optimize future trajectories in real time, while an inner PID loop controlled throttle, brake, and steering angles to track the optimized trajectories. Prokop simulated human response through his model in a double lane change maneuver at speeds from 70-120km/h with satisfactory results. However, Prokop concluded that actual human drivers employ a large set of objectives and constraints simultaneously in their optimization scheme, whereas his model only optimizes one objective at a time.
1.4.1.3: Autoregressive Methods

Pilutti and Ulsoy gave a preliminary study on the use of Auto-Regression with Exogenous inputs (ARX) models to describe human driver steering behavior based on vehicle lateral position in 1995 [32]. The authors proposed a closed-loop system in which driver model parameters are estimated online through recursive least squares algorithms. In 1999 [33], Pilutti and Ulsoy extended their work with a more complete analysis on the ARX human driver model, including investigation of how unavailability of excitations in human driver data effected model identification. After testing the linear ARX model on a Ford® developed driving simulator with human drivers, Pilutti and Ulsoy found through spectral analysis that the model lacked plasticity due to its linearity.

In 2000, Ulsoy attempted to supplement the linear ARX model with a nonlinear function to account for driver nonlinearities such as complacency and the inability to perceive small errors in vehicle lateral position [34]. Ulsoy experimented with several different nonlinear model structures, all using a Nonlinear Auto-Regressive Moving Average with Exogenous inputs (NARMAX) model to estimate parameters within the structures. He concluded that the addition of nonlinear functions does compensate for driver nonlinearities, however did not reduce model uncertainty and thus did not improve model plasticity. Ulsoy stated that the original linear ARX model remained the best choice for use in the driver model for the sake of simplicity. In 2001, Chen and Ulsoy [35] moved to a linear Auto-Regressive Moving Average with Exogenous inputs (ARMAX) model after an analysis on model residuals. However, the main contribution of this work was the simultaneous online modeling of driver uncertainty using a linear ARX model. Chen and Ulsoy concluded that the majority of model uncertainty comes from the variability of the human driver’s steering behavior.

In order to combat the variability of human drivers, Sekizawa, Inagaki, Suzuki et al. developed a Piece-Wise affine ARX (PWARX) model to recognize different modes in driver behavior and maintain a separate ARX model for each mode [36] [37]. The model, described as a natural extension of the Hidden Markov Model (HMM) uses an Expectation-Maximization (EM) algorithm for online parameter estimation. Shortly after, in 2010 Mikami, Okuda, Taguchi et al. modified the PWARX model into a Probability-weighted ARX (PrARX) model to make mode transitions continuous so more efficient gradient-based optimization techniques could be used for parameter estimation [38]. These mode-based ARX models, although still linear, showed increased model performance as the nonlinearity described by Chen and Ulsoy [35] was effectively decomposed into smaller, linear subsections.
1.4.1.4: Artificial Neural Networks

In 1990, Kraiss and Kuttelwesch presented a radical paradigm shift in the development of human driver models. In [39], Kraiss et al. proposed the use of Artificial Neural Networks (ANNs) to model human strategies for obstacle avoidance in the operation of a vehicle. Their model used a Multi-Layered Perceptron (MLP) network for path optimization and a Functional Link Network (FLN) for vehicle control to significantly decrease network training time through the standard backpropagation method. As a feasibility test, the authors demonstrated the capabilities of the ANN model in an elementary simulation of a vehicle traveling through a maze with inputs from a computer keyboard. A human "driver" navigated several obstacle courses while the network continuously trained on human control signals. After sufficient training, the neural network model could independently emulate the human teacher behavior and navigate obstacle courses excluded from the training set.

In 1993, Neusser, Nijhuis, Spaanenbungr et al. made a much more impressive implementation of ANN human driver modeling. In their work [40], Neusser et al. employed the use of the ANN to comprehensively model human-vehicle dynamics and control in lateral position tracking without requiring knowledge of the underlying physical/cognitive processes. Using experimental human driver data and a standard backpropagation trainer, Neusser et al. trained a three layer, 21-neuron network to reproduce human steering behavior in highway driving with better control performance than conventional controllers. Although the neural network training was computationally intensive, it was shown that the network could learn the total closed-loop behavior of the complex human-vehicle system without provision of a model or physical parameters, adapt to different driver behaviors/tendencies, and handle system nonlinearities with the nonlinear nature inherent in neural networks.

MacAdam and Johnson [41] later upgraded MacAdam’s optimal preview controller [28] to an ANN implementation in 1996. The preview notion was salvaged from the optimal controller and interfaced with a two layer neural network trained through backpropagation with learning rate scheduling and momentum. The ANN model was compared to human driver data for double lane change maneuvers and s-curve scenarios, demonstrating an impressive reproduction of human behavior. MacAdam and Johnson attributed much of the neural network model success to the use of sensor preview information in the estimation of driver steering control actions; shortening network training time and improving accuracy. The authors concluded that the adaptive nature of neural networks could effectively model human driver behavior over a wide range of operating conditions and scenarios.

Also in 1996, An and Harris used an adaption of ANNs, called the Cerebellar Model Articulation Model (CMAC) to model longitudinal car following with online training [42]. The advantage of the CMAC network was the network output’s linear relationship with
the weights of the network, allowing less computationally intensive training through backpropagation with learning rate scheduling for online training. Although linear, the CMAC network could still generally model low-dimensional nonlinear systems. The authors achieved satisfactory results with the model in comparison to human driver data; however they explained inaccuracies in the model due to nonstationary characteristics of the human as well as a lack of sufficient input variables. This conclusion seemingly undermines the validity of using the CMAC as a model for inherently nonlinear human drivers as the model can only “generally” capture nonlinear relationships with a small amount of inputs.

More recently, in 2005 Lin, Tang, Zhang and Yu [43] extended the work of Macadam [41] by significantly increasing the amount of inputs and preview points used by the ANN human driver model. The model used 7 inputs, ranging from current yaw angular velocity and roll angle to preview lateral offset; while outputting steering wheel angle and rate. Lin et al. also experimented with several types of more advanced ANN structures in an attempt to decrease network training time: the Counter Propagation Network (CPN) and the Radial Basis Function Network (RBFN). These ANN structures were also compared with the conventional Back-Propagation Network (BPN) used by Macadam. They found that the RBFN, CPN and BPN took 8, 8.5 and 40 minutes, respectively, to fully train. The authors compared a simulation of their model to human driving data in a single and double lane change maneuver as well as navigation of an s-curve, concluding that the RBFN network was the most promising model of human driver behavior due to its accuracy and low training time.

1.4.2: Motion Planning
Motion planning, or the calculation of a trajectory throughout an environment, has been a major focus of robotics researchers for many years. Many different methods have been explored, ranging from global optimal path pre-planning to real-time local path planning. The most notable and widely used methods include A* based algorithms, which optimize the path cost based on traversals through a grid-map, Genetic Algorithms that use evolutionary algorithms to find an optimal path, and Potential Field Theory which treats obstacles and goals as repulsive and attractive forces on the robot. These planners offer many different advantages and disadvantages in terms of optimality and computation cost and will be outlined in the following subsections.

1.4.2.1: A* and D* Search Algorithms
The A* search algorithm was presented by Hart, Nilsson and Raphael in 1968 [44]. The algorithm, based on a grid search, could find a minimum cost path between two nodes with examination of a minimum number of intermediate nodes. The process worked through the successive “expansion” and cost calculation of nodes along a path between a start and goal node. Path costs were determined with the use of a heuristic function
that would assign a cost-of-traversal to nodes if they were included in the path solution and sum the total cost of each traversal along the path. Hart et al. proved admissibility, or the guarantee to find an optimal path, and optimality of A* as long as a proper heuristic function was chosen. The heuristic function provided the user with the ability to customize the A* search algorithm for many different purposes by allowing them to choose how the cost of traveling from one node to another node was calculated. Another advantage of A* was that it could be used for any type of map that implements a node structure, including grid maps and visibility graphs.

In 1993, Stentz extended the A* algorithm into the D* algorithm by incorporating dynamic cost functions within path calculations [45] [46]. The new D* algorithm acted as a re-planning version of A*, where empty or incomplete node maps could be solved repetitively as a robot equipped with sensors would discover new features of the map. D* was much more efficient than A* as it used the dynamic cost functions to “repair” previous solutions when new map features were found, rather than completely restarting the solving process. In 1995, Stentz and Hebert demonstrated the capabilities of the D* algorithm by implementing it in a full size autonomous vehicle and successfully traversing a 0.3 square mile area at about 5mph [47]. During that same year, Stentz also significantly improved the runtime of the D* algorithm by a factor of 2-3 through the focusing of repairs made to path solutions [48]. Using an additional heuristic function, a second set of termination criteria for node expansion and cost updates was utilized to ensure that only the minimum amount of repairs were made. The improved algorithm also incorporated the use of a biasing function to improve cost accuracy by compensating for robot motion between planning iterations.

Several years after the development of D*, Koenig et al. proposed a similar, but separate extension of the A* algorithm called the Lifelong Planning A*, or LPA* [49] [50] [51]. The goal of the algorithm was to implement A* in such a way that it could once again be used for incomplete maps and continuously re-plan paths as the map was explored. This particular contribution employed the use of incrementally built search trees (groups of node extensions) that were reused and updated in the re-planning process rather than rebuilding them entirely every time the map was updated. After quickly realizing the similarity between the LPA* and D* algorithms, the authors made minor reconfigurations to LPA*, such as incorporating the D* bias functions for robot movement compensation, and called it D* Lite [52] [53]. The authors claimed that the D* Lite algorithm was functionally the same as D*, but algorithmically different and much more simple; offering decreased runtimes in certain implementations and making it easier to extend.

In parallel with the development of D* Lite, Likhachev et al. contributed the Anytime Re-planning A* (ARA*) algorithm in 2004 [54]. The ARA* algorithm combined the search tree reuse from LPA* and combined it with an algorithm that increased path optimality
based on available search time. This was done by executing a series of searches with decreasing sub-optimality bounds as re-planning iterations continued. The advantage of ARA* was that it would operate in real-time due to incremental tree reuse from LPA* and provide as optimal a solution as possible based on the given time allotment for the path calculation. One year later, the authors combined D* Lite with ARA*, calling it Anytime Dynamic A* (AD*) [55]. AD* was an adaptation of the ARA* algorithm to provide anytime solutions in a dynamic environment.

Most recently, Ferguson and Stent presented the Field D* algorithm, an extension of D* and D* Lite to use linear interpolation within grid nodes to find smoother and more optimal path solutions [56] [57]. The linear interpolation method treats node boundaries as continuous locations for possible path points, dramatically increasing the resolution of the path without requiring an increase in resolution of the occupancy map. Ferguson and Stent also proposed the Multi-Resolution Field D*, an implementation of Field D* using quad trees to allow grouping of grid nodes with similar costs into one node [58]. This allowed for less memory consumption by the map and less calculation for the planning algorithm. It allowed a single map to have coarse node grids for areas with the same cost, for example a large empty parking lot, and fine node grids for areas with high concentration of obstacles or un-traversable terrain, such as a parking lot filled with cars.

1.4.2.2: Genetic Algorithms
In 1993, Bessiere et al. proposed the use of genetic algorithms (GAs) for path planning in holonomic robots [59]. The search space was described as a subspace consisting of all possible paths starting from an origin point. Paths were coded into chromosomes as a list of rotate and move commands. The fitness function used was rather simplistic in that it only takes into consideration the distance of the last point in the path to the goal. The authors demonstrated that this evolutionary algorithm for motion planning could actually be implemented in real-time by the use of massively parallel computing. Each generation of the population was dispersed amongst 128 processors which performed the necessary GA operations on the assigned sets of individuals in parallel. Several years later, the authors built on their GA path planning implementation to create the Ariadne’s Clew Algorithm, an algorithm that uses two GAs in parallel to both explore and search a map for path solutions [60]. The explore algorithm continuously places landmarks throughout the search space. The search algorithm then looked for fixed length Manhattan motions created from the landmarks to find a path to a target from the current location. The main advantage given through this implementation is that the problem is solved in the trajectory space and thus there is no need to explicitly compute the configuration space [61].

Xiao and Michalewicz presented a different implementation of dual GA path planning one year later [62]. Their contribution included a GA for near-optimal global path
calculation and a second GA for optimizing the global path locally with the use of environment sensors positioned on the robot. The chromosomes are comprised of variable length ordered lists of x-y pairs corresponding to points on the map grid. The chromosome also includes a Boolean variable describing the feasibility of that particular point to describe if it causes a path intersection with an obstacle. A simple fitness function based on path length and feasibility is used throughout the evolutionary process. The authors provide several simulation scenarios demonstrating satisfactory performance. In 1997, Xiao et al. extended their work by increasing the amount of genetic operators and adaptively choosing the influence of each operator on a generation based on three estimated probabilities: the effectiveness of path improvement, calculation time cost, and side effects it could cause in future generations [63]. This extension led to significantly improved path calculation in time-variant environments.

Sugihara and Smith [64] created a GA implementation for 2 and 3-dimensional path planning in time-varying environments. Using monotonic assumptions, the authors contributed several novel coding schemes for representing paths and motions as chromosomes. For 2D path planning, a fixed length set of direction and distance pairs formed a single chromosome. The authors stressed the importance of the fixed length chromosome as it simplified genetic operations and improved the GA runtime. Sugihara and Smith created a 2D simulation environment with varying amounts of non-stationary obstacles and found that the GA could successfully optimize a path, however it required several minutes per optimization. The authors concluded their work with the suggestion of several coding schemes to implement their work on a 3D environment: chromosomes could be comprised of interleaved x-z plane and y-z plane points, interleaved x-y, x-z, and y-z plane points, or even variable length sequences of directions.

Recently, in 2003 Tu and Yang wrote about the importance of using variable length chromosomes for path planning in dynamic and complex environments [65]. They proposed a GA with variable length chromosomes coded with path directions and distance amongst a grid-based occupancy map. In 2004, Yang and Hu [66] dramatically improved the previous implementation by changing the chromosome structure to work with integer-assigned grid locations that acted as and ordered set of nodes in the planned path. They also used a novel set of knowledge-based genetic operators to significantly improve the algorithm performance. Aside from the normal selection, crossover and mutation operators, it included node-repair to move a node out of an obstacle to the next best location, line-repair to insert a node in the path to force it around an obstacle, deletion of a node if it increased fitness in any way, and improvement to move a node to a better location within a local neighborhood in the grid. The authors ran simulations on several grid sizes ranging from 16x16 to 32x32 grids and found optimal paths in 5.2-8.2 seconds on a 933MHz processor.
1.4.2.3: Potential Field Theory

In 1986, Khatib presented the first real-time obstacle avoidance implantation using artificial potential fields [67]. This algorithm was designed for both robotic manipulators and mobile robots in dynamic and/or incompletely mapped environments. Khatib described an operational space formulation that calculated repulsive and attractive forces on the robot from artificial potentials caused by goals, constraints and obstacles included in the system’s Lagrange’s equations. These potential fields were described in continuous and differentiable equations that could also limit the influence of a field to within a certain vicinity of the object itself. Khatib also implemented a method to set a maximum velocity for the robot’s degrees of freedom while still allowing accelerations and decelerations related to the potential forces. Khatib demonstrated the capabilities of potential field theory on a multiple degree-of-freedom robotic arm with satisfactory results in goal tracking and obstacle avoidance.

Borenstein and Koren extended Khatib’s work by combining certainty grids with artificial potential fields, calling it the Virtual Force Field (VFF) method [68]. In this local planner application, a grid centered on the robot is continuously updated with obstacle certainties found with sensors located on the robot. Repulsive potential forces created by a particular grid cell were proportional to the cell certainty level and inversely proportional to the squared distance of the cell to the robot, while attractive potential forces were constant in magnitude but always directed towards the goal. The potential forces were multiplied by the directional cosine to dampen heading changes in the robot and effectively control speed. All potential forces were summed together to create the total resultant force magnitude and direction on the robot. The authors also handled trap situations by continuously monitoring heading angle to robot-to-target angle and switched to a wall following algorithm until the trap was cleared. The VFF algorithm was tested on a small mobile robot and allowed it to successfully navigate obstacle courses; however oscillations in the trajectory were still present.

In 1991, Borenstein and Koren presented the Vector Field Histogram (VFH), an improvement of their VFF algorithm that lost less information in the data compression stages [69]. This particular implementation used a 2D histogram to replace the certainty grid around the robot, which was then compressed into a 1D polar certainty histogram. The algorithm would examine the polar certainty histogram for candidate valleys below a certain threshold and choose the valley that corresponded with an angle closest to the robot-to-target angle. This type of heading selection smoothed trajectories while increasing the amount of data used when compared to the original VFF method. In 1998, Ulrich and Borenstein took the algorithm one step further, adding two more stages of data compression and calling it VFH+ [70]. This extension uses the same polar certainty histogram, but then converts it into a binary polar histogram using hysteresis thresholds to generate even smoother trajectories. The binary polar
histogram was then masked to block heading angles that would require unachievable trajectories from the robot due to turning radius characteristics. Finally, a customizable cost function was used to choose a heading angle out of the remaining free angles on the masked binary histogram. This algorithm offered significant improvements and elegance to the VFF and VFH methods, creating smoother trajectories and effective real-time environment navigation.

1.4.3: Human-Computer Interface

Human-Computer Interfaces (HCIs) establish the main communication link between a human and computer in any type of system; ranging from the keyboard and monitor on a standard home PC to the low-fuel auditory alarm on a military fighter jet. In this particular field of research, the HCIs provide a means of physically communicating important driving information, such as vehicle environment descriptions and reference path suggestions, to a human operator. The main problem posed by HCIs is that complex computer signals must be presented to the human in a manner that can be efficiently understood through the human’s senses. Many researchers have attempted to tackle this problem by using a range visual and non-visual HCIs to communicate important driving information to the human operator. The most prominent contributions made are outlined in the following subsections.

1.4.3.1: Visual Interfaces

Lim et al. designed a Heads-Up Display (HUD) in 1999 to overlay lane boundaries on the road in low visibility conditions using a Differential Global Positioning System (DGPS) linked with a geo-spatial database [71]. The system used a Novatel DGPS to provide global pose and heading information, while the geo-spatial database provided lane position information for a particular location. The lane positions were then projected onto a 4”x6” combiner screen for display to the driver. The authors experimented with the HUD placed on a vehicle and experienced significant amounts of projected lane position at longer distances (300-400ft) due to heading error. One year later, they implemented a higher frequency Trimble DGPS system to mitigate the projected lane position error [72]. However, significant error was still present in the system due to DGPS positional error and the calculation of heading based solely on previous latitude and longitude points.

Steinfeld and Tan [73] [74] presented a study on the use of visual feedback for lateral lane position keeping in low-visibility conditions for snowplow drivers. Two separate interfaces designed to provide visual cues through a dash-mounted monitor were tested: a single bar graph and an overhead road map. The bar graph was used to communicate lateral lane position error; it was found that using a preview distance of 50 feet provided better performance than displaying instantaneous error. The road map provided a bird’s eye view of the vehicle, lane boundaries, and the predicted future
position of the vehicle based on the current steering angle. The road map demonstrated the strongest performance with maximal errors of ±1.5 feet. The authors concluded their work with a statement that properly trained users achieved a stable level of performance and found the interface itself intuitive and simple to use.

In 2011, Ravani Gabibulayev and Lasky extended Steinfeld and Tan’s work by improving the graphics of the road map display and added event-driven visual and audio cues based on the location of the predicted future vehicle position [75]. If the predicted vehicle position approached a lane edge, different tones would alert the driver to which side the deviation occurred on. Large yellow arrows would then be displayed on the screen to also assist the driver in correcting the lateral deviation. The improved interface contributed to half the number of lane boundary crossings and time spent looking at the display when compared to Steinfeld and Tan’s previous implementation. Much of this success was attributed to the beeping tones alerting the driver in a timely manner without requiring the gaze of the driver.

1.4.3.2: Non-Visual Interfaces

In 1965, Fenton proposed a tactile joystick for use in longitudinal vehicle control [76] [77]. The joystick, outfitted with a linearly-actuated “finger” at the tip, alerts the driver to speed deviations and headway before a leading vehicle while simultaneously allowing the driver to operate the vehicle with the joystick itself. At first, the interface was tested on a miniature-scale mechanical simulator of a car following scenario. Results indicated safer following habits with up to 90% reduction of headway variance and 50% reduction of velocity variance. Two years later, Fenton and Montano attached the tactile joystick interface to an actual vehicle and ran experiments in a straight road car following scenario [78]. The full-scale test reproduced similar results with the simulation, and demonstrated effective control of headways of 30-60 feet at speeds of up to 40 mph. Fenton concluded that the tactile interface provided substantial improvements in car following and speed maintenance with negligible effects from fatigue and desensitization.

More recently, van Erp and van Veen constructed a vibro-tactile seat to communicate course direction changes and distance information [79] [80]. The interface consisted of four vibrating motors placed under each thigh that vibrated at different intensities and gave rhythmic alternations of power to communicate information. Course direction changes were signified by vibrating all motors under the left thigh for a left turn, all motors under the right thigh for a right turn, and all motors together for a straight course. Distance to both waypoints and course changes were provided through the rhythmic power alternations and changing intensities that were associated with the total amount of remaining distance. The authors concluded that the vibro-tactile interface decreased the mental effort of the driver by 20% in a simulated experiment and provided the best performance when coupled with a visual cue system.
In 2008, Uchiyama, Covington and Potter designed a vibro-tactile glove to provide warnings, spatial representations and directional information to a visually impaired electric wheelchair operator [81]. The glove incorporated a 3x3 vibratory motor array located on the back of the hand and was controlled through a simple PC interface board. The wheelchair, outfitted with a computer and close-proximity sensor array, determined a reference direction and the locations of nearby obstacles. This information was then supplied to the operator through vibration patterns in the glove. Warnings were signified by long-duration pulses on the entire vibration array. Obstacle direction was communicated by vibrating the motor in the 3x3 array that most closely pointed toward the obstacle, while distance was indicated through pulse patterns of the same motor. Directional information was supplied by creating two short pulses on the middle vibration motor followed by another two short pulses on an outer ring motor corresponding to the desired direction. Although the glove implementation gave satisfactory results of wheelchair guidance in a controlled environment, the extremely low resolution would not allow the system to work in scenarios where more than two obstacles are present in the wheelchair’s field of view.

1.4.4: Model Predictive Control

Model Predictive Control (MPC) is an advanced method for controlling complex systems through the use of models that can predict the response of a system given the appropriate inputs. If the system output can be reliably predicted, then an optimization scheme may be used to determine an appropriate set of “manipulated” control inputs that will force the open loop response of the system to match a desired response. MPC provides a significant advantage over classical control theory in that it can control significantly nonlinear systems with time delays and higher order dynamics in a more robust manner. MPC has been exceedingly popular in the field of process engineering, ranging in use from chemical plants to oil refineries. While the generic MPC method has become clearly defined, current research focuses on exploring combinations of different modeling and optimization schemes as well as attempting to make a shift towards more robust online training models. Recently, a large push has also been put forward to incorporate the inherently nonlinear neural network modeling scheme into MPC implementations, making it more attractive to nonlinear and time-variant applications such as human modeling and interfacing.

While extensive literature on MPC can be found in [82] [83] [84] [85] [86] [87] [88], this section of the literature review will cover the earliest work on MPC and then shift in focus to cover the most recent and prominent works in neural network-based MPC. As a significant review of modeling literature has already been covered in this paper (Section 1.4.1), the review of MPC literature will concentrate on the higher-level strategic methods and the corresponding performance discussed within each referred work.
In 1976 and 1978, Richalet et al. [89] [90] presented a series of papers that discussed the first application of MPC in which they called the Model Predictive Heuristic Control (MPHC) method [91]. In this method, the authors proposed the use of a linear finite impulse response (FIR) model to predict the output behavior of a plant when subjected to any horizon of control inputs. A heuristic-based iterative algorithm would then be utilized to optimize a quadratic performance objective that could be customized for any particular application. Richalet et al. performed a stability analysis of the MPHC method and additionally applied it to several different process control problems, namely the synthesis of polyvinyl chloride (PVC), controlling a distillation column in an oil refinery, and controlling the steam generator of a power plant. The authors concluded that the MPHC method could effectively and robustly control linear SISO systems as long as the model mismatch stays within a defined range.

More recently, Arpornwichanop et al. compared MPC to generic model control (GMC) for controlling a batch reactor for chemical processes in 2002 [92]. For both methods, the authors implemented a theoretical model of the batch reactor and an Extended Kalman Filter (EKF) to predict the output of the plant. The MPC optimization process used the EKF model to find a set of manipulated control inputs by minimizing the sum of squared deviation between the reference outputs and the predicted model outputs over the prediction horizon. After comparison of the MPC and GMC methods, the authors found that the MPC method was more efficient and accurate in nominal operation and even in the case of model mismatch.

In 2005, the authors continued research on the MPC method and applied it to a steel pickling process using an inverse neural network model in order to better encapsulate plant nonlinearities [93]. The model was comprised of a feed-forward neural network that was trained offline using the Levenberg-Marquardt method on data calculated from a simplified nonlinear model of the plant. This particular implementation was interesting in that rather than using an optimization scheme to calculate the manipulated control input, the neural network model directly output the manipulated control input. The authors were able to achieve this because the neural network model emulated the inverse model of the plant; the network would input a reference value one time-step into the future and would calculate the required open loop plant input. The performance of the new MPC method was compared to a PI controller and demonstrated better transient performance with higher robustness even in the case of model mismatch. The only disadvantage of the MPC method was its lack of steady-state offset rejection.

The authors extended their work in 2009 by mixing [92] and [93] and ultimately reverting back to the standard model-optimizer MPC scheme [94]. In this work, the neural network model predicted future outputs of the steel pickling process based on knowledge of the past control inputs and plant outputs. Similarly to their past work [93], the model used a feed-forward network trained using the Levenberg-Marquardt method.
on offline data taken from a simplified nonlinear model of the plant. After successfully forward modeling the system, an optimization routine similar to that of [92] was used to find the set of manipulated control inputs based on the minimization of the sum of squared deviation between the predicted and future reference outputs over the prediction horizon. Once again, the authors compared the MPC method to a standard PI controller and found that the MPC continued to outperform the PI controller in transient performance and robustness; however the combined usage of the neural network model and optimization scheme eliminated the steady-state offset problem noted in [93]. In 2010 and 2011, Vasičkaninová et al. created a quite similar implementation of the MPC method, however their work proposed the use of an extra “smoothing” term in the optimization objective function to prevent large changes in the manipulated control inputs and ultimately decrease overall control effort [95] [96].

In 2006, Wang et al. presented a neural network based MPC method that used online neural network training to significantly decrease model prediction error and compared it with a similar offline trained model [97]. The Radial-Basis Function (RBF) neural network was used to model and predict future air-fuel ratios (AFRs) of an automotive engine with knowledge of past AFRs, fuel injection rates, and throttle angles. The offline RBFN model was trained using a Recursive K-Means (RKM) algorithm with data from a theoretical plant model while the online RBFN model was trained using the Recursive Least Squares (RLS) method from online simulation data. In the comparison, both the offline and online models coordinated with a Reduced Hessian SQP optimization method to minimize the sum of squared deviation between the reference AFR and the models predicted AFR while simultaneously penalizing large changes in the manipulated fuel injection rate control effort. The offline and online MPC methods were additionally compared with a PI controller where it was demonstrated that online MPC method outperformed the others with improved transient responses, lower control efforts, and improved model prediction.
1.5: Contribution
The Blind Driver Challenge® Problem Statement in Section 1.3 describes a need for transformation algorithms to generate navigation information that a human driver may interact with and provide a physical means for communicating that information to the driver in a non-visual manner. In order to solve these issues, this work proposes a Non-Visual Interface System as well as a series of adaptive and passive Non-Visual Interface Drivers.

1.5.1: Proposed Non-Visual Interface Drivers
The Problem Statement in (Section 1.3) describes a requirement for transformation algorithms that convert computer-synthesized motion planning signals into non-visual stimuli that can be understood and reliably interacted with by the human driver. It is also stated that the algorithms may be passive or adaptive in nature; interacting identically amongst all drivers or adapting to each particular driver’s tendencies. This work proposes two software algorithms that implement the passive and adaptive transformation algorithms:

1. **The Passive Non-Visual Interface Driver (PNVID)**
   This software implements a passive transformation algorithm that converts TORC AutonoNav™ motion planning signals into human-computer interface signals. Trajectory motions planned by the proven TORC AutonoNav™ systems are dependable and robust; however they are designed for fully autonomous vehicles outfitted with electro-mechanical actuators for vehicle control. The passive transformation algorithm within this software uses an innovative method to convert the AutonoNav™ navigation motions into non-visual stimuli that can be understood and reliably interacted with by a human driver.

2. **The Adaptive Non-Visual Interface Driver (ANVID)**
   This software implements an adaptive transformation algorithm utilizes a Model Predictive Control (MPC) technique to transform TORC AutonoNav™ motion planning signals into non-visual stimuli that adapt to each particular driver in real-time. This is achieved through the use of neural network-based models that can predict a driver’s reaction to particular non-visual stimuli. A Quasi-Newton optimization method utilizes the prediction model in parallel to find an improved set of non-visual stimuli based on the required navigation actions defined by the TORC AutonoNav™ motion signals. The advantage of this software is that it continuously adapts an internal model of the human in real-time to tailor the non-visual stimuli to the driver’s comfort and driving tendencies.
1.5.2: Proposed Non-Visual Interface System
The Problem Statement in (Section 1.3) describes a requirement for human-computer interfaces that can non-visually communicate the transformed navigation information to the human in a dependable manner. The third contribution of this work proposes a Non-Visual Interface System to solve this issue:

3. The Non-Visual Interface System
This system is comprised of a set of non-visual human computer interfaces and associated control/mounting hardware that, together, communicate the transformed navigation information to the driver in a dependable manner. The non-visual interfaces, named DriveGrip and SpeedStrip, inform the driver of necessary steering and speed actions that enable the driver to safely navigate a full-size vehicle. These non-visual interfaces are redesigned from previous prototypes to substantially improve the overall functionality, performance, and robustness of the system. The associated control and mounting hardware is developed to seamlessly integrate the Non-Visual Interface System within the TORC ByWire XGV™ research platform while maintaining the stock vehicle appearance and level of comfort.

1.6: Outline of Dissertation
Section 1 includes the introduction, problem statement, literature review, and contribution statement of this work. Section 2 introduces the Blind Driver Challenge® research platform: the TORC Robotics ByWire XGV™. This section describes the features and abilities of the research platform in which the contributions of this paper were implemented upon. Section 3 presents the 3rd contribution of this paper: the Non-Visual Interface System. This section presents the DriveGrip and SpeedStrip non-visual interfaces as well as the associated control and mounting hardware for platform integration. Because the 1st and 2nd contributions of this work rely heavily on the Non-Visual Interface System, they will be presented afterwards in Section 4 and 0. Section 4 presents the 1st contribution of this paper: The Passive Non-Visual Interface Driver (PNVID). This section describes the algorithm and software implementation that performs the necessary passive transformations to generate non-visual stimuli. Section 5 presents the 2nd contribution of this paper: The Adaptive Non-Visual Interface Driver (ANVID). This section describes the algorithm and software implementation that performs the necessary adaptive transformations to generate non-visual stimuli that adapt to each driver in real-time. 0 provides conclusions on the contributions of this work and suggests future avenues of research. Section 7 lists the references to previous literature used in this paper, and Appendix A includes additional appendices.
Section 2: The Blind Driver Challenge® Research Platform

The Blind Driver Challenge® research platform consists of all hardware and supporting software related to the implementation and control of the full-size vehicle used in this research effort. The platform itself is composed of the TORC ByWire XGV™ standard vehicle platform, perception sensors, semi-autonomous systems, and non-visual human-computer interfaces. These separate systems are all combined through a central network that uses a standardized method for system organization and communication. The research platform additionally includes a full software simulation package as well as a development tool for defining vehicle missions. The following subsections describe each of these platform components in more detail.

2.1: The TORC ByWire XGV™ Standard Platform

The ByWire XGV™ (Figure 2-1) is a Ford Escape Hybrid SUV modified by TORC Robotics for drive-by-wire operation. It provides access to vehicle sensor data, the ability to remotely control the vehicle, and includes safety systems. The features and capabilities of the ByWire XGV™ are summarized in the following subsections; detail in full can be found in the TORC ByWire XGV™ User Manual [98].

Figure 2-1: The TORC ByWire XGV™ research platform.
2.1.1: The Ford 2010 Escape Hybrid SUV

The base vehicle on which all other modifications were made is the Ford 2010 Escape Hybrid Limited SUV. This particular model incorporates several features: a hybrid electric power system, Active Park Assist, built-in drive-by-wire components, various onboard sensors, and an OBD-II based diagnostics system [99]. Each of these features provides significant advantages in TORC’s modification of the vehicle into a ByWire XGV™, and is detailed in [10]. A summary of these advantages shall be discussed here.

The hybrid electric power system on the vehicle includes a large 330V NiMH battery pack coupled with regenerative braking and alternator charging systems. The hybrid ability of the vehicle keeps the battery pack charged at all times during operation. This makes a large, stable and steady amount of electrical power available on the vehicle that can be easily tapped into for powering added electrical equipment required by the ByWire XGV™ conversion.

The Active Park Assist feature allows the vehicle to automatically parallel-park itself by controlling the steering angle while the driver controls throttle and brake. The vehicle thus integrates a fully actuated steering wheel outfitted with an encoder for feedback that can be used for drive-by-wire applications. Separately, the 2010 Ford Escape Hybrid also includes stock drive-by-wire controls for the shifting and throttle systems as well as a wide array of onboard vehicle sensors, including wheel speed sensors. These systems can similarly be tapped into for use by other systems. With the combined advantages gained from these factory features, the 2010 Ford Escape Hybrid Limited SUV could, in a relative sense, be easily and safely converted into the TORC ByWire XGV™ research platform.

2.1.2: TORC ByWire XGV™ Conversion

The TORC ByWire XGV™ conversion builds on the standard capabilities of the Ford 2010 Escape Hybrid SUV to add all of the necessary hardware and software for full access to and control of the vehicle state via remote control. The ByWire XGV™ relies on both factory installed components as well as custom-interfaced hardware added to the vehicle. The sensors, control features, onboard computers, and safety equipment included in the conversion are discussed in detail in [98]; however an overview is provided in the following subsections.

2.1.2.1: Sensors and Control Features

The features of the ByWire XGV™ can be logically separated into four main components that provide comprehensive control and awareness of the vehicle state: the Primitive Driver, the Motion Profile Driver, the Signals Driver, and the Velocity State Sensor. Each of these components are described below, and the capabilities of each are outlined in Table 2-1.
Primitive Driver: This component is primarily responsible for sensing and controlling low-level, open loop functions related to vehicle driving and general operation. The Primitive Driver (PD) provides direct, manual operation of the vehicle.

Motion Profile Driver: This component is responsible for sensing and controlling high-level, closed loop driving functions. The Motion Profile Driver (MPD) provides direct closed loop control of the vehicle’s wrench efforts (described in the Primitive Driver) based on a supplied trajectory plan.

Signals Driver: This component provides the sensing and control of vehicle signaling hardware states.

Velocity State Sensor: This component allows sensing of the vehicle velocity in the longitudinal direction.

Table 2-1: Vehicle Component Capabilities. All items listed can be sensed; items marked with an asterisk (*) can additionally be controlled.

<table>
<thead>
<tr>
<th>Primitive Driver</th>
<th>Motion Profile Driver</th>
<th>Signals Driver</th>
<th>Velocity State Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrench Efforts</td>
<td>*Motion Profile (Closed Loop Wrench Efforts)</td>
<td>*Turn Signal State</td>
<td>Longitudinal Velocity</td>
</tr>
<tr>
<td></td>
<td>*Vehicle Power State</td>
<td>*Horn State</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Parking Brake State</td>
<td>*Headlights State</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Transmission Gear Position</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Throttle (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Brake (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Steering Wheel Position (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete Devices</td>
<td>*Vehicle Power State</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Parking Brake State</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*Transmission Gear Position</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Horn State</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fuel Level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independent Wheel Speeds</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.1.2.2: Embedded Controller
All components and associated sensing/control features are managed through a central, onboard vehicle controller. The controller, located below the center console, is a real-time/FPGA system added by TORC Robotics. It includes a customized interface module that provides physical connection to TORC hardware as well as a CAN network link to the vehicle’s diagnostics system for connection to factory-installed hardware. The embedded controller uses a standard Ethernet connection for end-user interfacing (see Section 2.4.4: JAUS Interoperability). Power of the controller is governed through a dash-mounted key switch; powering off the controller enables the ByWire XGV™ to act as a standard vehicle as opposed to a drive-by-wire system.

2.1.2.3: Safety Equipment
The TORC ByWire XGV™ is a system developed primarily for autonomous and semi-autonomous research; thus there is an important requirement for safety features implemented on the vehicle. The ByWire XGV™ uses a comprehensive set of safety overrides to regain control of the vehicle or safely bring it to a controlled and complete stop. The safety system operates through three different safety modes: Manual Override, Emergency Stop Disable, and Emergency Stop Pause.

Manual override of the vehicle is enabled through two separate methods. The first method is to change the transmission gear from Neutral to Drive (Figure 2-2), as the drive-by-wire functions of the ByWire XGV™ are only enabled in Neutral gear. By switching to Drive, the drive-by-wire systems are software-overridden and the platform acts as a stock vehicle. The same effect can be achieved by using the second manual override method: depressing the Emergency Manual Override button located on the center console (Figure 2-2). The only difference with this method is that the drive-by-wire system is overridden in hardware by cutting power to the drive-by-wire modules and forcing the vehicle to revert to stock operation.

Emergency stop disabling of the vehicle is controlled through a variety of different methods, including emergency stop buttons located on the vehicle, wireless emergency stop buttons, and through software. Emergency vehicle disabling stops the vehicle by applying 100% brake and can also be configured to additionally cut off the engine power. The vehicle includes three on-board emergency stop buttons: one located on the center console inside of the vehicle (Figure 2-2), and two located on the outside of the rear-passenger doors. A wireless emergency stop system, called the TORC SafeStop™ [100], is also included (Figure 2-2). This handheld wireless system can raise an emergency stop event from up to 6 miles (line-of-sight) and has a 30 hour battery life. Depressing any of the on-board or wireless emergency stop buttons will disable the vehicle until the button is released. Emergency stop disabling can also be achieved through software (see Section 2.4.4: JAUS Interoperability) as the embedded controller on the vehicle maintains a complete list of current hardware and software
errors through an Error Manager component. An emergency stop disable will be triggered based on the criticality of any outstanding vehicle errors.

Emergency stop pausing of the vehicle is controlled through wireless pause switches and through software. An emergency stop brings the vehicle to a stop at a configurable deceleration rate without disabling any of the other systems. The wireless pause switch is also located on the handheld TORC SafeStop™ [100] system (Figure 2-2) used for emergency stop disabling. The vehicle can additionally be paused through software messaging over the embedded controller Ethernet port (see Section 2.4.4: JAUS Interoperability). Releasing the pause state via the wireless controller or through software messaging resumes the ByWire XGV™ normal operation.

Figure 2-2: TORC ByWire XGV™ Emergency Stop Disable/Pause and Manual Override Systems.

2.1.2.4: Power Systems
The TORC ByWire XGV™ conversion includes the provision of controlled power supply systems for usage by the end-user. These systems include a Tripp-Lite SMART1500RM2U Uninterrupted Power Supply (UPS) and a TORC PowerHub™ [101] for AC and DC power. The UPS allows for uninterrupted power when switching between vehicle battery power and wall outlet power during experimentation. It also provides eight 120VAC power outlets for powering equipment such as laptops during in-vehicle experiments. DC power for custom components and sensors is supplied through the TORC PowerHub™ system, which consists of rack-mounted 12VDC and 24VDC power modules. These modules each provide eight power outputs that can be individually controlled and configured through software via an Ethernet connection. Combined, the UPS and PowerHub™ systems provide the end-user with up to 500W of power for usage in their custom application.
2.2: Environmental Perception Sensors
The TORC ByWire XGV™ supplied the Blind Driver Challenge® with a professional and dependable base vehicle platform; however the XGV™ is only a drive-by-wire system and lacks environmental perception features that are essential to the driver assistance algorithms presented in this work. These algorithms depend on knowledge of the vehicle driving environment, which includes road position, obstacle detection and classification, and an inertial navigation solution. The sensors implemented on the Blind Driver platform to perceive these environmental variables are described below.

2.2.1: Inertial Navigation Solution
The inertial navigation solution describes the position, velocity, and orientation of the BDC platform in a global frame of reference. In this application, the solution is primarily used for navigation within a predefined roadmap described by the system’s travel mission. The solution from the sensor makes the system aware of its current position within the mission and also allows global motion planning to continuously calculate a path to the mission destination in real time. The high accuracy of the positional data makes lateral and longitudinal control of the vehicle possible even without active road detection. Therefore, the BDC application uses a combination of the inertial navigation solution and active road detection for global and local motion planning.

2.2.1.1: Sensor Specification
The sensor used to supply the inertial navigation solution is a NovAtel® SPAN-CPT™ Inertial Navigation System (INS) (Figure 2-3). An Inertial Navigation System uses a combination of a Global Positioning System (GPS) sensor and an Inertial Measurement Unit (IMU) sensor to estimate an accurate solution despite the inaccuracies and uncertainty inherent in the separate sensors themselves. The NovAtel® SPAN-CPT™ provides a full INS solution at a frequency of up to 100Hz with a global accuracy of 1.5m [102]. An OmniStar HP subscription was also included in our application to increase the global accuracy of the system to 0.1m. GPS satellite communication required by the INS is received through the NovAtel® GPS-702L high performance GPS antenna (Figure 2-3).
In order to combat inaccuracies in the global pose solution from GPS drift, the system also includes a TORC Localization Board (Figure 2-4). This JAUS interoperable equipment combines the INS solution from the NovAtel® SPAN-CPT™ with the independent wheel speeds measured from the ByWire XGV™ through a high-speed Extended Kalman Filter. This additional filtering uses knowledge of the vehicle wheel speeds to maintain a certain level of global positional accuracy even in the event of drift or sudden changes in GPS data that propagate through the NovAtel® SPAN-CPT™’s internal INS filtering. The addition of the TORC Localization Board provides significant advantages in terms of sensor accuracy; however, it is an International Traffic in Arms Regulations (ITAR) restricted item under Section 121.5 of the United States Munitions List [103].
2.2.1.2: **Platform Implementation**

The NovAtel® SPAN-CPT™ and the TORC Localization Board are both rigidly mounted to the BDC vehicle frame and are located on the equipment rack in the trunk of the vehicle. The SPAN-CPT™ is installed on the vehicle equipment rack located in the trunk as shown in Figure 2-5. The equipment rack is mounted on the vehicle through vibration isolators, preventing road/vehicle vibrations from influencing the measurements of the SPAN-CPT™. The NovAtel® GPS-302L GPS antenna is magnetically mounted to the rear-roof of the vehicle for maximum satellite signal quality. The TORC Localization Board is mounted (Figure 2-5) next to the SPAN-CPT™ as it shares a direct connection to the INS. The Localization Board is also directly connected to the Ford Escape’s diagnostics bus for maximum-speed acquisition of independent wheel speeds. Figure 2-6 displays the connection architecture for the collective inertial navigation solution system.

![Figure 2-5: Mounting configuration for (A) NovAtel® SPAN-CPT™ (B) TORC Robotics Localization Board (C) Vibration Isolating Mounts for equipment rack](image)
2.2.2: Obstacle Detection and Identification
The BDC platform uses obstacle detection and identification to determine the location and classification of static and dynamic objects within the vehicle’s vicinity. Obstacle detection is primarily responsible for reporting the position and shape of an object in relation to the vehicle frame. Identification performs classification algorithms on the position and shape of a detected object to categorize it. These abilities are essential for local motion planning to compute collision-free trajectories.

2.2.2.1: Sensor Specification
The BDC platform employs the use of three Ibeo LUX laser scanner sensors (Figure 2-7) for both front and rear obstacle detection and identification. These 4-plane laser scanners have a 650ft/200ft detection/identification range over an 85° horizontal/3.2° vertical field of view (FOV) [104]. The scanners provide 3-dimensional point cloud measurements at a frequency of up to 50Hz with a 1.57in distance resolution and 0.125° horizontal/0.8° vertical FOV resolutions for increased object visibility at longer ranges. The LUX also uses multi-pulse laser technology to provide reliable readings in the presence of atmospheric clutter such as dust and rain. Additionally, the sensors perform on-board, low-level object identification to distinguish if an object is valid, part of the ground, or atmospheric clutter. An Ibeo LUX Fusion system (Figure 2-8) is also available to synchronize data between multiple Ibeo LUX sensors. The synchronization can be used to describe data from the sensors in a single frame of reference and also to consolidate data between sensors with overlapping fields of view.
The BDC platform incorporates two overlapping, forward-facing LUX laser scanners (Figure 2-9) and one rear-facing LUX laser scanner (Figure 2-9). The forward facing scanners are mounted in the front bumper of the vehicle, positioned for a 119° horizontal field of view with 51° of overlap starting 1ft in front of the vehicle (Figure 2-11). The rear facing scanner is center-mounted below the rear bumper, providing an 85° horizontal field of view (Figure 2-11). The LUX Fusion system is also included to manage the multiple and partially overlapping sensors on the vehicle. The fusion system is additionally connected to the TORC Localization Board (Section 2.2.1) through a CAN network as it uses INS orientation data to supplement object classification in the event of vehicle pitch and roll. The fusion system is mounted directly to the vehicle’s equipment rack located in the trunk (Figure 2-12). The fusion system provides synchronized object point clouds and associated data directly to the vehicle’s onboard computer (Section 2.3) through an Ethernet connection. Figure 2-13 displays the connection architecture for the object detection and identification system.
Figure 2-9: Mounting configuration for 2x forward-facing Ibeo LUX sensors.

Figure 2-10: Mounting configuration for 1x rear-facing Ibeo LUX sensor.

Figure 2-11: Obstacle Detection and Identification Horizontal Field of View
Figure 2-12: Ibeo LUX Fusion System Mounting Configuration. Ibeo ECU (A) and Ibeo Sync Box (B)

Figure 2-13: Obstacle Detection/Identification System Connection Architecture
2.2.3: Active Road Detection

Active road detection permits the vehicle to distinguish road surface from the ground in front of the vehicle in real time. This can be used to supplement local motion planning with road position information for more accurate lateral positioning of the vehicle in the driving lane. Active road detection must, at minimum, report useable road surface to the system. However, active road detection can also be used for additional information such as lane detection.

2.2.3.1: Sensor Specification

The BDC platform uses two Allied Vision Technologies Prosilica GC1290C cameras (Figure 2-14) in a stereo vision mounting scheme for active road detection. The GC1290C cameras provide an RGB or HSL 1280x960 pixel image at up to 32 frames per second over a standard gigabit Ethernet network [105]. The cameras are additionally outfitted with a 4.2mm auto-iris lens, providing a 116.7° field of view with auto-stabilization of light levels sensed by the camera. This auto-stabilization simplifies machine vision by mitigating effects due to different lighting conditions on image color.

![Figure 2-14: Allied Vision Technologies Prosilica GC1290C Camera](image)

2.2.3.2: Platform Implementation

The GC1290C cameras are mounted inside of the BDC vehicle cabin at the top corners of the windshield (Figure 2-15), facing the road surface in front of the vehicle. Currently, active road detection only uses the left-side camera; however future developments will enable use of both cameras with depth perception for more advanced road detection. The single left camera is mounted facing directly forward of the vehicle, as shown in Figure 2-16. An Ethernet connection links the cameras to the vehicle’s onboard computer described in Section 2.3. Figure 2-17 displays the connection architecture for the active road detection system.
Figure 2-15: Mounting Configuration for 2x Allied Vision Technologies Prosilica GC1290C.

Figure 2-16: Active Road Detection Horizontal Field of View

Figure 2-17: Active Road Detection System Connection Architecture
2.3: TORC Robotics Semi-Autonomous Systems

The TORC Robotics semi-autonomous systems provide the Blind Driver Challenge® research platform with the ability to perceive the environment for the driver and calculate safe vehicle trajectories that the driver can follow. This system uses a wide array of software implemented on a central computing system to compile sensor information (Section 2.2) into an environmental model and plan trajectories through the model itself. This subsection describes both the computer hardware and software components that make up the platform’s semi-autonomous systems.

2.3.1: Computer Hardware

The Blind Driver Challenge® implementation of the TORC ByWire XGV™ includes a TORC Robotics dual computer system for providing the semi-autonomous functions essential to the driver assistance algorithms proposed in this work. The dual computer system directly connects to all environmental perception sensors and consolidates it into world models. The dual computer can additionally perform global and local motion planning using information from the world model and assigned mission. The primary purpose of this system is to provide non-visual interface control software with real time world model information and planned local trajectories.

2.3.1.1: System Specification

The Blind Driver Challenge® vehicle’s semi-autonomous features are centrally implemented through a TORC Robotics AutonoNav™ dual computer system installed inside of the vehicle (Figure 2-18). The AutonoNav™ includes two separate PCI-104 based computers with specifications outlined in Table 2-2. The computers are also JAUS interoperable and act as nodes within the BDC system over the vehicle’s Ethernet network, as discussed in Section 2.4.

Figure 2-18: TORC Robotics AutonoNav™ Dual Computer System
Table 2-2: Onboard Dual Computer System Specifications (Per Computer)

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel SP9300 Core 2 Duo (2.26 GHz)</td>
</tr>
<tr>
<td>Chipset</td>
<td>Intel GS45/ICH9M</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR-1067 (4GB)</td>
</tr>
<tr>
<td>Hard Drive</td>
<td>80GB SATA Solid State Drive</td>
</tr>
<tr>
<td>External Ports</td>
<td>2x Gigabit Ethernet</td>
</tr>
<tr>
<td></td>
<td>4x USB 2.0</td>
</tr>
<tr>
<td></td>
<td>1x Serial</td>
</tr>
<tr>
<td></td>
<td>1x VGA Video Out</td>
</tr>
<tr>
<td>Internal Ports</td>
<td>1x PCI-104 Expansion Slot</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>50W, 12-14VDC Input</td>
</tr>
<tr>
<td>Physical Dimensions</td>
<td>3.5&quot;H x 8.6&quot;W x 12.5&quot;D</td>
</tr>
</tbody>
</table>

2.3.1.2: Platform Implementation

The AutonoNav™ system is installed within the vehicle's main equipment rack, as shown in Figure 2-19. As previously described, the primary function of the dual computer system is to perform all semi-autonomous duties of the Blind Driver Challenge® research platform. The system is able to accomplish this because it is composed of a series of JAUS interoperable software modules written by TORC Robotics to perform semi-autonomous functions based on raw data from the environmental perception sensors (Section 2.2). For detailed information on the dual computer system software applications, including environmental perception and motion planning, see Sections 2.3.2.1 and 2.3.2.2.

The AutonoNav™ dual computer system interfaces with the vehicle’s central communication network (Section 2.4) through four separate Ethernet lines. The inclusion of several Ethernet lines allows for rapid data communication between the AutonoNav™ system and the array of semi-autonomy and drive-by-wire equipment. As this system is the central processor of all sensor data (with the addition of global and local motion planning), it is required that perception data and associated communications can be transmitted and received in parallel for increased speed. The full connection architecture of the AutonoNav can be found in Figure 2-20.
Figure 2-19: Mounting Configuration for TORC Robotics AutonoNav™ System (A)

Figure 2-20: TORC Robotics AutonoNav™ Dual Computer System Connection Architecture
2.3.2: Software Components

The Blind Driver Challenge® Research Platform hosts a variety of semi-autonomous software developed by TORC Robotics that is utilized by the driver assistance algorithms presented in this paper to aid the driver in perceiving and navigating their environment. A comprehensive description of this software can be found in [10] and [106], however a summary of the key software components are provided in this section as they are extensively referenced in Section 3 and 0. The pertinent software components can be logically grouped into Environmental Perception, Motion Planning, and Safety and Health Monitoring; each of which is summarized in the following subsections.

2.3.2.1: Environmental Perception

The driver assistance algorithms contributed in this paper act solely on required driving actions derived from vehicle trajectories calculated by TORC Robotics motion planning software (Section 2.3.2.2). In order to calculate these vehicle trajectories, the motion planning software must have complete knowledge of the environment it is operating within. The environmental perception software provides this service by interfacing with the platform’s environmental perception sensors (Section 2.2) and compiles the raw sensor data into a local environmental model around the vehicle. The local model is built based upon data from Localization, Object Detection/Classification, Road Detection, and Vehicle Interface software. The functionality of these software components are summarized in the following subsections.

2.3.2.1.1: Localization

The primary purpose of the Localization software component is to provide the vehicle’s instantaneous global position and local position. The global position describes the vehicles location in a global sense through latitude and longitude coordinates, while the local position describes the location in meters in a local frame originated at where the vehicle was first turned on. Both the global and local positions additionally include roll, pitch, and yaw (heading) orientation readings in their respective coordinate frames as well as RMS accuracy for both position and orientation. These measurements are taken directly from the Inertial Navigation Solution sensor system described in Section 2.2.1 and are compiled into JAUS interoperable (Section 2.4.4) messages that allow other software components to read these measurements from the Localization software. A description of these output messages are provided in Table 2-3 and Table 2-4.
Table 2-3: Report Global Pose JAUS Message. See [106] and [107] for complete message detail.

<table>
<thead>
<tr>
<th>Field #</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Units</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Presence Vector</td>
<td>U16</td>
<td></td>
<td></td>
<td></td>
<td>Presence of subsequent fields.</td>
</tr>
<tr>
<td>2</td>
<td>WGS84 Latitude</td>
<td>Scaled I32</td>
<td>Deg</td>
<td>-90</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>WGS84 Longitude</td>
<td>Scaled I32</td>
<td>Deg</td>
<td>-180</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Altitude ASL</td>
<td>Scaled I32</td>
<td>m</td>
<td>-10000</td>
<td>35000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Position RMS</td>
<td>Scaled U32</td>
<td></td>
<td>0</td>
<td>100</td>
<td>Indicates position data validity.</td>
</tr>
<tr>
<td>6</td>
<td>Roll</td>
<td>Scaled I16</td>
<td>rad</td>
<td>-(\pi)</td>
<td>(\pi)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Pitch</td>
<td>Scaled I16</td>
<td>rad</td>
<td>-(\pi)</td>
<td>(\pi)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Yaw</td>
<td>Scaled I16</td>
<td>rad</td>
<td>-(\pi)</td>
<td>(\pi)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Rate RMS</td>
<td>Scaled U16</td>
<td></td>
<td>0</td>
<td>(\pi)</td>
<td>Indicates rate data validity.</td>
</tr>
<tr>
<td>10</td>
<td>Time Stamp</td>
<td>Time Stamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-4: Report DUC Local Pose JAUS Message. See [106] for message detail.

<table>
<thead>
<tr>
<th>Field #</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Units</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Version</td>
<td>U8</td>
<td></td>
<td></td>
<td></td>
<td>Message version number.</td>
</tr>
<tr>
<td>2</td>
<td>Presence Vector</td>
<td>U8</td>
<td></td>
<td></td>
<td></td>
<td>Presence of subsequent fields.</td>
</tr>
<tr>
<td>3</td>
<td>Local X</td>
<td>Scaled U32</td>
<td>m</td>
<td>-500000</td>
<td>500000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Local Y</td>
<td>Scaled U32</td>
<td>m</td>
<td>-500000</td>
<td>500000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Local Z</td>
<td>Scaled U32</td>
<td>m</td>
<td>-500000</td>
<td>500000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Position RMS</td>
<td>Scaled U32</td>
<td>m</td>
<td>0</td>
<td>100</td>
<td>Indicates position data validity.</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>Scaled U32</td>
<td>m</td>
<td>0</td>
<td>10000000</td>
<td>Nothing transformation parameter.</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>Scaled U32</td>
<td>m</td>
<td>0</td>
<td>10000000</td>
<td>Easting transformation parameter.</td>
</tr>
<tr>
<td>9</td>
<td>Roll</td>
<td>Scaled U16</td>
<td>rad</td>
<td>-(\pi)</td>
<td>(\pi)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Pitch</td>
<td>Scaled U16</td>
<td>rad</td>
<td>-(\pi)</td>
<td>(\pi)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Yaw</td>
<td>Scaled U16</td>
<td>rad</td>
<td>-(\pi)</td>
<td>(\pi)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Attitude RMS</td>
<td>Scaled U16</td>
<td></td>
<td>0</td>
<td>(\pi)</td>
<td>Indicates attitude data validity.</td>
</tr>
<tr>
<td>13</td>
<td>Time Stamp</td>
<td>Time Stamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.3.2.1.2: Object Detection and Classification
The Object Detection and Classification software component perceives objects within a certain vicinity of the vehicle and classifies them as either static or dynamic objects. Both static and dynamic objects are described in the local frame (Section 2.3.2.1.1) by a center position and bounding rectangle, however the dynamic objects additionally include a velocity vector. This information is obtained by first reading raw sensor data from the on-board laser range finders (Section 2.2.2) and then is classified based on the local velocity of the object. Objects that are stationary in the local frame are classified as static obstacles, while objects that are non-stationary in the local frame are classified as dynamic obstacles. Instantaneous lists of all static and dynamic obstacles are compiled into JAUS interoperable (Section 2.4.4) messages that allow other software components to read these measurements from the Object Detection and Classification software. A definition of these messages can be found in [106].

2.3.2.1.3: Road Detection
The Road Detection software provides the Blind Driver Challenge® Platform with passive and active nearby road information. The passive information cross-references the vehicle’s current GPS location with the mission map (Section 2.6) to find the current RNDF map position as well as a list of nearby roads within the RNDF map. The software component compiles this information into JAUS interoperable (Section 2.4.4) messages [106] that allow other software components to access the nearby RNDF map information. It is important to note, however, that this method relies entirely on the accuracy of the localization sensors; thus disturbances such as poor GPS signal quality directly affect the perceived location of the vehicle on the road. This becomes a significant factor when driving on roads with relatively small lateral lane width as the global position of the vehicle directly dictates the lateral position of the vehicle within the lane. For example, if poor GPS signal provides ±2m of accuracy in any direction, the car may perceive itself as centered within the lane when in actuality it is centered on the lane boundary.

The active road detection attempts to improve road position readings by augmenting the passive road detection methods with information taken from the platforms forward facing camera sensors (Section 2.2.3). The active road detection method transforms images from the left camera into a 2-dimensional bird’s eye view of the forward road surface through homography. The software then applies a 4-bit cost map to the projected image and attempts to classify each cell between a range of 0 (no danger) and 16 (high danger). This enables the platform to generally distinguish road surface from other surfaces in optimal lighting conditions that help improve the road position reading from the passive road detection method. The calculated cost map is made available to other software components through JAUS interoperable (Section 2.4.4) messages defined in [106].
2.3.2.1.4: Vehicle Interface

The Vehicle Interface contains the series of software components that provides the information specific to the vehicle discussed in Section 2.1.2.1. The contributions of this paper primarily use the Primitive Driver and Velocity State Sensor to find the instantaneous steering wheel angle and speed of the vehicle, respectively. The instantaneous steering wheel angle provides a percentage range between -100% (-537° left steering lock) to +100% (+537° right steering lock). The velocity state simply describes the longitudinal speed of the vehicle in meters per second. The vehicle interface software runs directly on the ByWire XGV™ Embedded Controller (Section 2.1.2.2) and provides the necessary information to other software components through JAUS interoperable (Section 2.4.4) messages shown in Table 2-5 and Table 2-6 below. Fields with crossed-out descriptions are not supported by the TORC ByWire XGV™.

Table 2-5: Report Wrench Effort JAUS Message. See [106] and [107] for complete message detail.

<table>
<thead>
<tr>
<th>Field #</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Units</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Presence Vector</td>
<td>U16</td>
<td></td>
<td></td>
<td></td>
<td>Presence of subsequent fields.</td>
</tr>
<tr>
<td>2</td>
<td>Propulsive Linear X</td>
<td>Scaled I16</td>
<td>%</td>
<td>-100</td>
<td>100</td>
<td>Throttle percentage.</td>
</tr>
<tr>
<td>3</td>
<td>Propulsive Linear Y</td>
<td>Scaled I16</td>
<td>%</td>
<td>-100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Propulsive Linear Z</td>
<td>Scaled I16</td>
<td>%</td>
<td>-100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Propulsive Rotational X</td>
<td>Scaled I16</td>
<td>%</td>
<td>-100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Propulsive Rotational Y</td>
<td>Scaled I16</td>
<td>%</td>
<td>-100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Propulsive Rotational Z</td>
<td>Scaled I16</td>
<td>%</td>
<td>-100</td>
<td>100</td>
<td>Steering wheel position percentage.</td>
</tr>
<tr>
<td>8</td>
<td>Resistive Linear X</td>
<td>Scaled U8</td>
<td>%</td>
<td>0</td>
<td>100</td>
<td>Brake percentage.</td>
</tr>
<tr>
<td>9</td>
<td>Resistive Linear Y</td>
<td>Scaled U8</td>
<td>%</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Resistive Linear Z</td>
<td>Scaled U8</td>
<td>%</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Resistive Rotational X</td>
<td>Scaled U8</td>
<td>%</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Resistive Rotational Y</td>
<td>Scaled U8</td>
<td>%</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Resistive Rotational Z</td>
<td>Scaled U8</td>
<td>%</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-6: Report Velocity State JAUS Message. See [106] and [107] for complete message detail.

<table>
<thead>
<tr>
<th>Field #</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Units</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Presence Vector</td>
<td>U16</td>
<td></td>
<td></td>
<td></td>
<td>Presence of subsequent fields.</td>
</tr>
<tr>
<td>2</td>
<td>Velocity X</td>
<td>Scaled I32</td>
<td>m/s</td>
<td>-65.534</td>
<td>65.534</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Velocity Y</td>
<td>Scaled I32</td>
<td>m/s</td>
<td>-65.534</td>
<td>65.534</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Velocity Z</td>
<td>Scaled I32</td>
<td>m/s</td>
<td>-65.534</td>
<td>65.534</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Velocity RMS</td>
<td>Scaled U32</td>
<td></td>
<td>0</td>
<td>100</td>
<td>Indicates velocity data validity.</td>
</tr>
<tr>
<td>6</td>
<td>Roll Rate</td>
<td>Scaled I16</td>
<td>rad/s</td>
<td>-32.767</td>
<td>32.767</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Pitch Rate</td>
<td>Scaled I16</td>
<td>rad/s</td>
<td>-32.767</td>
<td>32.767</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Yaw Rate</td>
<td>Scaled I16</td>
<td>rad/s</td>
<td>-32.767</td>
<td>32.767</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Rate RMS</td>
<td>Scaled U16</td>
<td></td>
<td>0</td>
<td>π</td>
<td>Indicates rate data validity.</td>
</tr>
<tr>
<td>10</td>
<td>Time Stamp</td>
<td>Time Stamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.3.2.2: Motion Planning

The driver assistance algorithms presented in this paper act solely on required driving actions derived from vehicle trajectories calculated by the TORC Robotics motion planning software summarized in this subsection. The software is provided with a mission that outlines a series of destinations and monitors the driver-vehicle environment (Sections 2.2 and 2.3.2.1) to plan/re-plan trajectories through the environment to each destination. These trajectories are calculated through an algorithm that first determines the appropriate travel speed and subsequently finds, to a certain degree of optimality, a trajectory to take that will safely navigate the vehicle towards the next destination. The travel speeds are determined by a piece of software called the Speed Limiter, while the trajectory is found through a Trajectory Search. The trajectory is then encoded in a Motion Profile, which is communicated directly to the driver assistance algorithms contributed in this paper. The following subsections outline the high-level functionality of the Speed Limiter, Trajectory Search, and Motion Profile encoding used by the platform motion planner.

2.3.2.2.1: Speed Limiter

The primary purpose of the Speed Limiter is to determine a safe maximum speed that is derived from the mission speed (Section 2.6) combined with dynamic obstacles and stop points in the vicinity of the vehicle. Under normal driving conditions, the Speed Limiter commands the mission speed for any particular road within the map definition file (Section 2.6). Certain roads contain stop points, such as stop signs or traffic lights, in which the Speed Limiter will gradually reduce the maximum speed as it approaches
the stop points. This software additionally accounts for following other vehicles in traffic by monitoring dynamic obstacles in front of the vehicle. The Speed Limiter decreases the maximum speed as required to follow a lead vehicle that may be either stopped or traveling at speeds lower than that specified by the mission speed.

2.3.2.2.2: Trajectory Search
The Trajectory Search software is responsible for determining the actual trajectory that the vehicle should take given a certain environment and mission (Section 2.6). The search begins by examining the maximum speed at which it may travel provided by the Speed Limiter (Section 2.3.2.2.1). It then creates a cost-based occupancy map that is derived from Lane Detection and Obstacle Detection/Classification information. The occupancy map has a grid resolution of 20cm\(^2\), and covers an area that extends 30m in front of the vehicle and 15m to the rear and sides for a total area of 1350m\(^2\). Once an occupancy map is updated, the software then begins an A* trajectory search (Section 1.4.2.1) to find an appropriate set of future actions that safely satisfy the goals of the mission. The search evaluates candidate solutions and chooses the best based upon many factors, including heuristics such as lateral lane deviation, mission time, and safety. Under normal operation, the trajectory search is able to re-plan trajectories at a variable rate between 2 and 8 Hz.

2.3.2.2.3: Motion Profile Encoding
Motion Profile Encoding uses a vehicle model to transform a desired trajectory (Section 2.3.2.2.2) into a discretized series of future steering and speed commands otherwise known as a Motion Profile. A Motion Profile contains a list of consecutive future motions, each of which describes a single steering command, speed command, and command time interval. The steering command describes a desired turning curvature as well as the necessary curvature time-rate-of-change to utilize when traveling from the current curvature to the desired curvature. The speed command describes a desired longitudinal speed as well as the maximum acceleration that can be used when changing from the current speed to the desired speed. Lastly, the time interval describes the length of time for which the given steering and speed commands are valid. Figure 2-21 depicts the steering and speed commands of a single motion within a motion profile.

These motions, when executed sequentially in time, describe the necessary steering wheel, throttle, and brake actions that must be taken to realize the trajectory found in the motion planning trajectory search. Figure 2-22 displays an example of a desired trajectory and its corresponding Motion Profile transformation. Each motion within the Motion Profile describes a particular piece of the trajectory in time. Under normal operation, the Motion Profile will contain at least one motion and will typically describe 2-3 seconds of future steering/speed motion commands.
Figure 2-21: An example of a standard motion within a Motion Profile.

Figure 2-22: An example of a complete Motion Profile and the corresponding desired trajectory it was derived from.
The Motion Profiles calculated by Motion Planning are compiled into a JAUS interoperable (Section 2.4.4) message (Table 2-7) that is sent directly to the Non-Visual Interface Driver (Section 3 and 0) each time a new trajectory is calculated. Although the Motion Profiles typically contain 2-3 seconds of command data, only between 0.125-0.5 seconds of the data is actually used due to the 2-8Hz variable rate at which new trajectories are re-planned.

Table 2-7: Set Motion Profile JAUS Message. See [106] for message detail.

<table>
<thead>
<tr>
<th>Field #</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Units</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Version</td>
<td>U8</td>
<td></td>
<td></td>
<td></td>
<td>Message version number.</td>
</tr>
<tr>
<td>2</td>
<td>Motion Count (N)</td>
<td>U8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3+5n</td>
<td>Desired Velocity</td>
<td>Scaled I16</td>
<td>m/s</td>
<td>-0.5335</td>
<td>0.5335</td>
<td></td>
</tr>
<tr>
<td>4+5n</td>
<td>Maximum Acceleration</td>
<td>Scaled I16</td>
<td>m/s²</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>5+5n</td>
<td>Arctangent of Desired Curvature</td>
<td>Scaled I16</td>
<td>atan(1/m)</td>
<td>-π/2</td>
<td>π/2</td>
<td>Arctangent evenly distributes resolution of Scaled I16.</td>
</tr>
<tr>
<td>6+6n</td>
<td>Curvature Time Rate-of-Change</td>
<td>Scaled I16</td>
<td>1/(m-s)</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7+7n</td>
<td>Time Duration</td>
<td>U16</td>
<td>ms</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.3.2.3: Health Monitoring

The Blind Driver Challenge® Platform utilizes a special software, named the Health Monitor, to continuously oversee the operating condition of the other software components and take safety precautions in the event that a piece of software or hardware malfunctions. In this particular implementation, the Health Monitor will put the platform in a “Pause” state if sufficient system errors occur. The “Pause” state brings the vehicle to a controlled stop similarly to the “Emergency Stop” state; however it does so at a slightly less rate of deceleration. Once the system error has been either automatically or manually repaired, the Health Monitor will place the vehicle back in the “Resume” state and release the brake so that the system may continue normal operation.

The Health Monitor utilizes a series of JAUS interoperable (Section 2.4.4) messages [106] to request and send information related to the health of a particular software component. Each software component is responsible for updating the Health Monitor through JAUS messages with the most recent errors and warnings so that the appropriate action may be taken. In the event that a component experiences total failure (and thus cannot send updated error messages), other software components that rely on the failed component will report an error on its behalf.
2.4: System Network and Communication

The Blind Driver Challenge® platform incorporates a central Ethernet communication network that interconnects the many different components described in Section 2. This network facilitates the communication of many different types of data, including raw sensor information, component commands, and motion planning trajectories. The network is composed of a main Ethernet switch and uses a JAUS interoperable software scheme to define the method in which components communicate with each other. The network also includes an Operator Control Unit (OCU) that provides local or remote access to the Blind Driver Challenge® platform for interacting with onboard components. These network features are characterized in more detail throughout this subsection.

2.4.1: Network Switch and Wireless Access

The BDC research platform uses an Ethernet-based central communication network for all interactions between the ByWire XGV™, semi-autonomous systems, environmental perception sensors, and the non-visual interface computer. The central Ethernet network is connected through a D-Link DGS-1224T switch with wireless connection that is supplied through a Ubiquiti PicoStation2HP (Figure 2-23). The DGS-1224T switch [108] includes 24 Gigabit Ethernet ports and Jumbo Frame support for larger, heavy-load networks. It also allows virtual LAN support to separate JAUS communications networks from sensor-computer networks. IGMP snooping additionally monitors network traffic to streamline the multicast data broadcasting often seen in JAUS dynamic discovery. The PicoStation2HP wireless access point provides a transparent wireless connection between the inner vehicle communication network and an external Operator Control Unit (OCU) computer for remote vehicle monitoring and control. The PicoStation2HP can supply up to 1000mW of transmit output power for long-range, high bandwidth (11Mbps) connections [109].
2.4.2: The Operator Control Unit (OCU)

The Operator Control Unit (OCU) is a computer that enables remote monitoring and control of the Blind Driver Challenge® Platform. This computer interacts with the platform with a direct or wireless connection to the central communication network. The OCU incorporates a specially designed “Blind Driver OCU” program developed by TORC Robotics to provide high level control and monitoring of the platform. The OCU software includes functions for controlling mission-specific parameters, such as the map/destination and maximum speeds in driving areas. It also provides a driving performance indicator that describes the total percentage of time the driver spend inside of the road/lane boundary as well as the number of collisions made with obstacles (for simulation, Section 2.5). The OCU implements a dashboard feature that displays the most pertinent platform data to the operator, consisting of the current mission and associated travel speeds as well as an organized list of system errors. The remaining information provided by the OCU is found in organized, tabulated panels for quick and simple access during experiments. Currently, the OCU is installed and operated on a Lenovo® ThinkPad™ Edge 0301J9U laptop (specifications can be found in Table 2-8).

Table 2-8: Operator Control Unit (OCU) Computer Specifications

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Core i3-390M (2.66GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>Mobile Intel HM55 Express</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR3-1066MHz (2GB)</td>
</tr>
<tr>
<td>Hard Drive</td>
<td>250GB SATA 5400 RPM</td>
</tr>
<tr>
<td>External Ports</td>
<td>10/100/1000 Base T Ethernet</td>
</tr>
<tr>
<td></td>
<td>4x USB 2.0</td>
</tr>
</tbody>
</table>
2.4.3: Network Diagram

Figure 2-24 displays the network connection architecture for the entire Blind Driver Challenge® platform. This diagram ultimately serves as a list of which components of the platform communicate over the central Ethernet network. The data shared over these connections are described in detail within each component characterization presented in Sections 2.1, 2.2, 2.3, Section 3, and 3.5.

![Network Diagram]

Figure 2-24: System Network Architecture of the Blind Driver Challenge® Platform.
2.4.4: JAUS Interoperability

The Joint Architecture for Unmanned Systems (JAUS) is an architecture proposed by the United States Department of Defense for use in unmanned systems that focuses on software topology and communication protocols [110] [111] [107]. The purpose of JAUS is to establish a framework for unmanned systems developers to follow that will promote software organization and provide independence from code design and application hardware/technology. Software written using the JAUS framework will interface with other similar software through pre-defined, standardized communication methods; making the internal workings of the software that are particular to the application and designer transparent to the rest of the system. This transparency ensures code developed for one particular application is compatible with all other applications that follow the JAUS framework.

The Blind Driver Challenge® platform uses a complete JAUS implementation on the TORC ByWire XGV™, the semi-autonomy dual computer, and the non-visual interface computer. JAUS provides standardized and well-defined methods for communication between these components, promoting elegant software interaction and enabling efficient software development and debugging. The particular JAUS implementation used by the Blind Driver Challenge® platform will be discussed; however an overview of JAUS functionality is first presented.

2.4.4.1: JAUS Functionality Overview

The primary functionality of JAUS is provided through the designation of a standardized system topology and message specification. The system topology creates a logical hierarchy of software components to promote code organization. The message specification describes how each different software module interacts with the rest of the software system. System topologies and message definitions are comprehensively discussed in the JAUS Reference Architecture Version 3.3 [110] [111] [107], however summarizations are provided in the following subsections.

2.4.4.1.1: System Topology

Before discussing actual JAUS system topology, it is important to understand the lowest level of differentiation in robotic functions on which the topology is based. Any given robotics system can be ultimately divided into modules that are either “strictly software” or “software/hardware” based. Software/hardware modules are combinations of hardware that perform a specific function with associated interface/control software for that particular piece of hardware. An example of a software/hardware module is a GPS sensor and its interface software that reads the GPS data from the sensor itself. A strictly software module is a piece of software that performs tasks unrelated to the interface/control of a piece of hardware. An example of a strictly software module is a motion planner: it takes in environmental perception data from software/hardware
modules (which gather raw data from the sensor) and plans an appropriate path for the robot to traverse. Figure 2-25 portrays the difference between the strictly software and software/hardware modules.

Figure 2-25: Software/Hardware Module (top) and Strictly Software Module (bottom) functionality.

The JAUS system topology provides a method to organize these modules into a hierarchy composed of five different levels: System, Subsystem, Node, Component, and Instance. While the modules only truly exist at the lowest level (Instance), they are grouped together through the higher levels. The component level is strictly defined; however the node, subsystem, and system levels are specified by the designer for each particular application. A description of each level is provided on the next page, while a graphical representation of the topology can be seen in Figure 2-26.

**Instance:** The actual module that performs a specific duty. Instances allow differentiation between modules that have the same purpose. An example of an instance occurs when two or more sensors of the same type are placed on a system; an instance is created for each of the independent sensor modules.
Component: A grouping of module instances that is responsible for certain low-level capabilities. Components allow differentiation between module instances with different purposes. Some examples of components are a GPS sensor module, a mission planner module, and a robotic arm manipulator module.

Node: A group of all modules (including Components/Instances) that performs a specific high-level function. Nodes allow differentiation between groups of components with different overall collective purposes. Usually, each separate computer or controller on a system acts as a node. Examples of a node include a vision processor, a drivetrain controller, or an operator control unit.

Subsystem: A group of all nodes that performs an even higher-level specific function. Subsystems allow differentiation between groups of nodes with different overall collective purposes. An example of the use of subsystems is in the application of multiple coordinated unmanned ground vehicles: each separate vehicle is labeled as a different subsystem.

System: The comprehensive grouping of all subsystems for a particular robotics application. Only one system exists as it represents the collective robotics application itself. An example of a system is an entire group of unmanned ground vehicles constructed to map an unknown area.

Figure 2-26: Graphical depiction of the JAUS system topology.
Each module instance within a system is identified by an address consisting of identification numbers that are assigned to each topology level. These numbers range from 1 to 254, and are combined in an address of the form:

$$\text{SubsystemID.NodeID.ComponentID.InstanceID}$$

(Example: 173.024.215.001)

With a nested topology such as this, JAUS can address up to 254 subsystems, $254^2$ nodes, $254^3$ components, and $254^4$ instances in a single system. The identification number 255 is reserved for broadcasting communications to all identification numbers existing in the system at that level. This addressing scheme provides a method for differentiating and identifying separate modules during inter-module communication through the JAUS message specification.

### 2.4.4.1.2: Message Specification

The main advantages offered by JAUS interoperability comes directly from the usage of the JAUS message specification. This specification defines the method and format of all communication between any two pieces of software (modules). For example, if a motion planning module requires GPS data from a GPS module, the two modules would communicate through the JAUS message specification. Strict definitions of the messaging methods and formats maximize compatibility and efficiency by forcing software to be independent from the designer, particular hardware, or particular applications.

The message specification is composed of three main definitions: message classes, message composition, and message routing. There are seven different message classes, the most important of which are the command class, the query class, and the inform class. The command class of messages allows one software module to command an action from another software module. This action can be in software, hardware or both. The query class allows one module to query another module for some sort of information or data. The inform class is used to answer a query from a module with the requested information or data. Other classes include event setup and notification, node management, and experimental messaging. Experimental messaging allows designers to formulate application specific messages that are not included in the pre-defined JAUS message set. An additional form of messaging worth mentioning is the Service Connection, which allows one module to “subscribe” to data from another module by requesting a continuous stream of data rather than having to poll for the data each time via a query request.

Message composition defines the actual formatting of a message sent between software modules. Each message is composed of a 128-bit header and 4096-bit maximum data field. The header provides information for message routing, data information and flags, and message metadata. The data field follows the header with
the actual bits of data or information to be transmitted. This data can be anything from GPS coordinates to a command for an unmanned vehicle to turn in a certain direction. Figure 2-27 depicts the JAUS message composition, including header and data fields.

![Figure 2-27: JAUS Message Composition](image)

Message routing defines how message sources and destinations are identified as well as how the messages are actually transferred between the source and destination. Each and every module, whether it is the source or destination of a message, is assigned a unique four-field address as defined by the system topology. When building a message, the module completes the source/destination addresses in the header field. The message is then sent to a specialized component, called the Node Manager, which is included in every node. The purpose of the Node Manager is to act as a centralized message router; all messages from modules belonging to a particular node are sent to the node’s Node Manager, which then forwards the messages to the appropriate destination module. When messages are sent across two different nodes, the source module’s Node Manager forwards the message to the destination’s Node Manager, which finally delivers the message to the destination module. For communications between subsystems, a designer-defined Communicator component is used; however this function is normally added to the Node Manager in practice. In this manner, all communication is performed through the system’s Node Managers. Figure 2-28 depicts the overall communication paths between JAUS messages.
The Node Manager also performs dynamic system discovery: it continuously monitors a list of current running modules, including their identification address and their capabilities. Each instance of a module generates a heartbeat message at a specified frequency, allowing the Node Manager to discover when a new module comes online or a current module goes offline. The modules also store their own capabilities, which can be queried by other modules looking for a certain function within the system. This is further simplified by the use of pre-defined component addressing: certain functions and capabilities are linked to particular component ID numbers. For example, a velocity state sensor will always use component ID 42. This allows the system to perform a certain degree of dynamic discovery based solely on module addresses rather than having to query the capabilities of the module itself.

![Figure 2-28: JAUS Communication Diagram.](image)

The combination of the JAUS system topology and message specification allows for a highly organized, efficient, and self-maintaining autonomous system. The standardized method for software development also significantly simplifies extension of system capabilities by establishing a comprehensive set of compatibility rules. As a standardized research platform, the TORC ByWire XGV™ employs JAUS interoperability to take full advantage of these features.

### 2.4.4.2: Blind Driver Challenge® JAUS Implementation

The BDC Platform implements JAUS to create an organized system topology and standardize the communication between TORC Robotics and Virginia Tech software. The BDC application uses a topology that defines each separate BDC platform as its own subsystem; allowing operation of the two platforms simultaneously as a single system. Within each platform subsystem, there exists a node for each computer: the ByWire XGV™ embedded controller, the semi-autonomy dual computer system, the non-visual interface computer, and the Operator Control Unit computer. These nodes implement components and instances of software and software/hardware modules that are associated with the functionalities provided by the particular computer. For
example, the ByWire XGV™ embedded controller node provides access to and control of all features supplied by the ByWire XGV™. A high-level JAUS topology of the Blind Driver Challenge® system is pictured in Figure 2-29. Low-level descriptions of each node within the BDC platform subsystem are given in the following subsections.

Figure 2-29: High-Level JAUS Topology of the Blind Driver Challenge® Platform
2.4.4.2.1: The TORG Robotics ByWire XGV™

The TORG ByWire XGV™ (Section 2.1) incorporates a full JAUS implementation to maximize system organization, stability, and compatibility for the end-user's research applications. The platform's central embedded controller follows the JAUS specification in terms of topology and message specification. These particular implementations are described in detail in the TORG ByWire XGV™ Manual [98], however summaries are provided in the following paragraphs.

The ByWire XGV™ in its entirety acts as a single node in the system, allowing the end-user to establish subsystem organization to match particular application needs. The ByWire XGV™ node exists on the platform's central embedded controller and contains six components (each containing one instance): the Node Manager, the Motion Profile Driver, the Primitive Driver, the Signals Driver, the Velocity State sensor, and the Error Manager. The capabilities of the first five components are described previously in Section 2.1.2.1. The sixth component, the Error Manager, simply maintains a list of errors within the ByWire XGV™ that are used to detect when an emergency stop is required due to system failure.

The ByWire XGV™ uses the JAUS messaging specification for all communication between components, both internal and external to the ByWire XGV™ node. JAUS messages internal to the platform’s central embedded controller are communicated via shared memory. External JAUS messages between end-user components and the ByWire XGV™ are communicated via UDP [112] through the central embedded controller’s Ethernet port. Messaging over UDP Ethernet allows the end-user to connect every computing node within the system over a simple Ethernet network.

Overall, the ByWire XGV™ supports 54 different JAUS messages, 12 of which are experimental (not specified by the JAUS RA [107]). This provides the end-user with a highly customizable platform, offering total control over the XGV™ as well as unrestricted access to its data. A comprehensive definition of supported JAUS messages, including both standard and experimental messages, can be found in the TORG ByWire XGV™ Manual [98] for implementation by the end-user.

2.4.4.2.2: The TORG Robotics Semi-Autonomy Dual Computer System

The semi-autonomy dual computer system (Section 2.3) actually exists as two separate nodes within the BDC system: one node for each computer of the dual computer layout. One node (Alpha) is primarily responsible for sensor interfacing and motion planning, while the second node (Bravo) is responsible for additional sensor interfacing and health monitoring. These two separate nodes work together to provide world models and trajectory planning for use with the non-visual interfaces. A detailed description and message listing of the Alpha and Bravo node components can be found in [106], however a high-level summary is provided in this work.
The Alpha node consists of the Object Detection and Motion Planning components. Object Detection directly interfaces with the laser range finders (Section 2.2.2) on the platform to determine the location and classification of objects within the vicinity of the vehicle. This component maintains a list of all obstacles found and gives other components of the platform subsystem access to this list through JAUS messaging. The Motion Planning component uses data from all perception-related components on the Alpha and Bravo nodes to calculate motion trajectories for the platform to follow. These trajectories are made available to the non-visual interface control system through experimental JAUS messages.

The Bravo node consists of the Localization, Road Detection, and Health Monitoring components. Localization collectively interfaces with the TORC Robotics Localization Board and the Inertial Navigation System of the platform (Section 2.2.1). This component maintains the GPS position and orientation of the platform and supplies this information to the network via JAUS messaging. Road Detection interfaces with the platform’s camera sensors, Localization, and the mission file to determine road and lane position relative to the vehicle’s coordinate reference frame. The road and lane positions are made available through experimental JAUS messages. The Health Monitor component receives health information from all components within the platform subsystem to control emergency stop/pause states of the ByWire XGV™. All health information is communicated to the Health Monitor through experimental JAUS messages.

2.4.4.2.3: The Non-Visual Interface Computer
The Non-Visual Interface Computer (Section 3.5) acts as a JAUS node that implements the Non-Visual Interface Driver (NVID) component. The NVID is the piece of software that uses semi-autonomy data from the platform to control the non-visual interfaces and provide information to the driver. The driver is able to pull important perception and planning data from the semi-autonomy dual computer system and additionally provide health monitoring information all through JAUS messaging. Full detail of the NVID functionality can be found in Section 4 and 0 of this work.

2.4.4.2.4: The Operator Control Unit (OCU)
The OCU computer acts as a node within the platform subsystem that incorporates the OCU component (Section 2.4.2). The OCU component uses both standard and experimental JAUS messages to supply the operator with access to platform data and control of platform features. Displayed platform data is continuously updated with JAUS service connections to help the operator monitor the subsystem in real-time. The OCU component provides control over other components using the JAUS command set specification and retains the highest level of command authority below the Health Monitor component.
2.5: The Research Platform Simulator

The Research Platform Simulator is a comprehensive software simulation of the Blind Driver Challenge® platform, including the TORC ByWire XGV™, semi-autonomy systems, and non-visual interface systems. The simulator creates a 3D and fully customizable environment that a virtual BDC platform exists and operates in. The simulator runs transparently with the semi-autonomous and non-visual interface software, allowing quick and efficient testing and debugging during development efforts. The simulator also serves as a training station for drivers so that they may gain experience with how to use the non-visual interfaces to operate the vehicle.

The main piece of software behind the simulator is the TORC Robotics SimVironment™ program. SimVironment™ creates the 3D environment surrounding the simulated BDC platform, including roads, lanes, and static/dynamic obstacles (Figure 2-30). These environments can be modeled after existing real-world road networks or can be fabricated into non-existing road networks for testing different driving scenarios. SimVironment™ interacts with a simulated TORC ByWire XGV™ program and the TORC semi-autonomy perception software components to allow full and transparent operation of the BDC platform within the 3D environment.

![SimVironment™ 3D Vehicle Simulation Software](Figure 2-30: TORC Robotics SimVironment™ 3D Vehicle Simulation Software)
The transparency of the research platform simulator comes directly from the system’s JAUS interoperability. Even though some of the software modules are internally different (such as the simulated ByWire XGV and perception sensor modules), the collective cooperation of all system modules is unaffected. This capability is achievable since all cross-software communication is standardized through JAUS messaging. For example, the operation of the Non-Visual Interface Driver (NVID) remains exactly the same with real and simulated ByWire XGV™ platforms due to the fact that the NVID communicates with both of these platforms through the same JAUS messages. As long as real and simulated modules share the same input and output message sets, the internal workings of the module may be radically different without affecting the overall method of cooperation in the system.

Since the simulator and actual BDC platform are transparent to cooperative software through JAUS interoperability, the same exact Non-Visual Interfaces, Driver software, and Controller hardware are used on the simulator. This allows true NVI testing and training as the researcher/operator can utilize the interfaces within the simulator in the same exact manner as on the actual BDC platform. The only true difference between the NVI system integration in the platform and on the simulator is the mounting scheme; the driver sits in a normal desk-chair on the simulator as compared to the driver’s seat of the actual platform. For this reason, the DriveGrip adapter box is removed from the headrest mount and is attached to the back of the simulator desk-chair using Velcro®.

As the simulated platform is driven on a computer, a Logitech® Driving Force™ GT force-feedback steering wheel/pedal controller is used to input steering and velocity commands from the driver. The wheel turns 900° lock-to-lock and includes a centering spring effect to closely match the operation of the BDC platform steering wheel. While the Driving Force™ GT steering wheel does provide simple feedback to the driver, the more complex feedback associated with actual driving, such as lateral and longitudinal g-forces, is not provided. This actually poses the only influential discrepancy between the simulator and real-world driving. When driving the actual BDC platform, the driver uses not only the tactile inputs from the non-visual interfaces, but also the tactile and inertial inputs that come from the motion and operation of an automobile. These inertial inputs from the vehicle actually play an important role in the driver’s operation of the vehicle. For example, when driving in the simulator, the driver’s only feedback for actual speed (SpeedStrip communicates speed error) is the position of the gas or brake pedal. The positions of these pedals only provide a slight sense of constant speed and no information on the actual level of speed. Although the simulator suffers from this downfall, it still provides a reasonably accurate environment for testing the NVIs and training drivers, as will be discussed in Section 3 and 0 of this work.
2.6: Route and Mission Development

In order to assist the driver with steering and velocity control of the vehicle, Blind Driver Challenge® platform requires knowledge of a goal position and map of how to reach the goal position. This information is provided to the system through a Route Network Definition File (RNDF) [10], which contains a GPS-based map and list of road networks to traverse to arrive at a destination. Ultimately, the platform’s motion planner uses this information to generate vehicle trajectories that follow the desired roads to the goal destination. The RNDF files are created by the operator using the Route & Mission Development Tool™ (RMDT), a piece of software created by TORC Robotics. This subsection summarizes the RNDF and RMDT™; however more detail can be found in the RMDT™ Manual [113].

2.6.1: The Route Network Definition File (RNDF)

The responsibility of the RNDF is to provide the BDC platform's motion planner with a map of roadways to take to arrive at a target destination. This RNDF is comprised of three main components: a geo-referenced aerial image of the operating area, a map of routes that are traversable by the platform, and a mission that defines which routes will be taken to arrive at a target destination. The geo-referenced aerial image is a top-down view of an area on the surface of the earth that is coupled with geological data, such as latitude, longitude, and elevation. These images, also called orthoimagery, are orthorectified to remove photographic distortions and ensure that each pixel of the image can be uniformly assigned to a latitude, longitude, and elevation value [114]. The orthoimages of areas to be driven by the platform provide highly accurate geo-spatial information on the positions of roads and lanes within that area.

Routes are collections of road segments and zones that the platform has the capability of driving on. The segments and zones are defined relative to orthoimagery of a certain area, and thus are ultimately defined in latitude and longitude coordinates. Road segments are collections of control points and splines that define the center of a road. The road segments also contain information about the number, direction, and width of lanes within the road. Segments have entry and exit points that join separate segments together, creating longer segments, corners or intersections. The exit points can be configured as stop points between segments to account for stop signs. Zones are polygonal areas that are used to map open areas, such as parking lots. Zones have entry/exit points and can also include parking spots. Missions consist of a start and target destination as well as a list of which routes to take to reach the target. The list of routes is simply a list of which road segments and zones to follow along a path to the target. The combination of the mission and map of routes provides the platform with a coarse list of GPS waypoints to achieve coupled with lane information for fine waypoint control.
2.6.2: The Route & Mission Development Tool™ (RMDT)
The TORC Robotics RMDT™ is a software tool used to develop RNDF files for guidance control of the Blind Driver Challenge® platform. The tool is very simple to use and operates in two different stages: Map creation and Mission creation. First, maps are generated that include all of the road segments and zones that make up the routes that can be traversed by the vehicle. Missions are then specified that define a start and target position as well as which routes to take in reference to the map generated.

Maps of an area are created with the use of orthoimagery of the particular area. The RMDT™ permits the user to open an orthoimage an essentially draw RNDF road segments and zones over the orthoimage. This allows the user to create routes directly off of a detailed image map that includes GPS latitude and longitude data. Road segments are created by adding GPS control points to the orthoimage map and creating adjustable splines that link the points to account for the various types of road curvature. The RMDT™ permits the user to specify the number, width, and direction of lanes within each segment. Zones are similarly created in that control points are superimposed over the orthoimage; however these points are linked together as polygons instead of splines. Each segment and zone has a control point that can be used as an entry or exit point. The RMDT™ allows the user to connect entry and exit points between routes also using splines. Exit points can additionally be configured as stop points to account for stop signs at intersections. Upon complete specification of all the routes that make up a map, the map is then saved for use by the motion planner and mission creator.

Missions are created solely in reference to the map that they are intended to work within. The RMDT™ loads a previously created map that includes the orthoimage of the area and the routes created for that particular area. The user can then specify start, waypoint and destination points on the map in reference to the control points of each road segment/zone. This defines the target location and exactly which road segments and/or zones to travel when attempting to reach the target location. The missions are then saved along with the map that they are dependent upon. Once the map and missions are complete, they are compiled into the Route Network Definition File (RNDF) which is directly used by the BDC platform motion planner. At runtime, the appropriate RNDF for the platform to follow is selected by the operator through the Operator Control Unit (Section 2.4.2).
Section 3: The Non-Visual Interface System

The Non-Visual Interfaces (NVIs) for driver assistance are primarily responsible for communicating important driving information to the driver in a manner that does not require the use of vision. The NVIs are a cornerstone technology for the Blind Driver Challenge®; however they can also be used as supplemental sources of information for sighted drivers as well. The current BDC system incorporates the use of two main non-visual interfaces: DriveGrip and SpeedStrip. The improvement and performance analysis of these interfaces, as well as the development of the Non-Visual Interface Controller (NVIC), are contributions of this paper and are presented in this subsection.

3.1: DriveGrip – Steering Assistance Interface

DriveGrip is a Non-Visual Interface that communicates steering information through vibrational haptics [8]. As described in Section 1.2.1.3, DriveGrip incorporates the use of vibrational motors positioned over the fingers of each hand using a glove-mounted configuration. The vibration sensations created from this interface are able to cue the driver with primary and/or supplemental steering information based on calculated trajectories from local motion planning. A prototype of the interface was proposed and tested in [8]; however this work presents a significant improvement of the DriveGrip system; transitioning it from a simple prototype to an advanced, dependable, and complete implementation.

3.1.1: Functionality

As described in [8], the function of the DriveGrip interface is to communicate the instantaneous steering wheel angle error to the driver through vibratory haptics imposed on the index, middle, ring and little finger of each hand. The instantaneous steering wheel angle error communicated through this interface is given by the simple equation:

$$\theta_E = \theta_R - \theta_A$$  \hspace{1cm} (3.1)

where $\theta_E$ is the instantaneous steering wheel angle error; $\theta_R$ is the instantaneous reference steering wheel angle specified by the transformation algorithms presented in Section 4 and Section 5; and $\theta_A$ is the instantaneous actual steering wheel angle measured by the Primitive Driver (Sections 2.1.2.1 and 2.3.2.1.4). It should be noted that, in this work, all steering wheel angles will be communicated in degrees.

In order to communicate the instantaneous steering wheel error to the driver, DriveGrip associates each finger with a range of error magnitude and each hand with an error direction. With regard to magnitude, vibrations on the index finger indicate the lowest range of error magnitude while vibrations on the little finger indicate the highest range of error magnitude. With regard to direction, vibrations on the fingers of the left hand indicate positive (clockwise) error while vibrations on the fingers of the right hand
indicate negative (counter-clockwise) error. With this configuration, the driver is instructed to steer towards the hand that the vibration is on at an amount indicated by the particular finger on which the vibration is present. In this manner, the human acts as a simple Proportional controller. The following table dictates how each particular vibrato-tactile element is controlled in both a four finger and two finger vibration scheme:

Table 3-1: DriveGrip Vibro-Tactile Element Control Table for four finger and two finger vibration configurations.

<table>
<thead>
<tr>
<th>Hand Location</th>
<th>Finger Location</th>
<th>Activated Range Four Finger</th>
<th>Activated Range Two Finger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Little</td>
<td>(-\infty &lt; \theta_e \leq -T_L)</td>
<td>(-\infty &lt; \theta_e \leq -T_L)</td>
</tr>
<tr>
<td></td>
<td>Ring</td>
<td>(-T_L &lt; \theta_e \leq -T_R)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Middle</td>
<td>(-T_R &lt; \theta_e \leq -T_M)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>(-T_M &lt; \theta_e \leq -T_I)</td>
<td>(-T_L &lt; \theta_e \leq -T_I)</td>
</tr>
<tr>
<td>Right</td>
<td>Index</td>
<td>(T_I &lt; \theta_e &lt; T_M)</td>
<td>(T_I &lt; \theta_e &lt; T_L)</td>
</tr>
<tr>
<td></td>
<td>Middle</td>
<td>(T_M &lt; \theta_e &lt; T_R)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Ring</td>
<td>(T_R &lt; \theta_e &lt; T_L)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Little</td>
<td>(T_L &lt; \theta_e &lt; +\infty)</td>
<td>(T_L &lt; \theta_e &lt; +\infty)</td>
</tr>
</tbody>
</table>

where \(T_L, T_R, T_M,\) and \(T_I\) are the steering wheel angle error range thresholds for the little, ring, middle, and index fingers, respectively.

The positioning of the vibro-tactile elements on the fingers enables maximum tactile receptiveness by the driver as the hands are the most sensitive part of the human body to vibration [115]. A second advantage of this configuration is that it can communicate information to the human with minimal influence from human desensitization. The motors operate in a simple binary scenario; there is no variation of vibrational frequency and no formation of vibratory patterns. Only one vibro-tactile element in the entire DriveGrip interface vibrates at any given time; making it easy for the human driver to discern the vibration position and thus the error magnitude and direction even in periods of prolonged usage.

3.1.2: Interface Development

As the original DriveGrip interface and associated control hardware presented in [8] were prototypes, this paper proposes a complete redesign of the system to drastically improve performance, efficiency, and robustness. The redesigned system implements the same basic functionality as that of [8] on a completely new set of hardware that can fully unlock the potential of the interface. The new system was fabricated, assembled, and rigorously tested to meet the expected level of performance required for our driver assistance research application.
3.1.2.1: **Layout**

Figure 3-1 presents the functional layout of the redesigned DriveGrip system. The layout consists of five separate components: the vibratory motors, wiring harness, connector, mounting glove and LED indicators. The vibratory motors are located on the proximal phalanges of the index, middle, ring, and pinky fingers on each hand. This particular positioning harnesses the haptic sensitivity of the finger without monopolizing the use of the driver’s fingertips. These motors present information to the driver by creating a binary vibration on a single finger at a time. This dramatically decreases the conscious effort required by the driver to understand information from the vibro-tactile cues in real-time and makes the system significantly less susceptible to desensitization over prolonged periods of use.

However, the resistance to desensitization also presents a tradeoff in the form of low information resolution: the configuration only supports a maximum of 9 unique cues. Although it may seem more advantageous to supply the driver with higher-resolution cues, resolution itself plays a significant role in the human’s ability to understand information from the interface without becoming overwhelmed. Presenting the driver with data at a higher resolution requires more cognitive effort to understand, leading to confusion and increased rates of mental fatigue. The DriveGrip system must operate in real-time and in prolonged periods of use; therefore it was necessary to sacrifice high resolution communication and increase human reliability. This configuration was designed to match the human’s cognitive and physical capabilities, ensuring that the driver can dependably perceive and understand information from the interface at all times.

![Figure 3-1: Redesigned DriveGrip System Layout. Components include: Vibratory Motors (red), Standardized Connector (blue), Wiring Harness (Black), and Mounting Glove (gray).](image-url)
For observational and training purposes, a small red light emitting diode (LED) was placed on each vibration motor. Whenever a particular vibratory motor is powered, the attached LED turns on as an indicator for observers during experiments and training. This allows the researchers to visually monitor the vibratory pattern that the driver is feeling without physically interfering with the driver's operation of the vehicle.

The DriveGrip system layout also incorporates a connector (Figure 3-1) for attaching signal wires between the mounting gloves and the BDC platform. The connector is designed to allow the gloves to be easily connected and disconnected from the platform in the event that the driver is exiting or entering the vehicle while wearing the DriveGrip system. The connector also standardizes how the system attaches to the platform so that different sets or versions of gloves may be interchanged between platforms.

The wiring harness used to connect the vibratory motors to the standardized connector is laid out to minimize the effects of wire fatigue due to hand movement. The wires are sewn onto the glove and connected to components at strategic points, creating large bending radii to minimize the stress applied to the wires under high-cycle bending.

The DriveGrip system layout components are collectively attached to the mounting gloves shown in Figure 3-1. These mounting gloves are worn by the driver and comfortably enforce positioning and contact of the vibratory motors over the primary phalanges of the hand. They also aid in the positioning and protection of the standardized connector and wiring harness to minimize fatigue on the wire and associated connections. Due to the configuration of the vibratory motors on the hand, the gloves are only required to be half-fingered and thus leave the fingertips free for normal use by the driver.

3.1.2.2: Component Specification

The redesign of the DriveGrip system was centered on the use of a new type of vibration motor, called the LilyPad Vibe Board from SparkFun® Electronics (Figure 3-2). The LilyPad Vibe Board is composed of a 10mm diameter enclosed cellphone vibration motor attached to a 20mm diameter printed circuit board (PCB) designed for attachment to clothing. The PCB also includes soldering pads so the end-user can select the proper wiring application. The motor operates on 5VDC (220mW) and creates relatively large vibration amplitudes of 0.8G. The entire package is 20mm in diameter, 4.2mm thick, and weighs only 1.2g. The features of the LilyPad Vibe Board provide many significant advantages for use in the new DriveGrip interface:

- Small, lightweight package that can be easily attached to a glove for comfortable positioning over the driver's fingers.
- High vibration amplitude for guaranteed driver perception at a low power cost.
- Open soldering pads for custom, high fatigue-limit wiring applications.
Figure 3-2: The SparkFun® LilyPad Vibe Board used in the DriveGrip Interface. (http://static3.watterott.com/2008488-2.jpg, 2012, Used under fair use, 2012.)

The LED’s used for signifying LilyPad Vibe Board actuation are standard RadioShack® brand 3mm Red LEDs. These LEDs supply a 5.0MCD with a 25° viewing angle in a small T1 package for high visibility and negligible effects on glove comfort and layout. The LEDs have an operating voltage of 2.25VDC and consume a maximum of 60mW of power each.

The standardized connector used in the redesigned system is built from the Leviton® Quickport Cat5e Wall Jack (Figure 3-3). This connection scheme uses Ethernet-based wiring to connect DriveGrip to the BDC platform, creating two main advantages: a wide variety of flexible, high-fatigue limit cable already exists; and the RJ45 standardized wire connectors implement snap-in features for quick connection and disconnection. Ethernet wires contain 8 conductors that can support 577mA of current each, supplying enough channels and power transmission for each of the four 100mA-max motor/LED combinations per glove. The Leviton® RJ45 Quickport connector used for this Ethernet-based scheme is actually meant for use in home wall-jacks; however we have repurposed it for use as the standardized connector for the DriveGrip glove.

Figure 3-3: Leviton® QuickPort Cat5E Wall Jack used for the DriveGrip Glove standard connector. (http://www.aartech.ca/images/cache/9bed8ee11bd34a45a84acc100c8b1af5.jpg, 2012, Used under fair use, 2012.)
The wiring harness is comprised of two different types of wire: Assmann Electronics 28AWG 9-conductor stranded ribbon cable and RadioShack® 22AWG stranded hookup cable. The ribbon cable is extremely flexible, lightweight, and durable against breaks from high-cycle fatigue. This type of cable is used to connect the motor grounds and positive common lead to the standardized RJ45 connector. The heavier, 22AWG stranded cable is used to connect the common positive leads between motors. In this configuration, the 22AWG wire does not experience significant deformation due to bending. Thus, the 22AWG wire can provide support for portions of the smaller 28AWG ribbon cable in low-flexibility locations within to the proximity of the motors.

The mounting glove chosen for this application is the Harbinger® Power Weightlifting Glove (Figure 3-4). This weightlifting bar glove is composed of a half-finger design with focus on grip comfort and fabric breathability. This particular design is easily repurposed for gripping a steering wheel in the DriveGrip application. The mesh fabric on the back of the glove acts as an excellent base for gluing and sewing components into place while still maintaining user comfort. The glove also features a Velcro® wrist strap for simple placement and removal from the hand without having to significantly pull on areas of the glove and possibly damage mounted components and/or wiring.

![Figure 3-4: Harbinger® Power Weightlifting Gloves used as a mounting glove for the DriveGrip system.](http://www.sportslabstores.com.au/images/regent0155.jpg, 2012, Used under fair use, 2012.)

### 3.1.2.3: Electrical Schematic

Figure 3-5 displays a wiring schematic for connecting the LilyPad Vibe Board/LED combinations to the RJ45 Ethernet connector on both left and right gloves. Each LilyPad/LED combination is composed of a series LED and 150Ω resistor in parallel with the LilyPad Vibe Board. This parallel combination allows the LilyPad and the LED to share the power and ground leads, ensuring that the LED emits light when the LilyPad is powered and vibrating. As all LilyPad/LED combinations share a common
+5VDC power lead, the 150Ω resistor is used to drop the 5VDC to ~2.25VDC and ensure no more than 26mA of current passes through the LED.

The cathode of each LilyPad/LED combination is separately connected to the RJ45 connector of each glove. While the combinations share a common anode, the cathodes act as the control channels for enabling/disabling current flow through the components. The common anode is connected to pin 7 on the RJ45 connector, while the four LilyPad/LED combination cathodes are connected to pins 8, 6, 4, and 2. Pins 1, 3, and 5 are left disconnected.

![Figure 3-5: DriveGrip Left/Right Glove Wiring Schematic](image)

### 3.1.2.4: Assembly Design

The assembly design of the DriveGrip system is very important as it helps decrease wire fatigue and additionally helps protect components on the glove. This design begins with the assembly of the enclosed vibration components for each finger. Figure 3-6 shows the wiring layout of the LilyPad Vibe Board/LED combination for the left and right hand configurations. As seen on the front of the combination, the heavier 22AWG wire is soldered on top (not through) of the LilyPad Vibe Board anode soldering through-hole and directly protrudes from the bottom of the PCB. The smaller 28AWG ribbon cable wire also attaches to the top of the cathode soldering through-hole, however this wire wraps around the vibration motor on top of the board to act as tensile relief and protrude the wire at the same location of the anode wire. The anode/cathode wire bundle protrusion assists in decreasing the effect of fatigue as the wires collectively support themselves against bending deformation.
As seen on the rear of the LilyPad/LED combination board, the LED anode and cathodes are soldered on top of the reverse side of the anode/cathode solder through-holes, effectively placing it in parallel with the LilyPad Vibe Board. The 150Ω resistor is placed between the LED and LilyPad cathodes to drop the 5VDC supplied to the LilyPad down to ~2.25VDC for the LED.

Figure 3-6: SparkFun® LilyPad Vibe Board and Indicator LED Wiring/Soldering Layout for the left and right gloves.

After the LilyPad/LED vibration components are properly wired, they are completely covered with hot-glue on both sides. As depicted in Figure 3-7, the hot-glue holds components such as the LED and drop-down resistor in place and also protects soldered connections on both sides of the PCB. The hot-glue is shaped with a mostly flat bottom to maximize the contact pad between the LilyPad Vibe Board and the driver’s finger. The wiring bundle exiting the vibration component is also hot-glued to strengthen the bundle against tensile and high-cycle-bending stresses.

Figure 3-7: DriveGrip vibration component with hot-glue shell for protection, shape and comfort.
The fully enclosed combination of the LilyPad Vibe Board, LED, and associated hardware now make up a single vibration component. As previously shown in the DriveGrip high-level layout (Figure 3-1), a vibration component must be placed over the index, middle, ring and pinky fingers of each hand. This is achieved by hot-gluing vibration components directly onto the finger segments of the Harbinger® half-fingered glove. The finger segments of the glove are elastic and hold the vibration component against the top surface of each finger. The standard RJ45 connector is also hot glued into place on the back of the glove as indicated in Figure 3-1 as well.

The main wire harness that links the RJ45 connector to each individual vibration component is constructed from the Assmann Electronics 28AWG ribbon cable. The 9-conductor ribbon cable is stripped down to 5-conductor, keeping the polarity indicated wire intact. The polarity indicated wire is designed to connect the common anode chain of the vibration components to the RJ45 connector. The remaining 4 wires connect the RJ45 connector to the separate cathodes of each vibration component. Figure 3-8 displays how the ribbon cable wire stem is routed to each vibration component and the RJ45 connector. The heavier 22AWG wire is used to connect the common anodes between each vibration component, forming a U-shape that does not pass below the base knuckle of the proximal phalanx. The 28AWG ribbon cable runs along the bottom of the U-shapes, stripping a conductor off of the ribbon at each vibration component. The single ribbon cable conductor attached to each vibration component is taped to the 22AWG wire for extra support and wire management. This particular design creates a single wire harness that runs just above the bending axis of the base knuckle so that it is not stretched over the top of base knuckle when the fingers are clenched. The U-shaped loop between each vibration component creates large bending radii to decrease wire fatigue when fingers are clenched independently from one another.

![Figure 3-8: DriveGrip main wire harness layout.](image-url)
The main wire stem is routed between the vibration component grouping above the base knuckles down to the RJ45 connector between the index finger and the thumb. This routing incorporates 1.5" of extra length in the ribbon cable wire to create flexibility when the hand is clenched around a steering wheel. The Leviton® QuickPort RJ45 Ethernet wall jacks terminate wires using an IDC contact. The ribbon cable stem enters the RJ45 connector housing, wherein each conductor is separated and punched into the appropriate IDC contact previously described in the wiring schematic (Figure 3-5).

The final stage of assembly consisted of sewing elastic cloth over the tops of the vibration components as well as the RJ45 connector. The cloth provided additional protection to components as well as added to the aesthetic appeal of the DriveGrip system. The cloth sewn over the RJ45 connector additionally covers half of the main ribbon cable wire stem between the connector and the vibration components. This helps keep the wire stem located between the thumb and index finger, as well as protecting the slacked wire stem from getting snagged or pulled.

3.1.2.5: Completed Redesign

The final DriveGrip redesign can be seen in Figure 3-9, and Figure 3-10. Careful inspection of these figures shows how the assembly design described in the previous subsection was implemented in the actual application.
3.1.3: Platform Implementation

The DriveGrip gloves are attached to the BDC platform through two separate Ethernet cables. This implementation was designed to position the platform-side Ethernet connections directly over the shoulders of the driver so that the Ethernet wires would run along the length of the arm directly into the RJ45 port on the DriveGrip gloves. In order to position the Ethernet wires in this manner, a mounting plate and breakout box is attached just below the headrest of the driver seat. The following subsections will describe the breakout box, mounting plate, and Ethernet wiring layout in more detail.

3.1.3.1: DriveGrip Breakout Box

The DriveGrip system includes a breakout box to convert the DB9 connection from the Non-Visual Interface Controller (Section 3.4) into the dual Ethernet cable connection used with the DriveGrip gloves. Figure 3-11 shows a schematic of the breakout box that includes connector types and pin-outs. The breakout box is composed of a female DB9 IDC input with 2x RJ45 outputs housed within a RadioShack® 4"x2"x1" project enclosure. The RJ45 connectors are the exact same connectors used on the DriveGrip
gloves (Figure 3-3). The breakout box can be seen attached to the DriveGrip seat mount in Figure 3-12 of the next subsection.

**Figure 3-11: DriveGrip Breakout Box Schematic**

### 3.1.3.2: DriveGrip Seat Mount

The DriveGrip seat mount (Figure 3-12) acts as a base for the DriveGrip breakout box and also positions the DriveGrip Ethernet wires directly over the shoulders of the driver. The mount is made from 0.25in thick aluminum for strength and has two through-holes so that the support rods of the headrest can be inserted through the mount directly into the seatback. The support rods and positioning of the headrest hold the mount in place on the driver’s seat (Figure 3-13). The DriveGrip breakout box is attached to the center of the mount, which provides a through-hole for the DB9 connection between the breakout box and the Non-Visual Interface Controller (Section 3.4). Adjustable Ethernet wire clamps are located on the outward protrusions of the mount, and are positioned at an angle to help manage the Ethernet wire location around the arms of the driver. The Ethernet cables exit the rear of the breakout box and then pass underneath the adjustable clamps to hold them in position and at the desired length.

**Figure 3-12: DriveGrip System Headrest Mount with Breakout Box (Rear View)**
3.1.3.3: Wiring Layout

The Ethernet cables that link the DriveGrip breakout box and gloves are made from 5ft Video Products Inc. CAT5e Super Flat Cable and are positioned along the driver’s arms. This particular wiring scheme allows for routing the highly flexible, 0.08in thick flat Ethernet wires from the BDC platform to the driver’s hands in a non-restricting way. The wires run down the length of the arm from the shoulder to the hand, and are attached to the proximity of the elbow with an elastic Velcro® band. This band helps keep the length of wire close to the arm of the driver, allowing normal motion such as hand-over-hand driving without restriction from the wires.

The DB9 connection between the DriveGrip breakout box and the Non-Visual Interface Controller (Section 3.4) is routed through the bottom of the mount and down the front of the seatback. It is then pushed through the gap between the seat and seatback where it is finally connected to the controller underneath the seat. The wire located on the front of the seatback is later obscured by the SpeedStrip non-visual interface (Section 3.2). This layout allows for safe and hidden routing of the DB9 connection without any modification to the vehicle.
3.1.4: Performance Analysis

As described in earlier in this section, the DriveGrip interface is responsible for communicating the instantaneous steering wheel angle error to the driver via vibrotactile components placed on the fingers of each hand. The driver subsequently uses this information to control the position of the steering wheel and thus control the heading of the vehicle as it navigates through an environment. This subsection documents a series of experiments conducted to test the ability of the driver to track reference steering wheel angle signals using only the information communicated by the DriveGrip interface and without performance degradation due to desensitization.

NOTE: The results presented in this work were obtained from test subjects in collaboration with the National Federation of the Blind and cooperate with the Virginia Tech Institutional Review Board (IRB) Human Subject Research Protocol.

3.1.4.1: Step Reference Signal Tracking

In this experiment, drivers were asked to track a range of randomized step reference steering wheel angle signals to analyze and compare the transient response properties elicited by each of the four DriveGrip operation configurations (DOCs) defined in Table 3-2. The drivers were subjected to small (30°), medium (60°), and large (180°) amplitude step references and the corresponding responses were recorded. A total of 10 responses for each step amplitude and DOC combination were recorded to determine the average driver response characteristics induced by each DOC.

Table 3-2: DriveGrip Operation Configurations (DOCs).

<table>
<thead>
<tr>
<th>Operation Configuration</th>
<th>Vibration Configuration</th>
<th>Error Threshold Tolerance</th>
<th>Error Threshold Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFST</td>
<td>Four Finger</td>
<td>Strict</td>
<td>T_F: 0.1°, T_M: 10°, T_R: 20°, T_L: 40°</td>
</tr>
<tr>
<td>FFLT</td>
<td>Four Finger</td>
<td>Lenient</td>
<td>T_F: 5°, T_M: 10°, T_R: 20°, T_L: 40°</td>
</tr>
<tr>
<td>TFST</td>
<td>Two Finger</td>
<td>Strict</td>
<td>T_F: 0.1°, T_M: N/A, T_R: N/A, T_L: 25°</td>
</tr>
<tr>
<td>TFLT</td>
<td>Two Finger</td>
<td>Lenient</td>
<td>T_F: 5°, T_M: N/A, T_R: N/A, T_L: 25°</td>
</tr>
</tbody>
</table>

While the experiments conducted in this analysis were performed with several drivers, the results presented in this section are taken from a single driver that represents the average abilities of the drivers tested in terms of the transient properties discussed for each response shown. The choice of a representative driver for this discussion allows a more focused investigation on the abilities of the DriveGrip interface and not on the abilities of an interchangeable driver. In the interest of a more efficient analysis, a set of representative data from the recorded results will be shown in this section. The complete set of data recorded in these tests can be viewed in Appendix A.1.1 and will be compared with the representative data in this discussion.
Figure 3-14 displays the average (μ) driver response with standard deviation (σ) to a 60° step reference signal using each type of operation configuration listed in Table 3-2. The transient properties of each response were recorded and are displayed in Table 3-3. The “adjusted” settling time property shown in this table measures the time it takes for the response to settle within 1° of its final value once it enters and remains within ±40° of steering wheel angle error. This metric allows better comparison of operation configuration performance and will be explained in more detail in the discussion section.

![Graph](image)

Figure 3-14: Average driver response to a 60° step reference signal utilizing various DriveGrip operation configurations defined in Table 3-2.

Table 3-3: Transient characteristics of the driver responses plotted in Figure 3-14.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FFST</td>
<td>370 ms</td>
<td>860 ms</td>
<td>4.8°</td>
<td>1990 ms</td>
<td>-0.1° &lt; e_{ss} &lt; 0.1°</td>
</tr>
<tr>
<td>FFLT</td>
<td>370 ms</td>
<td>1230 ms</td>
<td>0°</td>
<td>990 ms</td>
<td>-0.6°</td>
</tr>
<tr>
<td>TFST</td>
<td>320 ms</td>
<td>1280 ms</td>
<td>0.8°</td>
<td>1250 ms</td>
<td>-0.1° &lt; e_{ss} &lt; 0.1°</td>
</tr>
<tr>
<td>TFLT</td>
<td>370 ms</td>
<td>840 ms</td>
<td>0°</td>
<td>630 ms</td>
<td>-0.2°</td>
</tr>
</tbody>
</table>

A quick observation of the step response results shown in Figure 3-14 seems to indicate that the driver behaves similarly to a second-order dynamic system with pure time delay. Further observation demonstrates that the operation configurations using strict error threshold tolerances imitate an underdamped system while the configurations using lenient tolerances conversely imitate an overdamped system. As shown in Table 3-3, the transient portion of each driver response can be quantified and compared by the driver’s time delay, rise time, overshoot, adjusted settling time, and steady-state error. Furthermore, close examination of these quantifications clearly demonstrate the functionalities designed into the DriveGrip interface defined in Section 3.1.1.
Let the discussion begin with an analysis of the time delay exhibited by each average driver response shown in Figure 3-14. The time delays of these responses are directly associated with the driver’s reaction time (RT), or the time it takes for the driver to cognitively perceive the stimulus created by the DriveGrip interface and make the required neuromuscular action [116] [117]. The reaction time of the driver is an important factor to consider when generating the reference signals for the driver to follow as the driver’s response will always be delayed by the reaction time. This will adversely affect the driver’s navigation performance unless the reaction time delay is anticipated by the reference signal.

Table 3-3 indicates that the time delay of the driver’s response for each DOC is relatively constant at an average time of approximately 357ms. Additionally, the 30° and 180° amplitude step responses shown in Appendix A.1.1 prove that the 357ms reaction time is constant for any amplitude step signal utilizing any of the four DOCs. This is made possible by the shape of the step reference itself; the reference value instantly changes and thus instantly activates the appropriate vibro-tactile element within the DriveGrip interface. Although the results shown in the previous subsection and Appendix A.1.1 suggest that the driver’s reaction time is relatively constant, future research could be conducted to prove that the reaction time may vary slightly depending on which finger and hand the stimulus was first imposed upon.

The rise times recorded in Table 3-3 present the first characteristic of the driver-steering system that distinguishes it from the standard second order system it seems to emulate so closely. For a second order system, one would expect the rise time to remain the same even when subjected to inputs of varying step amplitudes. However, the results in Table 3-3 and Appendix A.1.1 report rise times that differ depending on the DOC used and that increase with step amplitude. These peculiar trends are caused by the manner in which the instantaneous error is communicated to the driver through the DriveGrip interface. The drivers are trained to associate the vibration on each finger with a certain range of steering wheel angle error. During interaction with the DriveGrip interface and the steering wheel, the driver subconsciously re-associates the finger vibrations to constant values of steering wheel angular velocity. This is most clearly demonstrated by the individual re-plot of the 60° step FFLT response in Figure A-6; four constant steering wheel angular velocities can be seen in response to each of the four fingers vibrated. Because the little finger indicates the most instantaneous error, the driver subconsciously turns the wheel at the fastest angular velocity. When the error falls within the range of the ring finger, the driver (after a reaction time delay) responds by decreasing the angular velocity to a lower constant value. The same trend can be observed with each finger range until the no-vibration range is reached; at this point the driver realizes that the instantaneous steering wheel angle is zero and ceases turning the wheel.
Figure 3-14 portrays how the driver subconsciously associates each DOC with a different set of steering wheel angular velocities. The driver reacts to the FFST and TFLT DOCs similarly with the highest ranges of angular velocities, leading to the shorter and more desirable rise times shown in Table 3-3. The FFLT and TFST DOCs cause the driver to respond with a slower set of angular velocities, leading to the lower rise times quantified in the results. The $30^\circ$ and $180^\circ$ step responses shown in Appendix A.1.1 prove that this trend is constant for any amplitude step reference signal.

The presence of overshoot and oscillation in the driver's responses are directly related to the utilization of strict error threshold tolerances. This can be observed in Figure 3-14: the FFST and TFST DOCs exhibit oscillatory behavior while the FFLT and TFLT DOCs do not. Overshoot and oscillations are caused by strict error threshold tolerances because the no-vibration range is compressed by such an amount that, at higher steering wheel angular velocities, the driver cannot react to the no-vibration stimulus in a timely manner. This causes the no-vibration range to be easily overshot when the driver is turning the wheel above a certain angular velocity. This is most clearly demonstrated in an individual re-plot of the driver's response to a $60^\circ$ step reference using the FFST DOC, pictured in Figure A-5. As the driver approaches the reference steering angle for the first time, the driver turns the wheel at the angular velocity at which he or she normally associates with an index finger vibration. At first, this angular velocity is too fast for the driver to recognize the miniscule no-vibration range in time; thus the reference value is overshot. The driver detects the overshoot because the left index finger begins to vibrate and begins to turn with a decreased angular velocity in the opposite direction. As the driver continues to overshoot the reference value and cycles between left and right index finger vibrations, the driver subconsciously decreases the angular velocity each time the direction is switched in an attempt to finally settle in the no-vibration range. Eventually the angular velocity is decreased enough that the driver can react to the no-vibration stimulus in time and cease turning the wheel.

Now that a clear understanding of the cause behind response oscillations has been reached, an analysis of the overshoot and adjusted settling time results can be conducted. In accordance with the previous discussion, Table 3-3 reports that the FFST and TFST DOC responses exhibit overshoots of $4.8^\circ$ and $0.8^\circ$, respectively, while the FFLT and TFLT responses both exhibit an overshoot of $0^\circ$. The difference in overshoot between the FFST and TFST DOCs is accounted for by the drivers associations of different angular velocities to the two and four-finger vibration configurations. As previously described and shown in Figure 3-14, the TFST DOC response exhibits much slower angular velocities than the FFST DOC response. The lower angular velocity associated with the TFST DOC enables the driver to reverse the turning direction faster when the reference value is passed, thus leading to a smaller overshoot.
The adjusted settling time quantifications listed in Table 3-3 are an indirect measurement of the oscillatory behavior and can thus also be described by the driver’s associations of different angular velocities with different DOCs. Before this analysis continues, let it be re-stated that the “adjusted” settling time quantification of this experiment measures the time it takes for the driver’s response to settle within $1^\circ$ of its final value once it enters and remains within $\pm 40^\circ$ of the reference value. By adjusting the settling time measurement in this way, the varying rise time of each response is eliminated and thus the measurement can be compared equally amongst the different DOCs.

Similarly to overshoot, one can expect a faster adjusted settling time for the TFST DOC when compared to the FFST DOC because the lower angular velocities used by the driver create smaller overshoots and thus less oscillations. However, the FFLT and TFLT DOC responses in Figure 3-14 produce much lower adjusted settling times because the lenient error threshold tolerances do not cause any time-consuming oscillatory behavior. The shortest adjusted settling time is achieved with the TFLT DOC as the driver utilizes the fastest steering wheel angular velocity set and creates no oscillatory behavior due to the lenient error threshold tolerances.

While the strict error threshold tolerances of the FFST and TFST DOCs yield the disadvantage of overshoot, they conversely provide the advantage of smaller steady-state error (SSE) from the reference signal. After the driver initially overshoots the reference value, the error threshold tolerances force the driver to continuously correct towards the reference value until they are within $\pm T_i^\circ$ of the reference value. Thus, the error threshold tolerances directly define the range of the steady-state error for each response. The SSE quantifications in Table 3-3 exhibit this behavior, reporting smaller steady state errors for strict DOCs when compared with the lenient DOCs. It should be noted that the SSE of the FFST and TFST is reported as a range because the resolution of the steering wheel angle encoder was $0.1^\circ$: the same value as the $T_i$ strict error threshold tolerance. The actual reading reported $0^\circ$ steady state error; however this can mislead one to believe that the system exhibits total SSE rejection when that is not the case. It should also be noted that the relatively small SSE quantifications for the FFLT and TFLT DOCs are the convenient result of the driver’s reaction time to the no-vibration stimulus. In actuality, the SSE for these lenient DOCs could be anywhere within the $\pm T_i^\circ$ range ($\pm 5^\circ$ in this case).

Comparison of the steady-state error quantifications for each DOC requires an understanding of how influential the steady-state error is on the driver’s navigation performance, particularly in the case of laterally tracking the center of a driving lane. In order to decrease controller effort and avoid overloading the driver, DriveGrip only requires that the driver bring the steering wheel angle to within $\pm T_i^\circ$ of the reference value. However, even relatively small values for $T_i$ can lead to substantial changes in
the vehicle trajectory if the longitudinal speed is high enough. Therefore, the influence of \( T_l \) and thus the SSE must be investigated. Such an investigation can be performed using a simple Ackerman steering model taken from [118]:

\[
\begin{align*}
\dot{x} &= v \cos \varphi \\
\dot{y} &= v \sin \varphi \\
\varphi &= \frac{v}{L} \tan \left( \frac{\theta}{R} \right) 
\end{align*}
\]

where \( x \) and \( y \) are the lateral and longitudinal position of the vehicle with respect to the origin, \( \varphi \) is the vehicle’s heading measured from the positive \( x \)-axis, \( v \) is the instantaneous velocity of the vehicle, \( L \) is the vehicle wheelbase, \( \theta \) is the instantaneous steering wheel angle, and \( R \) is the steering ratio.

Using this model allows the range of lateral deviation from the centerline of a straight lane over time to be found. The ranges for strict \( (T_l = 0.1^\circ) \) and lenient \( (T_l = 5^\circ) \) error threshold tolerances can be calculated by inserting the \( T_l \) values into the instantaneous steering wheel angle for (3.4) and solving (3.2)-(3.4) iteratively over small time steps. Figure 3-15 and Figure 3-16 below plot the resulting calculation and demonstrate the growth of the allowable lateral lane deviation over time for different error threshold tolerances and speeds. The ranges shown in Figure 3-15 demonstrate that the strict error threshold tolerances keep the lateral deviation well within acceptable limits (less than 2cm) even at speeds up to 30m/s (65mph). Conversely, Figure 3-16 indicates that the lenient error threshold tolerance begins to allow significant lateral deviations at speeds greater than 15m/s (30mph).

![Figure 3-15: Lateral deviation ranges caused by the strict error threshold tolerance \( (T_l = 0.1^\circ) \) at different vehicle speeds.](image)
After the careful analysis of the transient properties of the driver’s responses using each of the DriveGrip operation configurations, the TFLT DOC has proven to elicit the most desirable response from the driver when tracking a step reference signal. A quick observation of Table 3-3 shows that the TFLT DOC average response exhibits the fastest rise time, no overshoot, and the lowest adjusted settling time. The average response for this DOC has been re-plotted with more detail in Figure 3-17 to highlight how its functionality elicits such a response from the driver.

Figure 3-16: Lateral deviation ranges caused by the lenient error threshold tolerance ($T_T = 5^\circ$) at different vehicle speeds.

Figure 3-17: Average driver response to a $60^\circ$ step reference signal utilizing the TFLT DOC defined in Table 3-2.
Although the TFLT DOC exhibits the best transient properties, it is crippled at high vehicle speeds due to the large range of steady-state error permitted by the lenient error threshold tolerance. As previously shown in Figure 3-16, the large range of SSE is directly related to a large range of allowable lateral lane deviation. For this reason it is suggested that, at higher speeds, the tolerance switch to a more strict value to minimize lateral deviation and maintain safe navigation of the vehicle. While the stricter tolerance will comparably elicit less favorable transient properties, one must consider that much smaller changes in the reference steering wheel angle will occur at higher speeds. With such small changes in reference angles, the comparison between the transient characteristics of the strict and lenient tolerances will become negligible.

3.1.4.2: Ramp Reference Signal Tracking

In this experiment, drivers were asked to track a range of various ramp reference steering wheel angle signals to analyze and compare the response properties elicited by each of the four DriveGrip operation configurations (DOCs) defined previously in Table 3-2. The drivers were subjected to small (5°/s), medium (10°/s), and large (30°/s) ramp rates and the corresponding responses were recorded. A total of 10 responses for each ramp rate and DOC combination were recorded to determine the average driver response characteristics induced by each DOC.

Similarly to the step reference experiment, the results presented in this section are taken from a single driver that represents the average abilities of the drivers tested in terms of the quantitative characteristics discussed for each response shown. The choice of a representative driver for this discussion allows a more focused investigation on the abilities of the DriveGrip interface and not on the abilities of an interchangeable driver. In the interest of a more efficient analysis, a set of representative data from the recorded results will be shown in this section. The complete set of data recorded in these tests can be viewed in Appendix A.1.2 and will be compared with the representative data in this discussion.

Figure 3-18 displays the average (μ) driver response with standard deviation (σ) to a 10°/s ramp reference signal using each type of operation configuration listed in Table 3-2. Due to the manner in which the driver interacts with the DriveGrip interface, a final steering wheel angle rate is never achieved. For this reason, the qualitative characteristics shown in Table 3-4 compare the mean absolute error (MAE) of each response to the reference rather than the rise time, overshoot, settling time, and steady-state error.

The purpose of the ramp reference signal experiments conducted in this work are to determine if the functionality of the DriveGrip interface can additionally enable rate tracking despite that the interface was only designed for constant value tracking. Observation of Figure 3-18 demonstrates that rate tracking can indeed be
accomplished; however the tracking exhibits poor performance and, due to the functionality design of the interface, can lead to large oscillations centered on the reference ramp signal. This analysis will begin by examining the effects of the ramp signal on measuring the initial reaction time of the driver, followed by a discussion of the causes leading to poor rate tracking ability, and concluded with the comparison of each DriveGrip operation configuration’s rate tracking performance.

Figure 3-18: Average driver response to a $10^\circ$/s ramp reference signal utilizing various DriveGrip operation configurations defined in Table 3-2.

Table 3-4: Quantitative characteristics of the driver steering responses in Figure 3-18.

<table>
<thead>
<tr>
<th>Operation Configuration</th>
<th>Avg. Time Delay</th>
<th>Avg. Mean Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFST</td>
<td>310 ms</td>
<td>2.21°</td>
</tr>
<tr>
<td>FFLT</td>
<td>810 ms</td>
<td>4.58°</td>
</tr>
<tr>
<td>TFST</td>
<td>350 ms</td>
<td>1.56°</td>
</tr>
<tr>
<td>TFLT</td>
<td>850 ms</td>
<td>5.31°</td>
</tr>
</tbody>
</table>

The average time delays reported in Table 3-4 and depicted in Figure 3-18 indicate that the DOCs utilizing lenient error threshold tolerances cause longer initial delay times than those seen in the strict tolerances. This increase in initial time delay is the direct result of the interaction between the ramp rate and the index finger error threshold, $T_I$. Recall that $T_I$ is the minimal amount of instantaneous steering wheel angle error that must be reached before the DriveGrip interface transitions from the no-vibration state to
the index-finger vibration state. The increase in time delay comes from the time taken for the ramp signal to increase the instantaneous steering wheel reference angle from the starting position past the $T_i$ threshold. Once the $T_i$ threshold is passed, the DriveGrip interface makes the first communication to the driver that the steering wheel is not within an acceptable range of the reference angle. At this point, one can expect the remaining time delay before the initial response to be comparable with that of the step responses previously discussed. This effect can be clearly seen by comparing the initial time delays portrayed in Figure A-14 and Figure A-18 of Appendix A.1.2. The initial time delay and thus reaction time of each response is offset by the intersection of the $T_i$ error threshold line and the time axis.

The average driver responses to a $10^\circ$/s ramp signal in Figure 3-18 suggest that the DriveGrip interface does enable the driver to track a steering wheel rate to a certain degree. Further observation shows that, while the driver can track the rate, a steady state is never reached. This behavior arises because the driver is trained to associate the vibration locations with an angular position error and not with an angular velocity error. Although it may seem that the driver is directly tracking the rate in Figure 3-18, the driver is actually indirectly tracking the rate by tracking a continuously moving reference angle position. A certain degree of ramp tracking can be achieved with this indirect method; however the interface’s design to track a constant value can cause momentary drops in the angular rate of the driver’s response and even elicit a complete reversal of the angular rate direction.

The lapses in rate tracking ability stem from the interaction between the DriveGrip interface and the driver when reaching or overshooting the reference steering wheel angle. If the error threshold tolerances are large enough, then it is certainly possible for the driver to turn the wheel faster than the reference rate and reach the no-vibration zone located in the $\pm T_i$ instantaneous error range. Because the driver is trained to associate no vibrations with a correct steering wheel angle, the driver stops turning the wheel and holds the angular position constant. Depending on the rate of the ramp, the instantaneous reference angle soon travels outside of the $\pm T_i$ range while the driver holds the wheel angle constant. Once outside of this range, the fingers begin to vibrate once more and the driver continues to correct the steering wheel angle. This behavior is most clearly shown in Figure A-18 of Appendix A.1.2; here it is easy to observe the momentary drop-offs in the angular rate as the no-vibration zone is entered and exited.

If the no-vibration range around the reference value is overshot and the driver begins to lead the ramp signal, a complete reversal of the driver’s angular rate direction can occur. As described in the step analysis, overshooting the no-vibration range causes the fingers on the opposite hand to vibrate. Because the driver is trained to associate each hand with the direction of where the reference angle is located in relation to the current angle, the driver responds by turning the wheel in the opposite direction.
Although the driver is correctly steering towards the instantaneous reference angle at that point, he or she is turning in the opposite direction of the reference rate. The mistake is quickly realized by the driver when the reference value is overshot in the other direction; however, the initial reversal can create substantial deviations from the intended vehicle trajectory even in a short period of time. An example of this scenario is perfectly demonstrated in Figure A-15 of Appendix A.1.2.

Although all of the DriveGrip operation configurations of Table 3-2 are susceptible to the issues described in this discussion, some still outperform the others. Examination of Figure 3-18 shows that each DOC response relatively tracks the rate of the ramp reference; however, the DOCs with strict error threshold tolerances appear to do so with a smaller time lag. The time lags associated with the responses shown in Figure 3-18 are caused by the same reason for the initial time delays previously discussed in this subsection. Thus, the lower value of $\tau_i$ in the strict DOCs creates smaller time lags when tracking the reference ramp and yield lower mean-squared error as demonstrated in Table 3-4. Although the TFST and FFST DOCs elicit very similar responses from the driver, the two-finger configuration is chosen as the best DOC for tracking a ramp reference signal due to its slightly lower time lag. The average response for this DOC has been re-plotted with more detail in Figure 3-19 to highlight how its functionality elicits such a response from the driver.

![Figure 3-19: Average driver response to a 10°/s ramp reference signal utilizing the TFST DOC defined in Table 3-2.](image)

Figure 3-19: Average driver response to a 10°/s ramp reference signal utilizing the TFST DOC defined in Table 3-2.
3.1.4.3: **Arbitrary Reference Signal Tracking**

In this experiment, drivers were asked to track an arbitrary reference signal that was recorded from a sighted driver's steering actions while driving through an S-curved road course. Similarly to the step and ramp-type reference experiments, the drivers tracked the arbitrary signal 10 times with each of the four DriveGrip operation configurations defined previously in Table 3-2 to compare the average responses elicited from the drivers. The results presented in this section are taken from a single driver that represents the average abilities of the drivers tested in terms of the quantitative characteristics discussed for each response shown. The choice of a representative driver for this discussion allows a more focused investigation on the abilities of the DriveGrip interface and not on the abilities of an interchangeable driver.

Figure 3-20 and Figure 3-21 display the average ($\mu$) driver response with standard deviation ($\sigma$) to the pre-recorded arbitrary reference signal using each type of operation configuration listed in Table 3-2. The average driver responses shown in these figures clearly demonstrate that the driver can track the arbitrary reference signal, albeit with some amount of pure time delay. For this particular experiment, a time-shifted mean absolute error (MAE) calculation was performed to determine the pure time delay and MAE of each DOC response. This calculation works very similarly to cross-correlation: the DOC response is iteratively time-shifted along the reference signal and the MAE is measured at each time shift applied. The resulting calculations shown in Figure 3-22 and quantified in Table 3-5 indicate that the four-finger DOC responses yield lower average pure time delays and MAE than the responses elicited from the two-finger DOCs. The higher performance of the FFST and FFLT DOCs can be attributed to the higher resolution that the error is communicated through. When compared to the four-finger DOCs in Figure 3-20 and Figure 3-21, the lower-resolution two-finger configurations exhibit slower rise times that lead to higher time delays and MAE. The slower rise times of the two-finger configurations in this experiment are caused by instantaneous steering wheel angle errors that hardly surpass the $\pm T_L$ thresholds; almost the entire tracking process is done through the index fingers and thus at a relatively slow constant angular rate that spans a large range of angular error. Conversely, the four finger DOCs split this large range of angular error into smaller ranges that are communicated through the middle and ring fingers. The increased resolution provides the driver with more information and at a quicker rate so that faster rise times can be achieved with less risk of overshoot and oscillation.

A comparison of the MAE and pure time delay similarities between the FFST and FFLT DOC responses listed in Table 3-5 proves that these two performance metrics alone cannot indicate which of the DOCs induce the best tracking response from the driver. In order to determine the best operation configuration, a third performance metric based on controller (driver) effort will be compared. Examination of Figure 3-23 shows that the
driver response induced by the FFST DOC tends to oscillate when a steady value is reached and overshoot peaks and valleys of the reference signal. The oscillations and overshoot are caused by the strict $T_e$ error threshold tolerance, as has been discussed earlier in the step response analysis. Whenever oscillation or overshoot occurs, the driver must put more effort into correcting his or her response. Conversely, the FFLT DOC elicits a much smoother response due to the lenient $T_e$ threshold and requires a minimal amount of control effort from the driver.

Figure 3-20: Average driver response to an arbitrary reference signal utilizing the FFST and TFST DriveGrip operation configurations defined in Table 3-2.

Figure 3-21: Average driver response to an arbitrary reference signal utilizing the FFLT and TFLT DriveGrip operation configurations defined in Table 3-2.
Figure 3-22: Time-shifted Mean Absolute Error between an arbitrary steering reference signal and the driver’s response utilizing each of the DriveGrip operation configurations defined in Table 3-2.

Table 3-5: Quantitative characteristics of the driver steering responses plotted in Figure 3-20 and Figure 3-21.

<table>
<thead>
<tr>
<th>Operation Configuration</th>
<th>Avg. Initial Response Time</th>
<th>Avg. Time Lag</th>
<th>Avg. Mean Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFST</td>
<td>410 ms</td>
<td>400 ms</td>
<td>1.10°</td>
</tr>
<tr>
<td>FFLT</td>
<td>510 ms</td>
<td>460 ms</td>
<td>1.06°</td>
</tr>
<tr>
<td>TFST</td>
<td>380 ms</td>
<td>570 ms</td>
<td>1.45°</td>
</tr>
<tr>
<td>TFLT</td>
<td>500 ms</td>
<td>540 ms</td>
<td>1.37°</td>
</tr>
</tbody>
</table>

Figure 3-23: Comparison of the average driver response to an arbitrary reference signal utilizing the FFST and FFLT operation configurations defined in Table 3-2.
Although the FFLT DOC requires less driver effort, it still suffers from the steady-state error allowed by the lenient $\pm T_I$ range. Recalling from Figure 3-16, the lenient tolerances allow acceptable levels of lateral lane deviation at lower speeds; however significant deviations may begin to occur at speeds greater than 15m/s (~30mph). It is therefore suggested that a new hybrid DOC be implemented to require the minimum control effort from the driver while still maintaining the necessary degree of accuracy to safely navigate the vehicle within a driving lane. The Hybrid DOC would be composed of steering angle error thresholds that are dependent upon the speed of the vehicle. Such a configuration would allow more lenient tolerances at lower speeds and increase the strictness with higher speeds. An example of a possible Hybrid DOC is shown in Figure 3-24 and defined by the equations:

\[
T_I = 8e^{-0.1S_A} \quad \quad (3.5) \\
T_M = 16e^{-0.1S_A} \quad \quad (3.6) \\
T_R = 32e^{-0.1S_A} \quad \quad (3.7) \\
T_L = 64e^{-0.1S_A} \quad \quad (3.8)
\]

where $S_A$ is once again the instantaneous forward speed of the vehicle. The use of exponential functions in the error threshold definitions creates lenient error thresholds at lower speeds that exponentially decrease as the speed increases. While these equations serve as an example, further research must be conducted to determine what relationship between the error thresholds and speed induces the best response from the driver.

![Figure 3-24: Example of speed-dependent steering wheel angle error thresholds for a new Hybrid DriveGrip operation configuration.](image-url)
3.1.4.4: Desensitization Rejection

In this experiment, drivers were subjected to substantial periods of continuous DriveGrip interaction to examine the influence of desensitization on the interface’s ability to communicate the instantaneous steering wheel angle error to the driver. The tests were conducted while practicing at the Virginia International Raceway (VIR) for an upcoming vehicle demonstration. Before practice began, the drivers were asked to track an arbitrary steering wheel angle reference signal using the FFST DriveGrip operation configuration previously defined in Table 3-2. Afterwards, the drivers performed approximately seven hours of practice driving within an 8 hour period using the DriveGrip interface. After this extended period of interface usage, the arbitrary signal tracking experiment was conducted once more with the same reference signal and FFST DOC utilized previous to the practice period. The responses from each driver before and after the practice period were recorded for comparison to analyze the effects of desensitization over the seven hour usage period. The experiment was conducted with two drivers over the course of three days; thus the pre-practice and post-practice responses of each driver were recorded three times.

Figure 3-25 and Figure 3-26 plot the average (μ) and standard deviation (σ) of the three responses for each test and driver of the experiment. The results indicate that the average driver responses remain relatively the same even after a seven hour DriveGrip usage period. The differences in the responses for each driver at the first valley and last peak are not the result of desensitization or fatigue but of slightly different interactions between the driver, the steering wheel, and the DriveGrip interface. Close examination of these differences demonstrates that the combination of minor fluctuations in the angular velocity of the driver’s response and the shape of the reference signal can lead to oscillations in some cases but not in others. This is made most evident in the first valley of each driver’s pre and post-practice response. The difference in responses at the final peak of the reference signal can be attributed to an increase in the driver’s overall tracking performance as a result of the seven-hour practice period. The drivers each overshoot the final peak at the same point; however the post-practice responses show a substantial decrease in oscillations during the driver’s attempt to correct the overshoot. With these slight differences accounted for, the similarities of the drivers’ pre and post-practice responses shown in Figure 3-25 and Figure 3-26 clearly indicate that the drivers can reliably track a reference signal even after seven hours of continuous interface usage. Since the DriveGrip interface is the only source of steering input to the driver, this conclusion can be fairly extended to state that the drivers’ interaction with the DriveGrip interface does not become hindered by desensitization in periods of prolonged utilization.
While this analysis indicates that the overall tracking ability is not hindered by desensitization, it is suggested that future research be conducted to examine the effects of desensitization at the interface level rather than at the response level. Experiments may be conducted to activate each vibro-tactile element of the interface at random and instruct the driver to make a pre-planned action in response to the stimulus created by each element. Such an experiment would directly examine the effects of desensitization at its fundamental source by analyzing how the driver’s tactile perception of the stimuli changes over periods of prolonged usage.
3.2: SpeedStrip – Velocity Assistance Interface

SpeedStrip is a Non-Visual Interface used to provide vehicle speed information to the driver through vibrational haptics [5]. As described in Section 1.2.1.2, SpeedStrip incorporates the use of vibrational motors positioned beneath the thighs and on the back of the driver. The vibration sensations created from this interface are able to cue the driver with primary and/or supplemental velocity information based on calculated trajectories from local motion planning. A prototype of this interface was proposed and tested in [5], however this work presents a significant improvement of the SpeedStrip system; transitioning it from a simple prototype to an advanced, dependable, and complete implementation.

3.2.1: Functionality

As described in [5], the function of the SpeedStrip interface is to communicate the instantaneous speed error to the driver through vibratory haptics imposed on the legs and back of the driver. The instantaneous speed error communicated through this interface is given by the simple equation:

\[ S_E = S_R - S_A \]  \hspace{1cm} (3.9)

where \( S_E \) is the instantaneous speed error; \( S_R \) is the instantaneous reference speed specified by the transformation algorithms presented in Section 3 and 0; and \( S_A \) is the instantaneous actual speed of the vehicle measured by the Velocity State Sensor (Sections 2.1.2.1 and 2.3.2.1.4). It should be noted that, in this work, all vehicle speeds will be communicated in meters per second (m/s).

In order to communicate the instantaneous speed error to the driver, SpeedStrip associates leg and back positions with a range of error magnitude and direction. With regard to magnitude, vibrations closest to the buttocks of the driver indicate the lowest range of error magnitude while vibrations furthest from the buttocks indicate the highest range of error magnitude. With regard to direction, vibrations on the legs indicate positive error while vibrations on the back indicate negative error. With this configuration, the driver is instructed to accelerate when vibrations are felt in the legs and decelerate and/or stop when vibrations are felt in the back. The further the position of the vibration is from the buttocks indicates how much more acceleration or deceleration is required. Similarly to DriveGrip, the driver uses SpeedStrip to act as a simple Proportional controller. An additional case exists wherein all vibro-tactile elements activate simultaneously to instruct the driver to bring the vehicle to a complete stop. Drivers are trained to react to this full-stop request with the highest priority to avoid imminent collisions and maximize the safety of the system. The following table dictates how each particular vibrato-tactile element is controlled in both a four element and two element vibration scheme:
Table 3-6: SpeedStrip Vibro-Tactile Element Control Table for four element and two element vibration configurations.

<table>
<thead>
<tr>
<th>Body Location</th>
<th>Relative Location</th>
<th>Activated Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back</td>
<td>Shoulder</td>
<td>$-\infty &lt; S_E \leq -T_{B4} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td></td>
<td>Upper</td>
<td>$-T_{B4} &lt; S_E \leq -T_{B3} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td></td>
<td>Middle</td>
<td>$-T_{B2} &lt; S_E \leq -T_{B2} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td></td>
<td>Lower</td>
<td>$-T_{B2} &lt; S_E \leq -T_{B1} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td>Legs</td>
<td>Buttocks</td>
<td>$T_{L1} \leq S_E &lt; T_{L2} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td></td>
<td>Upper Thigh</td>
<td>$T_{L2} \leq S_E &lt; T_{L3} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td></td>
<td>Lower Thigh</td>
<td>$T_{L3} \leq S_E &lt; T_{L4} \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
<tr>
<td></td>
<td>Knees</td>
<td>$T_{L4} \leq S_E &lt; +\infty \text{ OR } S_R &lt; T_{STOP}$</td>
</tr>
</tbody>
</table>

where $T_{B1}$, $T_{B2}$, $T_{B3}$, and $T_{B4}$ are the lower, middle, upper, and shoulder thresholds of the back, respectively; $T_{L1}$, $T_{L2}$, $T_{L3}$, and $T_{L4}$ are the buttocks, upper thigh, lower thigh, and knee thresholds of the legs, respectively; and $T_{STOP}$ is the instantaneous reference speed threshold at which a full stop request is communicated.

This particular configuration of the vibro-tactile elements enables the driver to dependably perceive the tactile cues from SpeedStrip concurrently with the tactile cues from the DriveGrip interface (Section 3.1). As the legs and back are not principle tactile sense areas [115], the driver can primarily focus on the DriveGrip interface to navigate the road and secondarily focus on the SpeedStrip cues for maintaining safe speeds. A second advantage of this configuration is that it can communicate information to the driver with minimal influence from desensitization. The motors, similarly to the DriveGrip interface, operate in a simple binary scenario unless in the event of a full stop request. In situations other than full stops, only one motor in the entire SpeedStrip interface vibrates at any given time; making it easy for the driver to discern the vibration position and thus the error magnitude and direction even in periods of prolonged usage.

3.2.2: Interface Development

As the original vibro-tactile chair and associated control hardware presented in [5] were prototypes, this paper proposes a complete redesign of the system to drastically improve performance, efficiency, and robustness. The redesigned system implements the same basic functionality as that of [5] on a completely new set of hardware that can fully unlock the potential of the interface. The new system was fabricated, assembled, and rigorously tested to meet the expected level of performance required for our driver assistance research application.
3.2.2.1: **Layout**

Figure 3-27 displays the layout of the SpeedStrip non-visual interface. The layout includes 16 total vibration components, wiring, and a standard connector all mounted within a padded seat cushion designed to fit over the driver seat of the vehicle. The vibration components are grouped into 8 rows of 2 components; 4 of which are positioned below the legs and the other 4 of which are positioned on the back. The 8 rows of vibration components are operated one at a time to decrease the conscious effort required by the driver to understand information from the vibro-tactile cues in real-time. This makes the system significantly less susceptible to desensitization over prolonged periods of use. Similarly to the DriveGrip interface, the resistance to desensitization also presents a tradeoff in the form of low information resolution as the configuration only supports a total of 9 unique cues. However, the SpeedStrip interface must also operate in real-time and in prolonged periods of use; thus high resolution communication was again sacrificed to increase human reliability.

![SpeedStrip System Layout](image)

**Figure 3-27**: SpeedStrip System Layout. Components include: Vibratory Motors (red), Standardized Connector (blue), Wiring Harness (Green), and Mounting Seat Cushion (gray).

The SpeedStrip interface layout also incorporates a connector (Figure 3-27) for attaching signal wires between the seat cushion and the BDC platform. The connector is designed to allow the interface to be easily connected and disconnected from the platform for debugging and/or testing. The connector also standardizes how the system attaches to the platform so that different sets or versions of the SpeedStrip interface may be interchanged between platforms.
The wiring harness used to connect the vibratory motors to the standardized connector is laid out to minimize the effects of wire fatigue due to body movement and pressure on the interface. The wires are bundled together and connected to components at strategic points, creating large bending radii to minimize stress applied to the wires under high-cycle bending and pressure changes.

The SpeedStrip interface components are collectively attached to the padded seat cushion shown in Figure 3-27. The driver is seated over the entire cushion, which comfortably enforces positioning and contact of the vibratory motors on the legs and back of the driver. It also aids in the positioning and protection of the standardized connector and wiring harness to minimize fatigue on the wire and associated connections.

3.2.2.2: Component Specification

The vibration components and seat cushion used in the SpeedStrip interface are parts taken from a HoMedics® Back Revitalizer™ 8-Motor Back Massager with Heat. The vibration component, shown in Figure 3-28, is simply a 12VDC, 840mW DC motor with an offset weight packaged in a plastic shell. This type of “massaging” vibration motor is easily repurposed for the SpeedStrip application as it is already designed for interacting with the human body. The only alteration made to the vibration component was wiring the internal DC motor to the interface’s standard connector.

![Figure 3-28: HoMedics® Vibration Motor used in the SpeedStrip Interface](image)

The mounting seat cushion is created from the covers of two top-halves of the HoMedics® Back Revitalizer™ massage cushions and standard 1in thick foam. The covers from the massage cushions have built in padding and add to the aesthetics of the interface. The 1in foam is used to provide additional padding and also to serve as a Velcro® mounting base for the vibration components within the seat cushion. The seat cushion also includes a zipper around the sides of the interface for easy access to system internals.
SpeedStrip uses a female DB9 port as its standard connector, allowing direct connection between the interface and the Non-Visual Interface Controller (Section 3.4). The wiring harness that links the DB9 connector to the vibration components is created from standard 22AWG single conductor stranded wire. The medium gauge stranded wire provides enough strength and flexibility to withstand the high-cycle bending and pressure changes present in the usage of a seat cushion.

### 3.2.2.3: Electrical Schematic

Figure 3-29 displays the electrical schematic for the connections between the vibration components and the DB9 standard connector. All vibration components share a common anode, while each row of vibration components shares a common cathode. The 1 common anode and 8 separate cathodes are wired directly to the DB9 standard connector. In this application the common anode provides a positive voltage to all vibration components at all times, and is connected to pin 1 on the DB9 connector. The common cathode of each row acts as the control channel for the associated vibration components, and thus is connected directly to the Non-Visual Interface Controller (Section 3.4) through pins 2-9 of the DB9 standard connector.

**Figure 3-29: Electrical Schematic of the SpeedStrip Non-Visual Interface**

### 3.2.2.4: Assembly Design

The assembly design of the SpeedStrip interface is very important as it helps decrease wire fatigue and additionally helps protect components in the seat cushion. The assembly starts with creating the wire harness that links the vibration components to the DB9 standard connector. Figure 3-30 shows how the wiring harness is attached to each vibration component. With this particular wiring configuration, all solder joints
connecting the wiring harness to the internal DC motor are housed inside the plastic shell for protection. The wires protruding from the shell internals are then wrapped around to the other side of the shell with a large bend radius and hot glued into place. This creates stress relief on the solder joints located within the vibration component.

![Diagram of vibration component and hot glue](image1)

**Figure 3-30:** Wiring harness stress relief for the SpeedStrip interface vibration components (top view).

Another important feature of the assembly design is the use of Velcro® for positioning the vibration components on the mounting foam within the seat cushion. Figure 3-31 shows how Velcro® is used for highly reconfigurable positioning of the vibration components on the legs and back of the driver. Adhesive strips of Velcro® are placed on the 1in thick mounting foam and also on the vibration components themselves. In this manner, the vertical spacing of each vibration row can be easily adjusted. Horizontal spacing can also be adjusted by adding more adhesive Velcro® strips to the mounting foam. In this application, two layers of mounting foam surround the vibration components. This configuration ensures that the components and wiring are protected and held in position during prolonged periods of use. The double foam padding also increases the comfort of the interface for the driver.

![Diagram of reconfigurable positioning](image2)

**Figure 3-31:** Reconfigurable positioning of SpeedStrip vibration components with Velcro® strips.
The central wiring harness is attached to the DB9 standard connector via solder cups. Heat-shrink was placed over each solder connection to increase strength and prevent breaks from fatigue. The connection is also further stabilized by a tightly wound shroud of electrical tape between the DB9 body and the first 2in of wire protruding from the connector. The DB9 body is also sewn into place on the seat cushion at where the upper and lower halves meet. This positions the DB9 connection to the interface controller in the crease of the driver seat for protection and driver comfort.

3.2.2.5: Completed Redesign

Figure 3-32 displays the completed SpeedStrip non-visual interface positioned in the Blind Driver Challenge® platform. The combination of the carefully designed system layout and the assembly procedures ensure that this interface will dependably provide vehicle velocity information to the driver even in periods of prolonged usage.

Figure 3-32: Completed SpeedStrip Non-Visual Interface mounted inside of the Blind Driver Challenge® vehicle.

3.2.3: Platform Implementation

The SpeedStrip non-visual interface is integrated into the vehicle using a single seat mount and connection to the Non-Visual Interface Controller (Section 3.4). The SpeedStrip mounting scheme and corresponding wiring layout to the controller are described in this subsection.
3.2.3.1: **SpeedStrip Seat Mount**
SpeedStrip is integrated into the platform via placement on the driver seat (as shown in Figure 3-33) and a positioning mount on the headrest. The positioning mount is actually a part of the DriveGrip interface mount described in Section 3.1.3.2, simplifying the modifications made to the vehicle by combining both non-visual interface mounts into a single part. The SpeedStrip mount consists of two slots in the aluminum frame of the DriveGrip mount, in which elastic Velcro® straps loop through to hold the top of the SpeedStrip interface in position to prevent it from sliding and/or folding downwards. This configuration can be visualized in Figure 3-33.

![Figure 3-33: SpeedStrip and DriveGrip interface headrest mount.](image)

3.2.3.2: **Wiring Layout**
The connection between the SpeedStrip interface and the Non-Visual Interface Controller (Section 3.4) is a simple 2ft, 9-conductor ribbon cable spanned between the two components through the crease between the seat and back of the driver’s seat. As the interface’s DB9 standard connector is already positioned in the crease of the seat, the link cable connects to the interface and immediately crosses underneath the seat to hide and protect the cable. The link cable uses a small and unobtrusive IDC male DB9 connector that fits within the seat crease to maintain the seating comfort for the driver.
3.2.4: Performance Analysis

As described in earlier in this section, the SpeedStrip interface is responsible for communicating the instantaneous speed error to the driver via vibro-tactile components placed on the thighs and back of the driver. The driver subsequently uses this information to control the forward velocity of the vehicle while navigating through an environment. This subsection documents a series of experiments conducted to test the ability of drivers to track reference speeds using only the information communicated by the SpeedStrip interface and without performance degradation due to desensitization.

NOTE: The results presented in this work were obtained from test subjects in collaboration with the National Federation of the Blind and cooperate with the Virginia Tech Institutional Review Board (IRB) Human Subject Research Protocol.

3.2.4.1: Step Reference Signal Tracking

In this experiment, drivers were asked to track a range of various step reference speed signals to analyze and compare the transient response properties elicited by each of the two SpeedStrip operation configurations (SOCs) defined in Table 3-7. The drivers were subjected to small (5m/s) and medium (10m/s) amplitude step references and the corresponding responses were recorded. Unfortunately, the speed limiter installed on the TORC ByWire XGV™ (Section 2.1) prevented the ability to experiment with speeds greater than 10m/s. A total of 5 responses for each step amplitude and SOC combination were recorded to determine the driver response characteristics induced by each SOC.

Table 3-7: SpeedStrip Operation Configurations (SOCs)

<table>
<thead>
<tr>
<th>Operation Configuration</th>
<th>Vibration Configuration</th>
<th>T_{B_4}</th>
<th>T_{B_3}</th>
<th>T_{B_2}</th>
<th>T_{B_1}</th>
<th>T_{L_1}</th>
<th>T_{L_2}</th>
<th>T_{L_3}</th>
<th>T_{L_4}</th>
<th>T_{STOP}</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>Four Element</td>
<td>-2.5</td>
<td>-2</td>
<td>-1.5</td>
<td>-1</td>
<td>1</td>
<td>1.5</td>
<td>2</td>
<td>2.5</td>
<td>0.01</td>
</tr>
<tr>
<td>TE</td>
<td>Two Element</td>
<td>-2.5</td>
<td>N/A</td>
<td>-1</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
<td>2.5</td>
<td>0.01</td>
</tr>
</tbody>
</table>

While the experiments conducted in this analysis were performed with several drivers, the results presented in this section are taken from a single driver that represents the average abilities of the drivers tested in terms of the transient properties discussed for each response shown. The choice of a representative driver for this discussion allows a more focused investigation on the abilities of the SpeedStrip interface and not on the abilities of an interchangeable driver. In the interest of a more efficient analysis, a set of representative data from the recorded results will be shown in this section. The complete set of data recorded in these tests can be viewed in Appendix A.2.1 and will be compared with the representative data in this discussion.
Before the analysis begins, it is important to recall from Section 3.2.1 that the SpeedStrip interface is designed to require less focus from the driver than the DriveGrip interface. This design goal was important to ensure that the driver primarily focuses on steering the vehicle and secondarily focuses on the speed. The relatively large speed error thresholds defined in Table 3-7 require less speed correction from the driver and thus demand less of the driver’s focus. Additionally, the vibration elements were specifically placed on parts of the body that are not primary tactile sensing areas [115] and the drivers are trained to understand that the correction of the speed is not critical unless a full stop request is communicated. Therefore, the responses elicited by the SpeedStrip interface are expected to contain slower reaction times, longer settling times, and larger ranges of steady-state error than the steering responses unless a full stop is requested.

The driver responses to a 5m/s step speed reference signal using the two and four-element SOCs are shown in Figure 3-34 and Figure 3-35, respectively. A quick observation of these figures shows that the speed responses are discontinuous and are composed of ramps with varying rates and directions. This behavior emanates from the fact that the drivers must indirectly control speed by controlling acceleration and deceleration with the throttle and brake pedals. Since the driver only has the ability to control the rate of acceleration and deceleration, the speed response is completely composed of the acceleration and deceleration actions and thus the ramps of instantaneous speed.

![Figure 3-34: Driver responses to a 5m/s step speed reference signal using the FE SOC defined in Table 3-7.](image-url)
Figure 3-35: Driver responses to a 5m/s step speed reference signal using the TE SOC defined in Table 3-7.

Close observation of the no-vibration range indicates that the driver is able to make fine speed corrections without any communication from the SpeedStrip interface. This interesting behavior occurs because the driver has additional speed feedback from his or her perception of the vehicle’s inertia. After passing into the no-vibration range, the driver begins to make slight throttle adjustments in an attempt to reach a constant speed. Once a constant speed is achieved, all throttle adjustments cease.

The average transient properties of the FE and TE responses shown in Figure 3-34 and Figure 3-35 were calculated and are listed in Table 3-8. Inspection of the average time delays for each SOC indicates that the driver has an average response time of approximately 580ms. As expected, this response time is considerably slower compared to the average DriveGrip response time of ~357ms (Section 3.1.4.1). The positioning of the vibro-tactile elements on non-primary tactile sensation areas of the human body exhibit a slightly longer time delay between perception and reaction to the vibration stimuli created by the SpeedStrip interface.

Table 3-8: Average transient characteristics for the FE and TE SOC induced responses shown in Figure 3-34 and Figure 3-35.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>570 ms</td>
<td>0.946 m/s</td>
<td>6846 ms</td>
<td>0.52 m/s</td>
</tr>
<tr>
<td>TE</td>
<td>590 ms</td>
<td>0.222 m/s</td>
<td>8586 ms</td>
<td>0.28 m/s</td>
</tr>
</tbody>
</table>

Further examination of the driver’s initial responses to the step reference signal shows repeated patterns of a quick acceleration followed by a brief period of constant speed before the full acceleration of the response begins. This behavior is caused by the delay in time for the driver to remove his or her foot from the brake pedal and shift it over to the accelerator pedal. As soon as the foot is removed from the brake pedal, the automatic transmission and torque converter of the TORC ByWire XGV™ provide a
slight forward force on the vehicle even though the accelerator pedal is not depressed. This causes a small acceleration in speed that, on flat ground, quickly brings the vehicle to a constant speed of 0.2m/s. Once the driver moves his or her foot over to the accelerator and begins to depress the pedal, a much larger acceleration in speed is created and the driver begins to track the reference signal.

The average overshoot values listed in Table 3-8 indicate that the FE SOC typically induces a much higher overshoot than the TE SOC. The higher overshoot can be attributed to the faster accelerations used by the driver when interacting with the FE SOC. An examination of the responses in Figure 3-34 and Figure 3-35 shows that the FE induced response utilizes an average initial acceleration that is approximately 15% faster than that of the TE induced response. The larger accelerations lead to larger changes in the speed during the time it takes for the driver to react to the no-vibration stimulus, thus increasing the compared overshoot. The increase in initial acceleration seen in the FE induced response is caused by the driver’s expected performance increase when given higher resolution error communication. However, the increase in resolution creates error/vibration ranges that are too small for the driver to react to in a timely manner when using larger accelerations, and thus the reference value is quickly overshot. Conversely, the slower accelerations induced by the lower resolution TE response create smaller overshoot even though the same inner and outer error threshold values are used (Table 3-7).

Table 3-8 suggests that the FE SOC typically elicits responses with lower settling times. However, the settling time of the driver’s response utilizing either SOC is highly stochastic and largely independent from the SOC. As discussed earlier, the driver is able to continuously make fine adjustments in speed within the acceptable error and no-vibration range in an effort to maintain constant speed. These fine adjustments are random and entirely at the discretion of the driver; thus the settling times cannot be attributed to the type of SpeedStrip operation configuration in use.

The average steady state error (SSE) for each SOC induced response is quantified in Table 3-8. After the driver initially overshoots the reference value, the error thresholds of each SOC force the driver to continuously correct towards the reference value until the error is between TB1 and TL1 for the FE SOC or between TB2 and TL2 for the TE SOC. Thus, the error thresholds of each SOC directly define the range of the steady-state error for each response. Table 3-7 indicates that the lowest error threshold is the same for both SOCs, thus the SSE is expected to be within 1m/s of the reference value for both the FE and TE induced responses.

Comparison of the transient response characteristics induced by each SOC suggests that the TE SOC shows a marginal advantage when tracking step inputs. The only difference in performance discovered between utilization of the TE and FE SOCs is the
smaller overshoot typically elicited by the TE SOC. An additional advantage of the TE SOC that has not yet been discussed is that it requires less cognitive effort from the driver due to the decrease in stimulus resolution. The driver must only interact with a total of four vibro-tactile elements using the TE SOC as opposed to eight vibro-tactile elements with the FE SOC. The decrease in resolution provides less information to the driver which requires less processing and thus a decreased cognitive load. The combination of marginally higher transient performance with lower required cognitive effort demonstrates that the TE SOC is more appropriate for tracking step reference speeds with the SpeedStrip interface.

3.2.4.2: Ramp Reference Signal Tracking
In this experiment, drivers were asked to track a range of various ramp reference speed signals to analyze and compare the response properties elicited by each of the two SpeedStrip operation configurations (SOCs) defined previously in Table 3-7. The drivers were subjected to small (0.3m/s²) and medium (0.5m/s²) ramp rates and the corresponding responses were recorded. A total of 5 responses for each ramp rate and SOC combination were recorded to determine the quantitative driver response characteristics induced by each SOC.

Similarly to the step reference experiment, the results presented in this section are taken from a single driver that represents the average abilities of the drivers tested in terms of the quantitative characteristics discussed for each response shown. The choice of a representative driver for this discussion allows a more focused investigation on the abilities of the SpeedStrip interface and not on the abilities of an interchangeable driver. In the interest of a more efficient analysis, a set of representative data from the recorded results will be shown in this section. The complete set of data recorded in these tests can be viewed in Appendix A.2.2 and will be compared with the representative data in this discussion.

Figure 3-36 and Figure 3-37 display the driver responses to a 0.3m/s² ramp speed reference signal using each type of operation configuration listed in Table 3-7. Due to the manner in which the driver interacts with the SpeedStrip interface, a final acceleration value is never achieved. For this reason, the qualitative characteristics shown in Table 3-9 compare the mean absolute error (MAE) of each response to the reference rather than the rise time, overshoot, settling time, and steady-state error.

The purpose of the ramp reference signal experiments conducted in this work are to determine if the functionality of the SpeedStrip interface can additionally enable rate tracking despite that the interface was only designed for constant speed tracking. Observation of Figure 3-36 and Figure 3-37 demonstrates that rate tracking can be accomplished; however the tracking exhibits poor performance and, due to the functionality design of the interface, can lead to large oscillations centered just inside of
the smallest leg error threshold. This analysis will begin by examining the effects of the ramp signal on measuring the initial reaction time of the driver, followed by a discussion of the causes leading to poor rate tracking ability, and concluded with the comparison of each SpeedStrip operation configuration’s rate tracking performance.

Figure 3-36: Driver responses to a 0.3m/s² ramp speed reference signal using the FE SOC defined in Table 3-7.

Figure 3-37: Driver responses to a 0.3m/s² ramp speed reference signal using the TE SOC defined in Table 3-7.
Table 3-9: Average quantitative characteristics of the driver responses shown in Figure 3-36 and Figure 3-37.

<table>
<thead>
<tr>
<th>Operation Configuration</th>
<th>Avg. Time Delay</th>
<th>Avg. Mean Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>4008 ms</td>
<td>0.61 m/s</td>
</tr>
<tr>
<td>TE</td>
<td>4056 ms</td>
<td>0.59 m/s</td>
</tr>
</tbody>
</table>

The average time delays to a ramp reference signal listed in Table 3-9 are exceedingly larger than that of the step reference responses shown in the previous subsection. The increase in average time delay is actually attributed to the time delay between the start of the ramp signal and the initial traversal of the instantaneous speed error past the innermost error threshold. The FE and TE SOCs used in this experiment both utilize an innermost error threshold value of 1 m/s (Table 3-7). The 0.3 m/s² rate of the ramp reference signal causes the error to traverse this innermost error threshold approximately 3333 ms after the reference starts, thus contributing an extra 3333 ms of time to the average time delay of each driver response. Taking this increase into account, the average reaction time of the driver is approximately 699 ms after the initial stimulus created by the SpeedStrip interface.

The 0.3 m/s² and 0.5 m/s² ramp reference responses shown in this subsection and Appendix A.2.2 indicate that the driver is able to generally follow the ramp in speed with limited performance. As previously described, this occurs because the SpeedStrip interface is designed to communicate instantaneous speed error only and not acceleration error. The driver must then indirectly track the ramp in speed as a series of step responses. This behavior can clearly be seen in both the FE and TE SOC induced responses. The driver, having positive speed error, accelerates until the instantaneous speed error falls within the acceptable error (no-vibration) range. At this point, the driver then begins to use his or her perception of the vehicle’s inertia to make fine speed corrections and achieve a constant speed. However, the ramping of the speed quickly forces the error outside of the acceptable range in the positive direction. The driver responds by once again increasing speed until the error falls within the acceptable error range, and the process repeats for the remainder of the ramp reference. The rate that this process repeats is dependent upon the ramp reference: higher ramp reference rates lead to faster repetitions of the process. This behavior can be observed by comparing the 0.3 m/s² and 0.5 m/s² ramp reference responses in Appendix A.2.2.

At the acceleration values tested, the FE and TE SOC induced responses exhibit relatively similar behavior. This is expected as the ramping generally prevents the instantaneous speed error from traversing higher error thresholds, eliminating the functional difference between the FE and TE SOCs. Table 3-9 confirms this statement with similar average MAEs between the two operation configurations. Since the tracking performance does not indicate a clear advantage in either of the SOCs, the
best SOC will be chosen based on the cognitive load imposed. Recalling from the step analysis, the TE SOC requires the least amount of cognitive effort by the driver due to the lower resolution and thus lower amount of information that is constantly communicated to the driver. Thus, it is suggested that the TE operation configuration be used when attempting to track a ramp reference speed signal.

3.2.4.3: Arbitrary Reference Signal Tracking

In this experiment, drivers were asked to track an arbitrary reference signal that was recorded from a sighted driver’s brake and throttle actions while driving through an elongated S-curved road course. The drivers tracked the arbitrary signal with each of the two SpeedStrip operation configurations defined previously in Table 3-7 to quantitatively compare the responses elicited from the drivers. The results presented in this section are taken from a single driver that represents the average abilities of the drivers tested in terms of the quantitative characteristics discussed for each response shown. The choice of a representative driver for this discussion allows a more focused investigation on the abilities of the SpeedStrip interface and not on the abilities of an interchangeable driver.

Figure 3-38 displays the driver response to the pre-recorded arbitrary reference signal using the FE and TE operation configurations listed in Table 3-7. The driver responses shown in this figure clearly demonstrates that the driver can track the arbitrary reference signal, albeit with some amount of pure time delay. For this particular experiment, a time-shifted mean absolute error (MAE) calculation was performed to determine the pure time delay and MAE of each DOC response. This calculation works very similarly to cross-correlation: the DOC response is iteratively time-shifted along the reference signal and the MAE is measured at each time shift applied. The resulting calculations shown in Figure 3-39 and quantified in Table 3-10 indicate that the responses induced by each SOC exhibit relatively similar pure time delays and MAEs. The 260ms difference between the pure time delays of each response can be accounted for by the larger accelerations present in the FE induced response that were previously described in the step response analysis. The larger accelerations allow faster response times in significant reference speed changes, thus the FE SOC induces a lower pure time delay in the driver’s response. This behavior can easily be seen in the FE response shown in Figure 3-38.

While the mean absolute error analysis suggests that neither of the SpeedStrip operation configurations exhibits a superior tracking ability, the FE SOC provides the advantage of lower pure time lags due to the elicitation of faster accelerations. Lower time lags in the driver’s response requires less compensation by the reference signal planner and enables the driver to travel at faster speeds while still maintaining a safe level of control. Conversely, the TE SOC still provides the advantage of lower cognitive load on the driver as the decreased resolution requires less interaction. Thus, there
exists a tradeoff between the time-lag advantage of the FE induced response versus the reduced cognitive load induced by the TE response. Since the speed control through SpeedStrip is designed to be lenient, the rate at which the speed is increased is relatively unimportant. However, the rate at which the vehicle is slowed or stopped is extremely important to the safety of the driver and the surrounding environment. Figure 3-38 indicates that the reduced time lag of the FE response is only present at increases in the vehicle speed, while the decreases in speed exhibit the same time lags as the TE response. It is more important for the SpeedStrip interface to require less cognitive effort from the driver and allow more focus on the DriveGrip interface than it is to gain faster increases in vehicle speed. Thus, the TE operation configuration is suggested for tracking arbitrary speed reference signals.

Figure 3-38: Driver responses to an arbitrary reference signal using the FE and TE SOCs defined in Table 3-7.

Figure 3-39: Time-shifted Mean Absolute Error between an arbitrary speed reference signal and the driver’s response utilizing each of the SpeedStrip operation configurations defined in Table 3-7.
Table 3-10: Quantitative characteristics of the driver speed responses plotted in Figure 3-38.

<table>
<thead>
<tr>
<th>Operation Configuration</th>
<th>Avg. Initial Response Time</th>
<th>Avg. Time Lag</th>
<th>Avg. Mean Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>3250 ms</td>
<td>1740 ms</td>
<td>0.202 m/s</td>
</tr>
<tr>
<td>TE</td>
<td>3280 ms</td>
<td>2000 ms</td>
<td>0.188 m/s</td>
</tr>
</tbody>
</table>

3.2.4.4: Deceleration and Breaking Differentiation

An additional experiment was conducted to further examine the deceleration and braking behavior of the driver while tracking a decreasing instantaneous reference speed signal. In this experiment, drivers were subjected to successive downward ramps in reference speed and their deceleration and braking behavior was recorded. A full stop request, in which the reference speed falls to 0 m/s, is also included in the test to examine the effectiveness of the interface’s STOP vibration pattern. The results presented in this section are taken from a single driver that represents the average behavior of the drivers tested in terms of the qualitative characteristics discussed for each response shown.

The recorded braking and deceleration behaviors of the representative driver are shown below in Figure 3-40 and Figure 3-41. Each plot indicates the braking and deceleration periods within reach response as well as displays the error thresholds for each SpeedStrip operation configurations defined in Table 3-7. Figure 3-40 clearly demonstrates that, while utilizing the FE SOC, the driver will decelerate while the instantaneous speed error is within the lowest two error ranges and switch to braking once the instantaneous error surpasses the third error threshold, $T_{B3}$. The driver associates the lower and middle back vibration locations with smaller speed error and thus decelerates as a relatively small decrease in speed is required. Conversely, the driver associates the upper back and shoulder vibration locations with large error and switches to the brake to apply a larger decrease in speed in a shorter amount of time. The TE SOC induces a slightly different behavior from the driver due to the decreased resolution of information provided. Examination of Figure 3-41 shows that the driver utilizes a mixture of deceleration and brake “tapping” if the instantaneous speed error is within the lowest error range. The driver initially responds with a single tap on the brake and then switches to deceleration. If the speed error remains within the lowest error range for too long, the driver applies a second tap and repeats the process until the error falls within the acceptable range and the no-vibration stimulus is enabled.

It is interesting that, once the driver begins to slow down, they continue to decelerate even though the error falls within the no-vibration range. This possibly occurs because the drivers are trained to minimize the risks of safety and focus more on slowing/stopping rather than increasing speed. Thus, the behavior shown may be
caused by the driver’s assumption that the reference speed will continue to decrease even though the instantaneous speed error is currently within the acceptable range. This helps increase safety because, once the initial decrease in speed is communicated, the driver has the option to preemptively continue decelerating until the error falls outside of the acceptable range and new information is provided. The preemptive decelerations assist in driving the vehicle at the near-minimum of the acceptable speed range and keep the driver in better preparation for continued decreases in speed.

Figure 3-40: Deceleration vs. Braking using the FE SOC defined in Table 3-7.

Figure 3-41: Deceleration vs. Braking using the TE SOC defined in Table 3-7.
Once the reference speed signal fell below the $T_{STOP}$ error threshold defined in Table 3-7, both FE and TE operation configurations issued a full stop request to the driver by vibrating all four rows of vibro-tactile elements on the driver’s back. The point at which this occurs is depicted by the dashed vertical line in Figure 3-40 and Figure 3-41. Observation of the driver’s responses in each of the two figures indicates that the driver understands the full stop request and, after a slight pause accounting for reaction time, brings the vehicle to a controlled stop in as short amount of time as possible. In Figure 3-40, the driver was already in the process of braking when the full stop request was communicated. Approximately 530ms after the stimulus, the driver reacted by rapidly decreasing the vehicle speed until a complete stop was achieved. In Figure 3-41, the driver was also in the process of braking; however he was already close to a full stop from tracking the previous downward ramp signal. The driver most likely had already reached a full stop before perceiving the full stop stimulus; however the stimulus affirmed the full stop to the driver and the vehicle was held stationary.

### 3.2.4.5: Desensitization Rejection

In this experiment, drivers were subjected to substantial periods of continuous SpeedStrip interaction to examine the influence of desensitization on the interface’s ability to communicate the instantaneous speed error to the driver. The tests were conducted while performing several hours of transformation algorithm testing at the Chicken Hill parking lot on the Virginia Tech campus. Before the testing period began, a driver was asked to track the same arbitrary speed reference signal from Section 3.2.4.3 using the four-element SpeedStrip operation configuration defined in Table 3-7. The arbitrary reference signal and the resulting driver speed response were recorded for comparison to later data. Afterwards, the driver performed approximately 3 hours of system testing, which included constant usage of the SpeedStrip interface. After the 3 hour usage period, the arbitrary signal tracking experiment was conducted once more with the same reference signal and FE SOC utilized previous to the testing period. The driver’s post-testing response was recorded for comparison with the pre-testing response to analyze the effects of desensitization over the three hour usage period.

Figure 3-42 displays the driver’s response to the arbitrary reference signal before and after the 3 hour testing period. The highly similar responses suggest that the desensitization does not impede the driver’s ability to read the instantaneous speed error from the SpeedStrip interface even after a 3 hour period of continuous usage. The only notable differences in the two responses are the steady-state behavior and the faster acceleration response of the pre-testing results at the 27s time location. The differences in steady state behavior are the result of the driver’s discretion in maintaining constant speed within the acceptable error range as discussed in Section 3.2.4.1. The faster acceleration reaction in the pre-testing response at 27s was caused by the driver’s behavior in the acceptable error range prior to the increase in reference
speed. The driver was already in the process of accelerating to try and reach a constant speed when the increase in reference speed occurred, allowing a faster acceleration response by the driver.

With these slight differences accounted for, the similarities of the drivers’ pre and post-testing responses shown in Figure 3-42 clearly indicate that the drivers can reliably track the reference signal even after 3 hours of continuous interface usage. If desensitization were an influential factor, large delays and differences in the acceleration and deceleration rates would be present in the latter response as the driver would be unable to feel or differentiate the vibro-tactile elements of the interface. Examination of Figure 3-42 and the discussion of the results indicate that these differences are not present in the pre and post-testing responses. Therefore, it can be concluded that desensitization does not hinder the driver’s ability to read the instantaneous speed error information from the SpeedStrip interface even in prolonged periods of constant interface usage.

![Figure 3-42: Driver speed response before and after a three hour testing period using the FE SOC defined in Table 3-7.](image)

While this analysis indicates that the overall tracking ability is not hindered by desensitization, it is suggested that future research be conducted to examine the effects of desensitization at the interface level rather than at the response level. Experiments may be conducted to activate each vibro-tactile element of the interface at random and instruct the driver to make a pre-planned action in response to the stimulus created by each element. Such an experiment would directly examine the effects of desensitization at its fundamental source by analyzing how the driver’s tactile perception of the stimuli changes over periods of prolonged usage.
3.3: Simultaneous Interface Operation Analysis

The analysis within this subsection examines the change in performance of the DriveGrip and SpeedStrip non-visual interfaces during simultaneous interaction with the driver as compared to the individual performance analyses previously discussed in Sections 3.1.4 and 3.2.4. In this experiment, a driver was asked to simultaneously track arbitrary steering and speed reference signals using the DriveGrip and SpeedStrip interfaces. The arbitrary reference signals were previously recorded from one minute of actual driver data during the navigation of a single-lane road consisting of multiple curves and straightaways as well as a single stop sign. In an open parking lot, the driver simultaneously tracked the pre-recorded arbitrary steering and speed signals a total of 10 times while the associated responses were recorded. The driver utilized the FFLT (Table 3-2) and TE (Table 3-7) operation configurations for DriveGrip and SpeedStrip as Sections 3.1.4.3 and 3.2.4.3 determined that these configurations induce the highest performance when tracking arbitrary reference signals.

3.3.1: General Performance Analysis

Figure 3-43 displays a typical driver steering and speed response in regard to the quantitative and qualitative characteristics that will be discussed in this subsection. This figure plots the steering and speed responses on dual vertical axes to better depict the relationship between the performance of the DriveGrip and SpeedStrip interfaces during simultaneous operation.

![Figure 3-43: Driver steering and speed responses during simultaneous DriveGrip and SpeedStrip interaction using the FFLT and TE operation configurations defined in Table 3-2 and Table 3-7.](image)

Observation of Figure 3-43 suggests that the driver does exhibit the ability to simultaneously track the steering and speed reference signals with a certain degree of
performance. In order to compare this level of combined performance with the individual interface performances discussed in Section 3.1.4.3 and 3.2.4.3, a time-shifted mean absolute error (MAE) analysis was conducted between the typical simultaneous responses and the associated reference signals. The resulting data is plotted in Figure 3-44 and recorded in Table 3-11 below.

![Graph showing time-shifted mean absolute error](image)

**Figure 3-44:** Time-shifted Mean Absolute Error of driver steering and speed response during simultaneous DriveGrip and SpeedStrip interaction using the FFLT and TE operation configurations defined in Table 3-2 and Table 3-7.

**Table 3-11:** Quantitative characteristics of the driver steering and speed responses plotted in Figure 3-44.

<table>
<thead>
<tr>
<th>Non-Visual Interface</th>
<th>Time Lag</th>
<th>Mean Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriveGrip (FFLT)</td>
<td>400 ms</td>
<td>2.29°</td>
</tr>
<tr>
<td>SpeedStrip (TE)</td>
<td>1950 ms</td>
<td>0.36 m/s</td>
</tr>
</tbody>
</table>

When compared to the individual performance characteristics, the simultaneous performance characteristics exhibit similar average time delays with increased average MAE for the respective interface operation configurations used. These results suggest that the influence of using both interfaces simultaneously has negligible effects on time delay but generally decrease the driver's ability to match the reference signals. The similarities in the time delay demonstrate that the strategic positioning of the vibro-tactile elements of each interface on primary and secondary tactile sensation areas of the body successfully decouple the two sources of stimuli and enable the driver to perceive and differentiate the stimuli as though the interfaces were being used independently.

Conversely, the simultaneous operation does exhibit influence over the driver's reaction to the separately perceived stimuli. When compared to the average MAEs induced by the independent use of each interface, the MAEs exhibited by the simultaneous steering and speed responses are increased by 116% and 90%, respectively. The increases in
MAE occur because the driver must now process and react to two separate stimuli and decreases the amount of cognitive focus and fine motor skill that can be applied to each. Interestingly, splitting the driver’s focus between the two tracking duties approximately doubles the MAE exhibited by each tracking duty. Although the MAE is increased by a considerable amount in this case, the percentage error from the absolute mean of the reference signals are only increased from 2.7% to 5.7% for steering and from 5.7% to 9.1% for speed. Whether these levels of error are acceptable or not will be determined by the combined abilities of the TORC Robotics Motion Planning software and the transformation algorithms presented in Section 4 and 0 of this work.

3.3.2: Speed Tracking Performance Lapses

A quick glance at the responses shown previously in Figure 3-43 indicates that sudden lapses in speed tracking performance may occur when complex steering reference signals are present. This is made most evident in the 10-20s timeframe when a large performance decrease in speed tracking occurs simultaneously with a significant change in the steering reference signal. Close inspection of the performance lapse shows that the driver momentarily halts corrections in speed during this timeframe. This is a result of the driver temporarily ignoring the instantaneous speed error communicated by the SpeedStrip interface to focus more on tracking the significant change in the steering reference signal. As described in Section 3.2.1, drivers are purposefully trained to exhibit this type of behavior and assign priority to the steering reference signal if tracking both steering and speed simultaneously becomes too complex. When combined with the full-stop request training also described in Section 3.2.1, this training helps avoid collisions and maximize the safety of the system.

The performance lapses in reference speed tracking do not necessarily occur each and every time a complex steering action transpires. The occurrence of the performance lapse relies heavily on the complex cognitive state and abilities of the driver and appears stochastically with the presence of complex steering actions. This is demonstrated by the typical driver response shown in Figure 3-43, wherein only a single lapse in speed tracking performance manifests even though several large and complex steering actions are required.

While this discussion indicates that the driver will occasionally ignore SpeedStrip information in complex steering scenarios, it is important to note that driver does not accidentally ignore full-stop requests due to the full-back vibration stimulus used. This stimulus was designed to overpower all other stimuli in any driving scenario to ensure that the driver is instantly made aware of the full-stop request and can react accordingly to ensure the safety of the vehicle and outside environment. The driver properly reacted to the two full-stop requests in all ten of the arbitrary signal tests and demonstrates this behavior in the typical responses given previously in Figure 3-43.
3.4: Non-Visual Interface Controller (NVIC)

The Non-Visual Interface Controller (NVIC) is the piece of equipment responsible for controlling the command signals governing the DriveGrip and SpeedStrip non-visual interfaces. The NVIC acts as a switchbox that permits the non-visual interface computer (Section 3.5) to control power flowing through each of the vibration components in the collective set of non-visual interfaces. This subsection describes the low-level functionality, development, and implementation of the NVIC in the Blind Driver Challenge® platform.

3.4.1: Functionality

As previously described, the NVIC’s main purpose is to serve as a transparent switchbox that permits the NVI computer (Section 3.5) to control the power to each vibration component of the non-visual interfaces. In order to accomplish this, the NVIC must create an electrical power loop containing each independent vibration component and have a method for opening and closing each of these loops. Sections 3.1 and 3.2 state that each vibration component of the DriveGrip and SpeedStrip NVIs contains a common anode and separated cathode. The NVIC electrical loops are implemented by connecting the common anode of the vibration components to a positive voltage supply and placing a computer-controlled switch between the electrical ground reference and the separated cathodes. This creates a separate power loop for each vibration component of the NVIs that can be independently controlled by the non-visual interface computer through the NVIC. Figure 3-45 shows a simplified circuit of an electrical loop for the control of a single vibrational component from the non-visual interfaces.

![Figure 3-45: Example schematic of single non-visual interface vibration element.](image-url)
3.4.2: Controller Development

The Non-Visual Interface Controller was developed in order to meet the electrical needs of the NVIs and to create a simple interface with the non-visual interface computer (Section 3.5). This subsection presents the overall layout design (including component specification) as well as the electrical schematic and images of the fully assembled controller.

3.4.2.1: Layout and Component Specification

A high-level layout of the NVIC is depicted in Figure 3-46. This layout includes all of the necessary components needed to meet the functionality requirements of the NVIC stated in Section 3.4.1. A detailed specification of each primary component is given in the following text.

![Diagram of Non-Visual Interface Controller](image)

**Figure 3-46: Non-Visual Interface high level layout design.**

The Digital I/O computer interface is a National Instruments® USB-6501 OEM digital input/output PCB (Figure 3-47) used to interface the NVI computer with the NVIC switches. This piece of hardware provides 24 channels of reconfigurable digital input/output ports that can be controlled independently by a computer via USB. In our application, the channels of the USB-6501 are all configured for output to enable control of the NVIC switches. Each digital output channel operates at a 5VDC logic level...
(common to TTL) that can drive up to 8.5mA individually and 65mA collectively. The logic level can be controlled at a rate of up to 5MHz, although our application operates at approximately 0.02% of this frequency. The 24 channels are split into 3 groups of 8 channels; 1 group for controlling DriveGrip, a second for SpeedStrip, and a third for possible future expansions of the NVIC or non-visual interfaces.


The physical switches controlling the vibration components of the NVIs are Texas Instruments® ULN2803A high voltage/current Darlington transistor arrays. These integrated circuits contain eight NPN Darlington pairs with cathode clamp diodes that can support switching of high voltage and high current inductive loads. The Darlington pairs accept 5VDC logic level inputs and are designed to switch electrical loops directly to the reference ground. Each pair has a collector current rating of 500mA, providing plenty of power for the 45mA DriveGrip motors and 140(70x2)mA SpeedStrip motors.

The Darlington Array ICs are connected directly to the USB-6501 through a custom PCB board presented in this paper (Figure 3-48). This interface board simply provides a mounting point for the Darling Array ICs and also supplies 10-pin male rectangular box headers for connecting the IC outputs to the DB9 outputs of the NVIC. The double layer PCB contains all necessary traces to connect the digital output pins of the USB 6501 to the ICs and the outputs of the ICs to the box header output ports.
Figure 3-48: Custom Darlington Array Interface PCB used in the Non-Visual Interface Controller.

The components of the Non-Visual Interface Controller are all housed within a 5x2.5x2in RadioShack® Project Enclosure. This enclosure helps contain and protect the internal components of the NVIC as well as serve as a mounting point for the Blind Driver Challenge® platform. The enclosure is made from durable, yet easy to machine ABS plastic. A laser-cut acrylic plate is included within the enclosure to provide rigid mounting holes for the USB-6501 DIO board. Ventilation is additionally milled into the enclosure for proper heat dissipation from the internal components. A RadioShack SPST rocker power switch is positioned on the top of the enclosure to disable power to the non-visual interfaces.

3.4.2.2: Electrical Schematic

Figure 3-49 displays the electrical schematic of the Non-Visual Interface Controller. The circuit first receives digital output commands from the non-visual interface computer through the USB connection to the USB-6501. The USB-6501 then alters the logic level of each output pin accordingly, which in turn activates/deactivates the associated Darlington transistor pair. When the Darlington pair is activated (saturated), current is allowed to flow from the “output” pin of the array IC to the circuits reference ground. This acts as a switch that can open or close the electrical loop containing the power source and the non-visual interface anode/cathode loop. When the switch is closed and current can flow to ground, the current first passes through the non-visual interface vibration components and powers them in an ON state. When the switch is open, no current passes through the vibration components and they remain in an OFF state.
The NVIC includes a 12VDC power input port that is used to power the non-visual interfaces. The power input port uses the same interface as the TORC Robotics PowerHub™ output ports: a Molex 39-30-1022 2-pin female connector. The positive line from the input port first passes through a single-pole-single-throw (SPST) switch for power switching and then branches into two separate power lines. The input and output
wires of the SPST switch are connected through Molex 50-36-1678 and 39-03-9022 connectors so that the lid of the enclosure box can be removed. The first power line, along with the input power ground line, is passed directly into the 12VDC input line of the custom PCB supporting the Darlington Array ICs to power the SpeedStrip vibration components. The second power line includes a 270Ω, 0.5W resistor in series to drop the 12VDC input to 5VDC before it is routed into the custom PCB. The 5VDC line is used to power the DriveGrip vibration components and shares a common ground with the 12VDC line.

The outputs of the NVIC include two IDC female DB9 connectors: one for connection with the DriveGrip Adapter Box, and the second for connection directly with SpeedStrip. These output connectors are connected to the custom PCB support board through a 9-conductor ribbon cable. The ribbon cable connects to the PCB via 10-pin IDC female box headers that mate with the male box headers on the PCB support board. As seen in the schematic (Figure 3-49), DriveGrip connects with Port 0 on the PCB while SpeedStrip connects to Port 1 on the PCB. Port 2 is left open for future expansions of the NVIC. These port numbers are directly associated with how software running on the non-visual interface computer controls the necessary digital output pins on the USB-6501.

3.4.2.3: Completed Design

Figure 3-50 pictures the final assembly of the Non-Visual Interface Controller. This image also displays the USB and power input ports as well as the power switch and dual DB9 output ports. Figure 3-51 shows the internals of the completed NVIC design, including the USB-6501 OEM, custom PCB with ICs, and the output port wiring layout.
3.4.3: Platform Implementation

The Non-Visual Interface Controller is integrated with the vehicle through a Velcro® mount beneath the driver seat. The NVIC is small enough in size to be safely stowed under the seat and away from contact even with rear-seat passengers. The DB9 ribbon cables used to connect the NVIC with SpeedStrip and the DriveGrip Adapter Box link with the NVIC DB9 output ports and then immediately pass through crease between the seat and back of the driver seat. The SpeedStrip cable directly attaches to the SpeedStrip interface within the driver seat crease. The DriveGrip cable runs up the back of the driver seat, between SpeedStrip and the seat back, and connects to the DriveGrip Adapter Box located on the NVI headrest mount.

The USB and power wires linking the NVIC with the non-visual interface computer and TORC Robotics PowerHub™ is routed from the NVIC underneath the driver seat to the trunk of the vehicle where the computer and PowerHub™ are located. The wires are cut to 9ft in length and are routed underneath the left-side threshold panels of the rear passenger door until they reach the trunk of the vehicle. This helps protect the USB and power wires while additionally hiding them from view and maintaining the stock appearance of the vehicle.
3.5: Non-Visual Interface Computer

The Non-Visual Interface (NVICPU) hosts the software that calculates and generates the non-visual stimuli communicated through the DriveGrip and SpeedStrip interfaces. The JAUS interoperable (Section 2.4.4) software running on this computer interacts with the rest of the system over the central Ethernet communication network to obtain semi-autonomy data and calculate the appropriate non-visual stimuli. The computer is linked to the Non-Visual Interface Controller through a USB connection to enable physical control of the NVIs by the generation component of the hosted software.

Currently, the NVI computer consists of an ASUS G53 Laptop placed in the trunk of the vehicle. The purpose for a mobile computing solution is to enable remote NVI control and promote software access for debugging during the developmental process. The specifications of the ASUS G53 Laptop used in this application are provided Table 3-12.

<table>
<thead>
<tr>
<th>Table 3-12: Non-Visual Interface Computer Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
</tr>
<tr>
<td><strong>Chipset</strong></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td><strong>Hard Drive</strong></td>
</tr>
<tr>
<td><strong>External Ports</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
3.6: Non-Visual Interface System Connection Architecture

The connection architecture of the complete Non-Visual Interface system implemented on the Blind Driver Challenge® platform is depicted in Figure 3-52. The Non-Visual Interface Computer (NVICPU) is powered through an AC adapter connected to the Tripp-Lite Uninterrupted Power Supply (UPS) described in Section 2.1.2.4. The NVICPU interfaces with the vehicle system network (Section 2.4) through a LAN-based Ethernet interface. As commands are received over the Ethernet network, they are forwarded to the Non-Visual Interface Controller (Section 3.4) over a USB connection. The NVIC internally enables the appropriate 12VDC and 5VDC power loops, which travel through proprietary 9-conductor ribbon cables into SpeedStrip (Section 3.2) and the DriveGrip Breakout Box (Section 3.1.3.1). The DriveGrip Breakout Box splits the 9-conductor cable from the NVIC into two proprietary, Ethernet-based cables which directly connect to the DriveGrip Gloves (Section 3.1).

Figure 3-52: Non-Visual Interface System Connection Architecture
Section 4: The Passive Non-Visual Interface Driver (PNVID)

The Passive Non-Visual Interface Driver (PNVID) is a specialized software component designed to passively generate non-visual interface stimuli based on trajectories planned by the TORC Robotics AutonoNav™ system. This software acts as a reference transformation technology and solves the third issue of the Problem Statement (Section 1.3) by converting the planned trajectory information into non-visual stimuli that can be reliably understood and interacted with by the driver. The now complete instructional solution to the Blind Driver Challenge® utilizing the PNVID as a reference transformation technology is shown in Figure 4-1.

Figure 4-1: Complete instructional solution to the Blind Driver Challenge® utilizing the PNVID software. The Research Platform refers to the Blind Driver Challenge® Research Platform and is described in detail in Section 2. The NVIS refers to the Non-Visual Interface System and is presented in detail in Section 3.

This section focuses on providing the detailed functional requirements, algorithm definitions, and software implementation of the PNVID component. Afterwards, a thorough performance analysis is conducted to examine how well the driver can navigate through various driving environments utilizing the complete instructional solution with the PNVID reference transformation software. Finally, this section will be concluded with a discussion focusing on the first public demonstration of the Blind Driver Challenge® at the Rolex 24 GRAND-AM Race in January of 2011.
4.1: Functional Requirements

The primary function of the PNVID software is to transform planned driving actions from the TORC Robotics AutonoNav™ system into non-visual stimuli that can be communicated over the Non-Visual Interface System. As described in Section 2.3.2.2, the Motion Planner of the AutonoNav™ generates desired trajectories for the vehicle to follow at a variable rate of 2-8Hz and encodes them into motion profiles through a trajectory model. The motion profiles, defined in Section 2.3.2.2.3, contain sequential curvature and velocity data that can be transformed into non-visual stimuli and communicated to the driver so that he or she physically realizes the desired trajectory of the AutonoNav™ motion planner. The purpose of the PNVID software is to perform this transformation passively and generate the stimuli that are communicated over the Non-Visual Interface System described in Section 3. Unfortunately, this transformation is not straightforward due to several problems posed by the behavior of the TORC AutonoNav™ Motion Planner and the Non-Visual Interface System. These problems are described in detail throughout the remainder of this subsection and are later solved in the PNVID Algorithm Definition.

4.1.1: Motion Profile Sawtooth Patterns

As previously shown in Figure 2-22 of Section 2.3.2.2.3, the TORC AutonoNav™ motion profiles describe a sequence of curvatures and velocities that must be achieved in order to realize the desired trajectory from the Motion Planner. Each sequence is defined as a single motion and includes a time duration that describes the length of time to execute the motion for. The motions of the motion profile typically include 2-3s of future curvature and velocity information; yet a new motion profile is re-generated every 125-500ms (2-8Hz). The motion profiles are re-planned by the Motion Planner relatively quickly so that the driver can constantly compensate for changes in the vehicle’s position or for changes in the surrounding environment. However, the rapid regeneration of motion profiles causes equally rapid sawtooth patterns to emerge in the non-visual interface stimuli communicated to the driver.

In order to understand how these sawtooth patterns emerge, it is important to first understand how the motion profiles are used to generate non-visual interface stimuli. Review of Sections 3.1.1 and 3.2.1 indicates that the DriveGrip and SpeedStrip non-visual interfaces create different vibratory stimuli based on the instantaneous difference between the reference value and the actual value for steering wheel angle and speed, respectively. The actual values are determined by directly measuring the steering wheel angle and speed from the vehicle, while the reference values are derived from the AutonoNav™ motion profiles. Because of this derivation, any patterns present in the reference value signals are directly injected into the instantaneous error and thus the vibratory stimuli created by the Non-Visual Interface System.
The reference steering wheel angle signals are derived from the motion profiles by incrementally stepping through time for each motion within the current profile. The desired curvature signal described by the motion profile always starts at the current actual curvature and then makes the necessary ramp changes to achieve the desired curvature for each subsequent motion. Figure 4-2 displays this behavior with a motion profile that describes the desired curvature for 2.5s after the current time. The future desired curvature signal effectively becomes a reference for the driver to track for the time duration of the profile. Once the time duration is exhausted, the reference curvature signal ends and the driver no longer has a reference to track.

![Figure 4-2: Future desired curvature reference signal derived from the most recent motion profile and the current actual curvature.](image)

The sawtooth problem is created by the combination of this derivation and the rapid regeneration of the motion profiles. Each time a new motion profile is generated, the previous profile is discarded and the driver begins to track the new profile. Since the motion profiles are regenerated every 125-500ms, each motion profile is typically replaced well before its time duration is exhausted. The replacement often prevents progression through each motion profile past the initial ramp of its first motion. Additionally, the initial desired curvature of each new profile is always set back to the current actual curvature. This behavior is clearly exemplified in Figure 4-3. The rapid repetition of this behavior every 125-500ms is what causes the sawtooth pattern to continuously emerge in the reference curvature signal. Actual results of deriving reference curvature signals from the TORC AutonoNav™ system in this manner have been recorded and are displayed in Figure 4-4.
Figure 4-3: Desired curvature sawtooth pattern created by rapid regeneration of motion profiles. The motion profile regeneration intervals are denoted by the red dashed lines and, in this example, occur at 4Hz. The dotted lines indicate discarded motion profile data. The heavy black line displays the resulting desired curvature signal.

Figure 4-4: Actual desired curvature sawtooth pattern derived from TORC AutonoNav™ motion profile data during a driving experiment.

Although the reference curvature exhibits substantial sawtooth patterns, the stimuli created by the DriveGrip interface using in Section 3.1.1 is calculated from the reference steering wheel angle and not curvature. The curvatures defined by the TORC AutonoNav™ are actually derived from the steering wheel angle and the vehicle speed [98]:
where $C$ is the instantaneous curvature, $\theta$ is the instantaneous steering wheel angle in degrees, $R$ is the vehicle steering ratio, $v$ is the instantaneous vehicle velocity, $k$ is the vehicle understeer coefficient, and $L$ is the vehicle wheelbase. The steering wheel angle reference can then be reversely derived from the curvature reference by solving (4.1) for $\theta$:

$$\theta = \frac{180R(1 + v^2k)\tan^{-1}(CL)}{\pi}$$

(4.2)

This equation indicates that the reference steering wheel angle that the driver must track with the DriveGrip interface has an inverse tangential relationship with the reference curvature. Fortunately, the properties of the TORC ByWire XGV™ shown in Table 4-1 force this relationship to exist only in the near-linear range of the inverse tangential function, as shown in Figure 4-5.

**Table 4-1: TORC Robotics ByWire XGV™ research platform steering properties.**

<table>
<thead>
<tr>
<th>Property</th>
<th>Nomenclature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheelbase</td>
<td>$L$</td>
<td>2.619m</td>
</tr>
<tr>
<td>Steering Ratio</td>
<td>$R$</td>
<td>19.99</td>
</tr>
<tr>
<td>Understeer</td>
<td>$k$</td>
<td>0.0015</td>
</tr>
</tbody>
</table>

**Figure 4-5: Approximate linear relationship between steering wheel angle and vehicle curvature at varying speeds defined by (4.2).**
This near-linear relationship suggests that the reference steering wheel angle signal is approximately proportional to the reference curvature signal. Therefore, the rapid sawtooth wave pattern within the reference curvature is expected to propagate into the steering wheel angle reference signal that the driver must track. The propagation is exemplified in Figure 4-6 by applying (4.2) to the actual curvature data shown previously in Figure 4-4. This figure shows that the sawtooth waves do in fact proportionally propagate into the reference steering wheel angle; however the increasing velocity also causes a slight increase in the sawtooth wave amplitude proportional to the squared velocity due to the relationship in (4.2).

![Figure 4-6: Propagation of actual sawtooth patterns from Figure 4-4 into the desired steering wheel angle signal through (4.2).](image)

The main problem caused by the presence of sawtooth waves in the reference steering wheel angle signal is that the waves propagate into the instantaneous steering wheel angle error through (3.1). Recalling from Section 3.1.1, the instantaneous error directly designates which vibro-tactile element will be activated on the DriveGrip interface and thus designates the non-visual stimuli that are communicated to the driver. In most cases, the teeth propagated into the instantaneous error signal straddle one or more of the steering wheel angle error thresholds and cause rapid traversals of vibrations across the driver’s fingers. These rapid traversals cannot be understood by the driver and lead to confusion and a general loss of tracking ability. This behavior is clearly demonstrated in Figure 4-7 by calculating the instantaneous error with (3.1) for the steering wheel angle data shown in Figure 4-6 and plotting it against the FFLT DOC error thresholds previously defined in Table 3-2 of Section 3.1.4.
Figure 4-7: Propagation of sawtooth patterns from the desired steering wheel angle error (Figure 4-6) into the instantaneous steering wheel angle error signal through (3.1). The FFLT DOC error thresholds defined in Table 3-2 are overlaid to demonstrate the sawtooth pattern propagation into the non-visual stimuli created by the DriveGrip interface.

Since the sawtooth waves dramatically interfere with the driver’s ability to track the reference steering signal, the PNVID software must eradicate the waves from the reference and ensure that the resulting non-visual stimuli can be reliably understood and interacted with by the driver. The software must also perform this transformation in a way that minimizes additional time lag injected into the system to avoid further increases in the ultimate latency of the driver’s responses. For this reason, filtering methods such as low-pass filters or moving averages cannot be used to mitigate the sawtooth patterns.

4.1.2: Reference Signal Ramp Leading

It has already been discussed in Sections 3.1.4.2 and 3.2.4.2 that poor tracking ability and significant deviations from the intended vehicle trajectory can occur if a driver’s ramp response leads its corresponding ramp reference signal. Leading the reference signal ramp can often cause full ramp direction reversal in the driver’s response because the interfaces are only designed for instantaneous value tracking and not for rate tracking. The PNVID software must be able to detect this situation and modify the non-visual stimuli communicated to the driver to avoid ramp direction reversals in the driver’s response and thus maintain a safe level of tracking accuracy. This will help
improve the navigation performance of the driver by creating smaller deviations from the intended vehicle trajectory generated by the TORC AutonoNav™ Motion Planner.

4.1.3: Driver Time Delay
It has also already been discussed in Sections 3.1.4 and 3.2.4 that driver responses to reference steering and speed signals exhibit varying amounts of pure time delay that limit his or her ability to accurately recreate vehicle trajectories planned by the TORC Robotics AutonoNav™ system. Keeping in mind that these time delays can vary based on different drivers and situations, the PNVID software must mitigate the trajectory deviations that are caused by the pure time delays in the driver’s steering and speed responses. By using future planning data to account for the time delays, the PNVID can improve the driver’s accuracy in recreating the planned trajectories and increase the stability of the closed loop system.
4.2: Algorithm Definitions

The PNVID software component is constructed of nested algorithms that cooperate in both series and parallel to ultimately generate non-visual stimuli based on TORC AutonoNav™ planned trajectory data. The purpose of this subsection is to provide a detailed description of these algorithms and discuss how the functionalities of each contribute to the overall goal of the PNVID software. Each description will supply pseudocode that defines the algorithm in a simplistic and easy to follow manner. A legend of color-codes used in the pseudo-code descriptions has been provided in Algorithm 4-1.

Algorithm 4-1: Pseudocode Legend

<table>
<thead>
<tr>
<th>% Comments</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BuiltInFunction()</td>
<td>% Identifies a function or subroutine that is not necessary to explicitly define.</td>
</tr>
<tr>
<td>CustomFunction()</td>
<td>% Identifies a custom function or subroutine that will be explicitly defined.</td>
</tr>
<tr>
<td>PrivateVariables</td>
<td>% Identifies variables private to the parent function or subroutine.</td>
</tr>
<tr>
<td>PublicVariables</td>
<td>% Identifies variables public to all functions and subroutines.</td>
</tr>
</tbody>
</table>

The PNVID algorithm contains several levels of nested sub-algorithms. In order to present each sub-algorithm in an intuitive manner, the sub-algorithms will be described in the order at which they are first presented. The descriptions will begin with the PNVID() algorithm and continue through each sub-algorithm until all have been defined and discussed. Table 4-2 displays the hierarchy of the nested PNVID structure and identifies the section in which each sub-algorithm is defined and discussed.

Table 4-2: PNVID algorithm hierarchy. Each sub-algorithm is listed along with the corresponding section it is described within.

<table>
<thead>
<tr>
<th>Algorithm Nest Structure</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: PNVID()</td>
<td>4.2.1</td>
</tr>
<tr>
<td>1.1: CalculateNonVisualStimuli()</td>
<td>4.2.2</td>
</tr>
<tr>
<td>1.1.1: CalculateDriveGripStimuli()</td>
<td>4.2.3</td>
</tr>
<tr>
<td>1.1.1.1: CalculateSteeringError()</td>
<td>4.2.4</td>
</tr>
<tr>
<td>1.1.1.1.1: FilterSawtooth()</td>
<td>4.2.5</td>
</tr>
<tr>
<td>1.1.1.1.2: CheckRampLead()</td>
<td>4.2.6</td>
</tr>
<tr>
<td>1.1.1.2: ConvertToDriveGripStimulus</td>
<td>4.2.7</td>
</tr>
<tr>
<td>1.1.2: CalculateSpeedStripStimuli()</td>
<td>4.2.8</td>
</tr>
<tr>
<td>1.1.2.1: ConvertToSpeedStripStimulus</td>
<td>4.2.9</td>
</tr>
<tr>
<td>1.2 GenerateNonVisualStimuli()</td>
<td>4.2.10</td>
</tr>
</tbody>
</table>
4.2.1: PNVID()

The pseudocode describing the PNVID() algorithm is given below in Algorithm 4-2. The algorithm begins by immediately starting two parallel processes. These processes work in parallel to simultaneously calculate and generate the non-visual stimuli that are communicated through the DriveGrip and SpeedStrip interfaces (Section 3). The first algorithm, CalculateNonVisualStimuli(), calculates the stimuli to be activated on each interface based on the trajectories planned by the TORC AutonoNav™ Motion Planner (Section 2.3.2.2). A complete description of this sub-algorithm can be found in Section 4.2.2. The second algorithm, GenerateNonVisualStimuli(), reads the calculated stimuli from CalculateNonVisualStimuli() and activates the corresponding vibro-tactile elements of the DriveGrip and SpeedStrip interfaces. A complete description of this sub-algorithm can be found in Section 4.2.10. These parallel processes run indefinitely until the user requests a software stop.

Algorithm 4-2: PNVID() Pseudocode

```
Main PNVID()
  Start Parallel Processes
    CalculateNonVisualStimuli()
    GenerateNonVisualStimuli()
  End Parallel Processes
End Main
```

The simultaneous calculation and generation is made possible through the use of two separate loops in parallel. The advantage of a parallel structure is that it completely decouples the execution of each sub-algorithm while still enabling data communication between them. This greatly simplifies implementation and troubleshooting because the execution of each sub-algorithm is completely separate. The only interaction between the calculation and generation of the non-visual stimuli is the one-way passing of calculated stimuli data. An additional advantage posed by the parallel structure is that it can operate in true parallel on multicore processor computers, such as the Non-Visual Interface Computer described in Section 3.5.
4.2.2: CalculateNonVisualStimuli()

The purpose of the CalculateNonVisualStimuli() algorithm is to continuously calculate non-visual stimuli based on both the TORC AutonoNav™ Motion Planner trajectories and the driver’s steering and speed actions. This algorithm was designed to solve the 3rd issue of the Problem Statement (Section 1.3) and thus transform the planned trajectories into non-visual stimuli that can be reliably understood and interacted with by the driver. The pseudocode that defines this algorithm is presented in Algorithm 4-3.

Algorithm 4-3: CalculateNonVisualStimuli() Pseudocode

```
Subroutine CalculateNonVisualStimuli()
    While StopRequested = False
        Wait for 25ms Interval
        If NewMotionProfileFlag = True Then
            NewMotionProfileFlag = False
            Start Parallel Processes
                CalculateDriveGripStimuli()
                CalculateSpeedStripStimuli()
            End Parallel Processes
        End If
    End While
End Subroutine
```

The algorithm begins by starting a timed while loop that runs at 25ms intervals until the user requests a software stop. The software stop is a global variable that is controlled by the user through a GUI and notifies all algorithms to stop operating when requested. Inside of the timed loop, the algorithm continuously waits for a notification that a new motion profile has been received from the TORC AutonoNav™ Motion Planner through the NewMotionProfileFlag public variable. Whenever a new motion is sent by the planner, it is received by a separate piece of software called the Message Handler. The message handler parses the new motion profile message, stores it in the CurrentMotionProfile public variable, and sets the NewMotionProfileFlag to true.

Once the CalculateNonVisualStimuli() algorithm recognizes that a new motion profile is available, it resets the NewMotionProfileFlag to false and starts two parallel processes. These processes perform simultaneous calculations of non-visual stimuli for the DriveGrip and SpeedStrip non-visual interfaces based on the reference steering and speed values defined by the current motion profile. These sub-algorithms, called CalculateDriveGripStimuli() and CalculateSpeedStripStimuli(), are fully defined in Section 4.2.3 and Section 4.2.8. Once the stimuli calculation sub-algorithms complete, the CalculateNonVisualStimuli() algorithm reiterates until a software stop is requested by the user.
4.2.3: CalculateDriveGripStimuli()

The purpose of the CalculateDriveGripStimuli() algorithm is to continuously calculate the instantaneous DriveGrip stimulus based on trajectories generated by the TORC AutonoNav™ Motion Planner and the driver’s steering actions. The stimuli are calculated by determining the instantaneous steering wheel angle error through (3.1) and converting the error into a stimulus based on Table 3-1 in Section 3.1.1. The pseudocode that defines this algorithm in its entirety is presented in Algorithm 4-4.

Algorithm 4-4: CalculateDriveGripStimuli() Pseudocode

```plaintext
Subroutine CalculateDriveGripStimuli()
    For each Motion in CurrentMotionProfile
        MotionStartTime = ReadCurrentTime()
        While ReadCurrentTime() - MotionStartTime < Motion.Duration and StopRequested = False and _
            _NewMotionProfileFlag = False
            Wait for 25ms Interval
            InstantaneousSteeringError = CalculateSteeringError(Motion, MotionStartTime)
            DriveGripStimulus = ConvertToDriveGripStimulus(InstantaneousSteeringError)
        End While
        If StopRequested = True or NewMotionProfileFlag = True Then
            Exit For Loop
        End If
    End For
End Subroutine
```

The algorithm functions by iterating in time through each motion of the current motion profile. Recalling from Section 4.2.2, the CurrentMotionProfile public variable is set by the Message Handler whenever a new motion profile is sent by the Motion Planner. The algorithm begins by reading the first motion of the motion profile and stores it in the Motion variable. The time at which the motion was read is then found through the ReadCurrentTime() function and stored in the MotionStartTime variable. Next, a time while loop is started with an interval of 25ms to iterate through the Motion and evaluate the instantaneous DriveGrip stimulus at each 25ms time step. During each while loop iteration, the instantaneous steering wheel angle error is calculated through the CalculateSteeringError() function defined in Section 4.2.4 and stored in the InstantaneousSteeringError variable. The instantaneous error is then converted to a DriveGrip stimulus through the ConvertToDriveGripStimulus() function defined in Section 4.2.7 and the resulting stimulus is stored in the DriveGripStimulus public variable for parallel use by the GenerateNonVisualStimuli() algorithm (Section 4.2.10).

Once the difference between the current time and the MotionStartTime exceeds the motion duration, the motion has been exhausted and the while loop exits. The while loop additionally exits at any time if a software stop has been requested or a new motion profile becomes available. Upon exiting the while loop, the StopRequested and NewMotionProfileFlag variables are checked. If both are false, the algorithm begins the
next iteration using the next motion of the motion profile until the profile is exhausted. If either of the variables are true, the For loop exits and the algorithm ends.

4.2.4: CalculateSteeringError()

The purpose of the CalculateSteeringError() function is to calculate the instantaneous steering angle error at each iterative step through the motions of the current motion profile. This algorithm utilizes (3.1) and a series of modification techniques to calculate the instantaneous error in a way that improves the reference tracking performance of the driver. Pseudocode has been provided in Algorithm 4-5 to fully define the functionality of this algorithm.

Algorithm 4-5: CalculateSteeringError() Pseudocode

```plaintext
Function CalculateSteeringError(Motion) Returns InstantaneousSteeringError
    ReferenceSteeringAngle = FilterSawtooth(Motion)
    % Modify error to mitigate ramp leading tracking errors
    RampDelta = ReferenceSteeringAngle - ReferenceSteeringAngle_{i-10}
    If |RampDelta| > 10 Then
        % Reference signal is ramping.
        LeadingRamp = CheckRampLead(RampDelta)
    Else
        % Reference signal is not ramping.
        LeadingRamp = False
    End If
    If LeadingRamp = True Then
        InstantaneousSteeringError = 0
    Else
        InstantaneousSteeringError = ReferenceSteeringAngle - CurrentSteeringAngle
    End If
    LowPassFilter(InstantaneousSteeringError, f_c = 1Hz)
    Return InstantaneousSteeringError
End Function
```

The algorithm begins by first finding the instantaneous reference steering wheel angle through the FilterSawtooth() function defined in Section 4.2.5 and stores it in the ReferenceSteeringAngle variable. The CurrentSteeringAngle public variable stores the current actual steering wheel angle and is updated at a rate of 25Hz by a separate message handling process that will be described in Section 4.3. Typically, the instantaneous error could then be calculated using (3.1) with the actual and reference steering angles; however the functional requirements in Section 4.1.2 state that the algorithm must take certain steps to mitigate ramp leading situations.

The ramp mitigation process begins by first detecting if the reference signal is currently ramping. This is done by calculating a current finite-difference derivative of the reference signal and comparing it against a threshold. The finite-difference derivative is calculated using the following equation:
where \(i\) is the current iteration index. This finite-difference derivative condenses the past 250ms of reference signal data and finds the relative trend over that time period. The trends are then determined by rearranging (4.3) into a simpler form that can be compared to a threshold value:

\[
0.25\Delta \theta_R = \theta_{R,i} - \theta_{R,i-10}
\]

Using (4.4), \(\Delta \theta_R\) can now be compared to a threshold to determine if a ramp is occurring. If the reference angle changes more than 10° in 250ms, the reference signal is considered to be ramping. Therefore, if the absolute value of \(\Delta \theta_R\) is greater than 10°, the reference signal is either positively or negatively ramping. The algorithm uses this relationship to determine if the reference signal is ramping.

If the reference signal is not ramping, it is not possible for the driver's response to respond with a ramp that leads a ramp in the reference signal. Therefore, the LeadingRamp variable is set to false in this case. If the reference signal is ramping, however, the CheckRampLead() function defined in Section 4.2.6 is used to determine if the driver's response is leading the reference ramp or not. The result of this function is likewise stored in the LeadingRamp variable.

Once it has been determined if a ramp leading situation is currently occurring, the instantaneous steering wheel angle can be calculated. If LeadingRamp is false and thus a ramp leading situation is not present, the algorithm calculates the instantaneous error using (3.1) with the ReferenceSteeringAngle and CurrentSteeringAngle variables. If LeadingRamp is true and a ramp leading situation is present, the algorithm sets the instantaneous error as 0. When the error is set to 0, the no-vibration stimulus is generated and communicated to the driver. Recalling from Section 3.1, the driver is trained to hold the steering wheel steady while this stimulus is present. This tricks the driver into holding the steering angle constant and effectively allowing the ramping reference to catch up with the current steering wheel angle. Essentially, this algorithm enables the ramp leading situation to correct itself rather than trying to correct it through the driver and inducing poor ramp tracking responses as discussed in Section 3.1.4.2.

Finally, the instantaneous error is passed through a point-by-point low pass Butterworth filter to smooth the stimuli communicated to the driver through the DriveGrip interface. The smoothing is required due to the fact that the reference signal calculated by the FilterSawtooth() function is not continuous and causes discontinuities to propagate into the instantaneous error signal through (3.1). These discontinuities ultimately cause the generation of chaotic stimuli on the DriveGrip interface and render reference signal tracking by the driver impossible. Therefore, the low-pass filter is applied with an
empirically defined 1Hz cutoff frequency to create a more continuous error signal and thus more continuous stimuli set at the cost of increasing the overall system lag time. The algorithm completes by performing the LowPassFilter() function on the error and returns the instantaneous steering wheel angle error to the calling algorithm.

4.2.5: FilterSawtooth()

The purpose of the FilterSawtooth() function is to calculate transformed reference steering wheel angle signals that are derived from the current motion profile without injecting additional lag into the Blind Driver Challenge® system. Recalling from Section 4.1.1, the rapid regeneration of motion profiles by the Motion Planner causes sawtooth patterns to emerge in the untransformed reference curvature signals. Since the steering wheel angle is approximately proportional to the curvature, the sawtooth waves emerge in the reference steering wheel angle signal as well. This algorithm uses a series of case decisions to remove the sawtooth patterns at the curvature level and return an instantaneous reference steering wheel angle that can be reliably understood and interacted with by the driver. The sawtooth patterns are essentially removed by forcing the reference value to “ride” the peaks of the sawtooth wave, as exemplified in Figure 4-8. Before the actual pseudocode of the algorithm is presented, it is important to first discuss the case decisions that the algorithm is based upon.

Figure 4-8: Example of sawtooth filtering transformations on the reference curvature signal.

4.2.5.1: Transformation Cases

The sawtooth waves that emerge in the untransformed reference curvature signal exhibit a total of four behaviors that require different transformation actions to mitigate. The transformations determine an instantaneous reference value based on the comparison of the current curvature (CC), the desired curvature (DC), the untransformed reference curvature (URC), and the (PTRC) past transformed reference curvature. The CC is the instantaneous actual curvature that the vehicle is travelling at
and is read directly from the AutonoNav™ system. The DC is the instantaneous curvature required by the Motion Planner to recreate the planned trajectory. This value is specified by the desired curvature component of the current motion that the CalculateDriveGripStimuli() algorithm is iterating through. The URC is the untransformed reference curvature that is calculated by evaluating through the current motion profile in time. The URC is what exhibits the sawtooth patterns shown in Section 4.1.1. The PTRC is the transformed reference curvature calculated by the previous call of FilterSawtooth(). Using the PTRC in the transformation calculation is what enables persistence of the transformed reference curvature (TRC) signal and thus the ability to “ride” the peaks of the sawtooth wave.

Figure 4-9 depicts a simplified example of the four cases determined by the comparisons of the CC, DC, URC, and PTRC. This figure labels each case from C1 to C4 and additionally overlays the transformed action that is taken to mitigate the corresponding case. Detailed documentation of these cases and the corresponding transformations will be provided in the following subsections.

Figure 4-9: Sawtooth pattern behaviors and associated transformation actions. Each of the eight cases is labeled from C1 through C8.

4.2.5.1.1: Case 1 – Ramp
Recalling from Section 2.3.2.2.3, each motion within a motion profile includes a desired curvature (DC) and a curvature rate of change. The curvature rate of change defines a ramp rate, while the position of the DC in relation to the current curvature (CC) defines a ramp direction. The ramps always begin at the CC and thus always increase the distance between the CC and the untransformed reference curvature (URC). This case occurs when the ramping of the URC surpasses the transformed reference curvature from the previous transformation (PTRC) as it travels towards the DC in either direction.
Although this case describes different ramping directions, the same transformation action is still taken to determine the new instantaneous transformed reference curvature (TRC). Since the TRC is designed to “ride” the peaks of the sawtooth waves, the TRC must always increase with the URC in distance from the CC. Therefore, the instantaneous TRC is always set equal to the instantaneous URC in this case. Figure 4-10 exemplifies this behavior in more detail for both the upward and downward ramps exhibited by the URC. The complete detection conditions and the resulting transformation action of this case are recorded in Table 4-3. A high level demonstration of the TRC calculations in this case can be seen in Figure 4-9.

![Diagram of Upward and Downward Ramps]

**Figure 4-10: Detailed example of Case 1 behaviors and transformation action.**

**Table 4-3: Case 1 detection conditions and transformation action.**

<table>
<thead>
<tr>
<th>Case</th>
<th>Sub-Case</th>
<th>Detection Conditions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Upward Ramp</td>
<td>DC &gt; CC &amp; PTRC &gt; CC &amp; DC &gt; PTRC &amp; URC &gt; PTRC</td>
<td>TRC = URC</td>
</tr>
<tr>
<td></td>
<td>Downward Ramp</td>
<td>DC &lt; CC &amp; PTRC &lt; CC &amp; DC &lt; PTRC &amp; URC &lt; PTRC</td>
<td></td>
</tr>
</tbody>
</table>

**4.2.5.1.2: Case 2 – Level**

The purpose of Case 2 is to detect when the URC has reset back to the CC and hold the TRC constant as long as the URC continues to ramp towards the TRC. This detection and resulting action is what enables the TRC to “ride” the peaks of the sawtooth wave and creates a much simpler, yet still accurate, reference for the driver to attempt to track. This case occurs when the URC falls below the TRC and the DC still remains further from the CC than the TRC. Comparing the positions of the DC and TRC in relation to the CC enables the algorithm to determine if the URC is still ramping towards the TRC. The TRC should only remain constant if the reset URC is still ramping towards the TRC. If the reset URC is ramping away from the TRC, then a switch case (Case 4) has occurred.
Figure 4-11 exemplifies this behavior in more detail for both upward and downward URC ramps. The complete detection conditions and the resulting transformation action of this case are recorded in Table 4-4. A high level demonstration of the TRC calculations in this case can be seen in Figure 4-9.

![Graph showing Upward and Downward Ramps](image)

**Figure 4-11:** Detailed example of Case 2 behaviors and transformation action.

<table>
<thead>
<tr>
<th>Case</th>
<th>Sub-Case</th>
<th>Detection Conditions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Upward Ramp</td>
<td>DC &gt; CC &amp; PTRC &gt; CC &amp; DC &gt; PTRC &amp; URC ≤ PTRC</td>
<td>TRC = PTRC</td>
</tr>
<tr>
<td></td>
<td>Downward Ramp</td>
<td>DC &lt; CC &amp; PTRC &lt; CC &amp; DC &lt; PTRC &amp; URC ≥ PTRC</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2.5.1.3: Case 3 – Cut

Case 3 occurs when the URC is still ramping towards the TRC and the DC but the DC has suddenly become closer to the CC then the TRC. If the TRC were to be held constant in this case, the algorithm would report a TRC that is larger than the DC and is therefore inaccurate. Thus, the TRC must be instantaneously decreased so that it is never further from the CC than the DC. Because the decrease in distance between the DC and CC happens instantly, a new TRC cannot be calculated that “rides” the peak of the sawtooth waves unless the TRC resets to the CC along with the URC. This would allow a small portion of the sawtooth pattern to slip past the FilterSawtooth() algorithm and propagate into the DriveGrip stimulus. To avoid this error, the initial ramp to the new DC is ignored and the TRC is set equal to the DC. Although the TRC would “ride” above the peaks of the sawtooth wave if this case is immediately followed by Case 2, the equality with the DC would still provide an accurate reference to the driver.
Figure 4-12 exemplifies this behavior in more detail for cuts on both sides of the CC. The complete detection conditions and the resulting transformation action of this case are recorded in Table 4-5. A high level demonstration of the TRC calculations in this case can be seen in Figure 4-9.

Figure 4-12: Detailed example of Case 3 behaviors and transformation action.

Table 4-5: Case 3 detection conditions and transformation action.

<table>
<thead>
<tr>
<th>Case</th>
<th>Sub-Case</th>
<th>Detection Conditions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Cut Down</td>
<td>DC &gt; CC &amp; PTRC &gt; CC &amp; DC ≤ PTRC</td>
<td>TRC = DC</td>
</tr>
<tr>
<td></td>
<td>Cut Up</td>
<td>DC &lt; CC &amp; PTRC &lt; CC &amp; DC ≥ PTRC</td>
<td></td>
</tr>
</tbody>
</table>

4.2.5.1.4: Case 4 – Switch

Case 4 simply describes when the DC suddenly switches to the other side of the CC. When this occurs, the URC resets to the CC and starts to ramp in the switched direction of the DC. In response to this change, the TRC must be immediately reset with the URC so that the TRC may follow Case 1 along the switched ramp direction. Therefore, the TRC is set equal to the URC when this case occurs. This behavior is exemplified in Figure 4-13 for switches in either direction around the CC. The complete detection conditions and the resulting transformation action of this case are recorded in Table 4-6. A high level demonstration of the TRC calculations in this case can be seen in Figure 4-9.
Now that the transformation cases have been discussed in detail, a complete description of the FilterSawtooth() algorithm can be presented. The pseudocode that defines this algorithm is provided in Algorithm 4-6. The algorithm begins by reading the current values for the calculation. The current curvature is found using the ConvertSteeringToCurvature() function, which calculates the current curvature using (4.1) with the CurrentVelocity and CurrentSteeringAngle public variables. These public variables are updated at 25Hz by a separate message handling process that will be described in Section 4.3. The current curvature is then stored in the CC local variable. The desired curvature is specified by the current motion that is being iterated through by the CalculateDriveGripStimuli() algorithm. The desired curvature is then stored in the DC local variable. The current time is found using the ReadCurrentTime() function and stored in the CurrentTime variable. Next, the algorithm recalls past values from the previous time that the algorithm was called. The PTRC is set as the previous TRC value calculated by FilterSawtooth(). The past untransformed reference curvature (PURC) is set as the previous URC evaluated by FilterSawtooth(). The PastTime variable is set as the CurrentTime value from the previous iteration as well.
Algorithm 4-6: FilterSawtooth() Pseudocode

Function FilterSawTooth(Motion) Returns ReferenceSteeringAngle

% Read current values.
CC = ConvertSteeringToCurvature(CurrentSteeringAngle, CurrentVelocity) % Current Curvature = CC
DC = Motion.DesiredCurvature % Desired Curvature = DC
CurrentTime = ReadCurrentTime()
% Recall past values.
PTRC = TRC_{i-1} % Past Transformed Reference Curvature = PTRC
% Transformed Reference Curvature = TRC
PURC = URC_{i-1} % Past Untransformed Reference Curvature = PURC
% Untransformed Reference Curvature = URC
PastTime = CurrentTime_{i-1}
% Update URC
Delta = Motion.CurvatureRateofChange * (CurrentTime – PastTime)
If CC <= PURC < DC Then
    URC = PURC + Delta % DC > CC, ramp up.
Elseif CC >= PURC > DC Then
    URC = PURC – Delta % DC < CC, ramp down.
Else
    URC = PURC % URC = DC, hold constant.
End If
% Determine case and make corresponding transformation.
If DC > CC and PTRC > CC and DC > PTRC and URC > PTRC Then
    % Case 1: Upward Ramp
    TRC = URC
Elseif DC < CC and PTRC < CC and DC < PTRC and URC < PTRC Then
    % Case 1: Downward Ramp
    TRC = URC
Elseif DC > CC and PTRC > CC and DC > PTRC and URC ≤ PTRC Then
    % Case 2: Upward Ramp
    TRC = PTRC
Elseif DC < CC and PTRC < CC and DC < PTRC and URC ≥ PTRC Then
    % Case 2: Downward Ramp
    TRC = PTRC
Elseif DC > CC and PTRC < CC and DC ≤ PTRC Then
    % Case 3: Cut Down
    TRC = DC
Elseif DC < CC and PTRC < CC and DC ≥ PTRC Then
    % Case 3: Cut Up
    TRC = DC
Elseif DC < CC and PTRC ≥ CC Then
    % Case 4: Switch Down
    TRC = URC
Elseif DC > CC and PTRC ≤ CC Then
    % Case 4: Switch Up
    TRC = URC
End If
% Convert reference curvature to reference steering wheel angle
ReferenceSteeringAngle = ConvertCurvatureToSteering(TRC, CurrentVelocity)
Return ReferenceSteeringAngle
End Function
The next step that the algorithm takes updates the URC by evaluating the motion profile at the current time. The change in URC is calculated by reading the curvature rate of change from the current motion and multiplying it by the difference between the current and past iteration times. Since the rate of change specified by the motion does not have a sign, the change has no direction and is stored in the Delta local variable. The URC is then found by incrementing the PURC by Delta in the direction away from the CC and towards the DC. If the PURC is equal to the DC, then the ramping ends and the URC is set to the PURC to hold the constant DC for that particular motion.

At this point, the algorithm can detect the behavioral case of the URC sawtooth pattern and take the necessary transformation action. An eight-condition if/else if block is used to detect which of the previously discussed four cases is occurring. Once the case is identified, the TRC is set accordingly and completely removes the sawtooth pattern from the reference signal. The TRC is then converted to an instantaneous reference steering wheel angle through the ConvertCurvatureToSteering() function. This function simply uses (4.2) and the current vehicle velocity specified by the CurrentVelocity public variable to convert curvatures to steering wheel angles. Finally, the instantaneous reference steering wheel angle is returned to the CalculateSteeringError() algorithm that originally called FilterSawtooth().

4.2.6: CheckRampLead()

The purpose of the CheckRampLead() function is to determine if the ramp of the driver's response is leading the ramp of the reference steering wheel angle in the same direction. This function is always called after it has already been determined that the reference is ramping and only checks if the response is leading or lagging the reference. Pseudocode has been provided in Algorithm 4-7 to fully define the functionality of this algorithm.

The first step of the algorithm is to determine which direction the reference is ramping in. If the sign of the RampDelta parameter passed in from CalculateSteeringError() is positive, the reference signal must be ramping upwards. Otherwise, the finite derivative of the reference is negative and thus the reference signal would be ramping downwards. In either case, the current steering wheel angle is compared to the reference steering wheel angle to determine whether or not the driver is leading the reference signal. If the reference is ramping upwards and the current steering wheel angle is greater than the reference steering wheel angle, the driver is in fact leading the reference ramp signal. The driver is also leading the reference signal if it is ramping downwards and the current steering wheel angle is less than the reference steering wheel angle. Otherwise, the driver is determined to be lagging the reference signal. Once the result is found the CheckRampLead() function ends and returns the result to the CalculateSteeringError() caller.
Algorithm 4-7: CheckRampLead() Pseudocode

```plaintext
Function CheckRampLead(RampDelta) Returns LeadingRamp
    If RampDelta ≥ 0 Then
        % Ramp is increasing.
        If CurrentSteeringAngle > ReferenceSteeringAngle Then
            % Response is leading reference signal ramp.
            Return: True
        Else
            % Response is not leading reference signal ramp.
            Return: False
        End If
    Else
        % Ramp is decreasing.
        If CurrentSteeringAngle < ReferenceSteeringAngle Then
            % Response is leading reference signal ramp.
            Return: True
        Else
            % Response is not leading reference signal ramp.
            Return: False
        End If
    End If
End Function
```

4.2.7: ConvertToDriveGripStimulus()

The purpose of the ConvertToDriveGripStimulus() function is to select which vibrotactile elements of the DriveGrip interface will be activated based on the instantaneous steering wheel angle error. This algorithm simply inputs the instantaneous steering error from the calling algorithm and uses a series of case selections to apply Table 3-1 and determine the corresponding stimulus. Additional configuration parameters for the vibration configuration and error thresholds are included that specify which of the DriveGrip operation configurations from Table 3-2 are to be used. Once the corresponding stimulus is determined, the function ends and returns the stimulus to the calling algorithm. The complete pseudocode definition of this algorithm is presented in Algorithm 4-8.
Algorithm 4-8: ConvertToDriveGripStimulus() Pseudocode

Function ConvertToDriveGripStimulus(InstantaneousSteeringError) Returns DriveGripStimulus

\[ \theta_E = \text{InstantaneousSteeringError} \]

Select Case VibrationConfiguration

Case “Four Finger”

Select Case \( \theta_E \)

Case \(-\infty < \theta_E \leq -T_L\)

Return: Left Little

Case \(-T_L < \theta_E \leq -T_R\)

Return: Left Ring

Case \(-T_R < \theta_E \leq -T_M\)

Return: Left Middle

Case \(-T_M < \theta_E \leq -T_I\)

Return: Left Index

Case \(-T_I < \theta_E \leq T_I\)

Return: No Vibration

Case \(T_I \leq \theta_E < T_M\)

Return: Right Index

Case \(T_M \leq \theta_E < T_R\)

Return: Right Middle

Case \(T_R \leq \theta_E \leq +\infty\)

Return: Right Little

End Select

Case “Two Finger”

Select Case \( \theta_E \)

Case \(-\infty < \theta_E \leq -T_L\)

Return: Left Little

Case \(-T_L < \theta_E \leq -T_I\)

Return: Left Index

Case \(-T_I < \theta_E \leq T_I\)

Return: No Vibration

Case \(T_I \leq \theta_E < T_L\)

Return: Right Index

Case \(T_L \leq \theta_E \leq +\infty\)

Return: Right Little

End Select

End Function
4.2.8: CalculateSpeedStripStimuli()

The CalculateSpeedStripStimuli() algorithm continuously calculates the instantaneous SpeedStrip stimuli for the most current motion profile. The calculations of this algorithm are significantly different when compared to the calculations of DriveGrip stimuli because speed control is considered secondary to heading control (Section 3.2.1). It was demonstrated in Section 3.2.4 that the drivers exhibit significant amounts of pure time delay in speed tracking responses because the non-visual interface system is designed and the drivers are trained based on this consideration. This algorithm solves the time delay mitigation functional requirement described in Section 4.1.3 by utilizing futuristic reference speed data to offset the pure time delay of the driver. A complete definition of this algorithm is presented in Algorithm 4-9 and discussed in the following paragraphs.

Algorithm 4-9: CalculateSpeedStripStimuli() Pseudocode

```plaintext
Subroutine CalculateSpeedStripStimuli()
    MotionProfileStartTime = ReadCurrentTime()
    ReferenceSpeed = CurrentMotionProfile.LastMotion.DesiredSpeed
    While ReadCurrentTime() – MotionProfileStartTime < CurrentMotionProfile.Duration and_ _StopRequested = False and NewMotionProfileFlag = False
        Wait for 25ms interval
        InstantaneousSpeedError = ReferenceSpeed – CurrentSpeed
        LowPassFilter(InstantaneousSpeedError, f_c = 1Hz)
        SpeedStripStimulus = ConvertToSpeedStripStimulus(InstantaneousSpeedError, ReferenceSpeed)
    End While
End Subroutine
```

The algorithm begins by recording the timestamp at which the most current motion profile was received. Storing this timestamp in the MotionProfileStartTime variable allows the algorithm to later determine when the duration of the current motion profile has been completely exhausted. Next, the reference speed is set to the desired speed of the most futuristic motion of the current motion profile and remains constant for the entire duration of the profile. By setting the reference speed to the most futuristic desired speed, the large response pure time delays shown in Section 3.2.4 can be partially or completely offset. This allows the driver to track the reference speed with much less time delay and recreate the planned trajectories of the Motion Planner with higher accuracy. The reference speed is constant for the entire motion profile duration because the most futuristic information has already been utilized. This additionally requires less cognitive effort designated for speed tracking as the reference remains constant for considerable lengths of time.

After the reference speed is statically set in the ReferenceSpeed variable, a timed while loop starts and continuously calculates the instantaneous SpeedStrip stimulus at a rate of 40Hz (25ms period). The first step of one loop iteration is to calculate the
instantaneous speed error using (3.9) by finding the difference between the static ReferenceSpeed variable and the CurrentSpeed public variable updated by the separate message handling process later described in Section 4.3. Similarly to the steering error, the instantaneous speed error is passed through a low pass filter to remove any discontinuities that will propagate into the SpeedStrip stimuli and possibly confuse the driver. A cutoff frequency of 1hz was empirically chosen to balance the tradeoff between signal smoothness and additional lag injected into the system. Finally, the instantaneous speed error is converted to a stimulus and stored in the global SpeedStripStimulus variable for use by the GenerateNonVisualStimuli() algorithm. The conversion is performed by the ConvertToSpeedStripStimulus() function defined in Section 4.2.9. The timed while loop continues to execute these steps until the time duration of the motion profile has been exhausted, a new motion profile is available, or a software stop has been requested by the user.

### 4.2.9: ConvertToSpeedStripStimulus()

The purpose of the ConvertToSpeedStripStimulus() function is to select which vibrotactile elements of the SpeedStrip interface will be activated based on the instantaneous speed error. This algorithm simply inputs the instantaneous speed error from the calling algorithm and uses a series of case selections to apply Table 3-6 and determine the corresponding stimulus. Additional configuration parameters for the vibration configuration and error thresholds are included that specify which of the SpeedStrip operation configurations from Table 3-7 are to be used. Once the corresponding stimulus is determined, the function ends and returns the stimulus to the calling algorithm. The complete pseudocode definition of this algorithm is presented in Algorithm 4-10.
Algorithm 4-10: ConvertToSpeedStripStimulus() Pseudocode

Function ConvertToSpeedStripStimulus(InstantaneousSpeedError, ReferenceSpeed) Returns SpeedStripStimulus
    \( S_E \) = InstantaneousSpeedError
    \( S_R \) = Reference Speed
    If \( S_R < T_{STOP} \)
        If \( |S_E| > 0 \)
            Return: Shoulder, Upper Back, Middle Back, Lower Back
        End If
    Else
        Select Case VibrationConfiguration
            Case “Four Element”
                Select Case \( S_E \)
                    Case \(-\infty < S_E \leq -T_{B4}\) Return: Shoulder
                    Case \(-T_{B4} < S_E \leq -T_{B3}\) Return: Upper Back
                    Case \(-T_{B3} < S_E \leq -T_{B2}\) Return: Middle Back
                    Case \(-T_{B2} < S_E \leq -T_{B1}\) Return: Lower Back
                    Case \(-T_{B1} < S_E < T_{L1}\) Return: No Vibration
                    Case \(T_{L1} \leq S_E < T_{L2}\) Return: Buttocks
                    Case \(T_{L2} \leq S_E < T_{L3}\) Return: Upper Thigh
                    Case \(T_{L3} \leq S_E < T_{L4}\) Return: Lower Thigh
                    Case \(T_{L4} \leq S_E < +\infty\) Return: Knees
                End Select
            Case “Two Element”
                Select Case \( S_E \)
                    Case \(-\infty < S_E \leq -T_{B4}\) Return: Shoulder
                    Case \(-T_{B4} < S_E \leq -T_{B1}\) Return: Middle Back
                    Case \(-T_{B1} < S_E < T_{L1}\) Return: No Vibration
                    Case \(T_{L1} \leq S_E < T_{L4}\) Return: Upper Thigh
                    Case \(T_{L4} \leq S_E < +\infty\) Return: Knees
                End Select
        End Select
    End Function
4.2.10: GenerateNonVisualStimuli()

The GenerateNonVisualStimuli() subroutine is a simple algorithm that physically generates the stimuli calculated from the parallel CalculateNonVisualStimuli() algorithm. As shown in the CalculateDriveGripStimuli() and CalculateSpeedStripStimuli() algorithms, the instantaneous stimuli for each non-visual interface are recalculated and stored in public variables at 40Hz. This algorithm continuously reads the public stimuli variables and powers the vibro-tactile elements of each interface accordingly. The complete algorithm is defined in Algorithm 4-11 and is discussed in the following text.

**Algorithm 4-11: GenerateNonVisualStimuli() Pseudocode**

```plaintext
Subroutine GenerateNonVisualStimuli()
    While StopRequested = False
        Wait for 25ms interval
        PowerDriveGripElement(DriveGripStimulus)
        PowerSpeedStripElement(SpeedStripStimulus)
    End While
End Subroutine
```

The algorithm immediately begins a timed while loop that updates the power state of each vibro-tactile elements of the non-visual interfaces at a rate of 40Hz (25ms period). Each iteration passes the DriveGripStimulus and SpeedStripStimulus public variables into power subroutines that actuate the corresponding stimuli. These subroutines, called PowerDriveGripElement() and PowerSpeedStripElement(), send signals to the Non-Visual Interface Controller (Section 3.4) via USB that command which vibro-tactile elements of each non-visual interface must be activated. The NVIC then ensures that the selected elements transition to or remain in the powered state while the unselected elements transition to or remain in the unpowered state. These steps repeat continuously until the user requests a software stop and forces the algorithm to end.
4.3: Software Implementation

The PNVID algorithm defined in Section 4.2 was implemented in specialized software called the PNVID Component. The PNVID Component was specifically designed to realize the PNVID algorithm and provide the necessary JAUS interoperability for implementation on the JAUS-based Blind Driver Challenge™ system described in Section 2.4.4.2. This subsection documents the development tools used to write the PNVID Component software, presents the algorithm that defines the software itself, and concludes with a discussion of how the software is executed within the comprehensive Blind Driver Challenge™ system.

4.3.1: Development Tools

The development of the PNVID Component software application relied heavily on the use of two important programming tools. The first tool, called LabVIEW™, is a graphical programming language designed by National Instruments that was used to write the actual software of the PNVID Component. The second tool, called the TORC JAUS Toolkit™ (JTK), is an add-on module to the LabVIEW Development Environment and dramatically decreases the amount of development required to create a JAUS interoperable software application. The following subsections provide a more detailed description of these development tools and outline the advantages each provided in the implementation of the PNVID Component.

4.3.1.1: The LabVIEW™ Graphical Programming Language

The PNVID algorithm presented in this section was realized in software using the National Instruments LabVIEW™ graphical programming language. The LabVIEW™ programming language provides a method for software developers to write code in a graphical format [119]. The language enables software to be written as a block diagram using a series of blocks, called Virtual Instruments (VIs), and connection wires. Each VI represents a single function or subroutine that can be called by other VIs within the program. The VIs can require any amount of inputs and produce any amount of outputs. The inputs and outputs of different VIs can share data through wires connecting the input and output terminals of each VI. An example of LabVIEW™ graphical programming with VIs and wiring is shown in Figure 4-14.

Aside from the graphical coding interface, software development with LabVIEW™ is similar to other sequential based programming languages such as C. The major advantages of using LabVIEW™ to implement to PNVID algorithm are:

- Effortless parallel process development [120].
- Seamless integration with thousands of different hardware devices [121].
- Development environment includes an extensive collection of built-in functions for data acquisition, signal processing, math, and hardware integration [122].

- Additional development modules are available that are specialized in many different engineering fields, such as Controls and Embedded Systems [122].

- G-code used to develop VIs is already a compiled language [123].

These advantages remove much of the development issues that arise with the use of sequential programming languages such as C. The LabVIEW™ programming language is specifically designed to be relatively high-level, handling most of the low-level functions with an extensive amount of built-in VIs. This allows the developers to focus more on the algorithm development rather than code implementation. However, the low level functions built into the development environment are always accessible to the developer if low-level code implementation is required for more complex algorithms.

![LabVIEW Graphical Programming example: Generate Sine Point.vi definition and Block Diagram](image)

**Figure 4-14:** LabVIEW Graphical Programming example: Generate Sine Point.vi definition and Block Diagram

### 4.3.1.2: The TORC Robotics JAUS Toolkit™ (JTK)

The TORC JAUS Toolkit is an add-on module for the LabVIEW Development System designed to greatly simplify the process of adding JAUS interoperability to software applications [124]. The JTK, in its simplest form, is a large collection of LabVIEW VIs that implement the JAUS Reference Architecture Specification (RAS) V3.3 [107] [110] [111]. The advantage of the JTK is that it provides a series of high-level VIs that handle most of the complex, low-level JAUS functionality internally. These VIs perform functions such as parsing RAS defined messages and even defining experimental messages customized for a particular software application.
Aside from providing a full implementation of the RAS, the JTK additionally provides VIs that internally manage the basic functions of a JAUS component. These VIs start separate, parallel threads that run in the background and handle all of the low-level component requirements, such as dynamic registration, core functions, and transport services. The JTK also includes a complete Node Manager executable that performs all of the necessary requirements of the Node Manager component defined in the RAS. These highly convenient features enable the software developer to quickly and easily create software that interfaces with any other JAUS interoperable system.

4.3.2: Algorithm Definitions
The PNVID Component is constructed of several LabVIEW Virtual Instruments (VIs) that cooperate in both series and parallel to ultimately implement the PNVID algorithm defined in Section 4.2. The purpose of this subsection is to provide a detailed description of these VIs and discuss how the functionalities of each contribute to the overall goal of the PNVID software. Each description will supply pseudocode that defines the VI algorithm in a simplistic and easy to follow manner. A legend of color-codes used in the pseudo-code descriptions has been provided in Algorithm 4-12.

Algorithm 4-12: Pseudocode Legend

<table>
<thead>
<tr>
<th>% Comments</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom.vi</td>
<td>% Identifies custom software VIs that will be explicitly defined.</td>
</tr>
<tr>
<td>TORC_JTK.vi</td>
<td>% Identifies TORC JAUS Toolkit VIs that will be summarized.</td>
</tr>
<tr>
<td>PublicVariables</td>
<td>% Identifies variables public to all software VIs.</td>
</tr>
</tbody>
</table>

The PNVID Component contains several levels of nested sub-VIs. In order to present each and every VI in an intuitive manner, the sub-VIs will be described in the order at which they are first presented. The descriptions will begin with the PNVIDComponent VI and continue through each sub-VI until all have been defined and discussed. Table 4-7 displays the hierarchy of the nested PNVIDComponent structure and identifies the section in which each sub-VI is defined and discussed.

Table 4-7: PNVID Component custom VI hierarchy and section index.
4.3.2.1: PNVIDComponent.vi

The purpose of the PNVIDComponent VI is to realize the PNVID algorithms presented in Section 4.2 and provide full compatibility with the Joint Architecture for Unmanned Systems (JAUS). As described in Section 2.4.4, the Blind Driver Challenge™ platform implements JAUS to create an organized system topology and standardize the communication between TORC Robotics and Virginia Tech software. Since the PNVID algorithm relies on data from TORC Robotics software, the PNVIDComponent VI must therefore possess the ability to communicate with the TORC software through JAUS defined methods and messages. Pseudocode that fully defines these functionalities has been provided in Algorithm 4-13 below.

**Algorithm 4-13: PNVIDComponent.vi Pseudocode**

```
Main PNVIDComponent.vi
    While Reset = True
        InitializeComponent.vi
        StartComponent.vi
        Start Parallel Processes
            ExecutePNVID.vi
            MonitorStopReset.vi
        End Parallel Processes
        CloseServiceConnections.vi
        StopComponent.vi
    End While
    DestroyComponent.vi
End Main
```

Generally speaking, the PNVIDComponent VI utilizes the TORC JTK to create a JAUS component that executes the PNVID algorithm. According to the Reference Architecture Specification in [110], the component must have the ability to be started, stopped, and reset at the discretion of other system components. The while loop that this VI begins with enables the JAUS component to be reset infinitely within the execution of the VI itself. The pseudocode within the while loop represents a single life cycle of the JAUS component and exits when a stop or reset has been requested by the driver. If a reset is requested, the while loop repeats and resets the component. If a stop is requested instead of a reset, the while loop exits and the component is destroyed in memory through the DestroyComponent VI of the JTK.

A single life cycle of the JAUS component begins with the InitializeComponent custom VI defined in Section 4.3.2.2. This VI creates or recreates the JAUS component in memory and sets the necessary configuration parameters. The StartComponent JTK VI is then called to start the background process that dynamically connects to the TORC Node Manager and handles the core subgroup functionalities of the JAUS component defined in [107]. Once the component has been started, the ExecutePNVID and MonitorStopReset custom VIs are started in parallel. The ExecutePNVID VI defined in
Section 4.3.2.3 performs additional JAUS-related initialization procedures and subsequently executes the actual PNVID algorithm. The MonitorStopReset VI defined in Section 4.3.2.7 continuously monitors if a JAUS stop or reset has occurred and sets the Stop and Reset public variables accordingly. Once a stop or reset occurs, both parallel processes exit and the CloseServiceConnection JTK VI is called to close the active service connections of the PNVID component. The component is then stopped and disconnected from the TORC Node Manager with the StopComponent JTK VI. This marks the end of a single life cycle of the PNVID JAUS component.

4.3.2.2: InitializeComponent.vi

The purpose of the InitializeComponent VI is to prepare the JAUS component background process for execution and set the necessary component configuration parameters. The configuration parameters are loaded from a preset configuration file each time the VI is called and passed into a series of JTK VIs that actually perform the initialization procedures for the component. The configuration parameters define the component name, identification type, and ID that are used for dynamic identification by the TORC Node Manager and other software components within the Blind Driver Challenge™ system. The parameters additionally describe the list of incoming and outgoing messages that are supported by the component along with the respective service ID associated with each message. Once the configuration information has been loaded and set in memory, the InitializeComponent VI ends and the StartComponent JTK VI can be called.

4.3.2.3: ExecutePNVID.vi

The purpose of the ExecutePNVID VI is to ensure that the necessary TORC software components are online, secure the necessary message service connections, and finally run the PNVID algorithm defined in Section 4.2. The full definition of this VI is presented with pseudocode in Algorithm 4-14 below. The VI begins with calling the WaitForComponents VI defined in Section 4.3.2.4. This custom VI forces the thread to wait until all the required TORC software components are online. Next, the VI calls the EstablishServiceConnections VI defined in Section 4.3.2.5 to create the necessary message service connections with the TORC software components. Once these first two steps are complete, the SetReadyState JTK VI is called to set the component to the ready state. This identifies to other software components in the system that the PNVID component has completed initialization and is now under normal operation.

The normal operation of the PNVID component consists of three parallel processes. The first process is the MessageHandler custom VI and is defined in Section 4.3.2.6. This VI manages all incoming messages, including service connections, and stores the associated information in the necessary public variables for use by the PNVID algorithm. The PNVID algorithm itself is realized by the second and third processes running in parallel with the MessageHandler VI. The second parallel process is the
CalculateNonVisualStimuli VI, which realizes the CalculateNonVisualStimuli() sub-algorithm of the PNVID algorithm previously defined in Section 4.2.2. The third and final parallel process is the GenerateNonVisualStimuli VI, which realizes the GenerateNonVisualStimuli() sub-algorithm of the PNVID algorithm previously defined in Section 4.2.10. All three of these parallel processes execute the PNVID algorithm continuously until a JAUS Stop or Reset event occurs through the MonitorStopReset VI defined in Section 4.3.2.7.

Algorithm 4-14: ExecutePNVID.vi Pseudocode

Subroutine ExecutePNVID.vi
   WaitForComponents.vi
   EstablishServiceConnections.vi
   % Initialization complete, begin normal operation.
   SetReadyState.vi
   Start Parallel Processes
      MessageHandler.vi
      CalculateNonVisualStimuli.vi
      GenerateNonVisualStimuli.vi
   End Parallel Processes
End Subroutine

4.3.2.4: WaitForComponents.vi
The purpose of the WaitForComponents custom VI is to force the calling VI to wait until all required JAUS components are online in the JAUS system network. The VI is able to perform this function by continuously requesting the subsystem configuration from the NVICPU Node Manager and checking to see if the required service IDs are present. Once all required service IDs are present, the VI returns execution back to the calling VI. The required component services are specified by a custom JAUS component configuration file and only include the services that are vital to the proper operation of the PNVID component. More information on the custom component configuration file will be discussed later in Section 4.3.3.

4.3.2.5: EstablishServiceConnections.vi
The purpose of the EstablishServiceConnections custom VI is to force the calling VI to wait until all required message service connections are established with the other software components. The VI is able to perform this function by continuously requesting the required service connections from each software component until all have been established. Once the service connections have been successfully established, the VI returns execution back to the calling VI. The required service connections are specified by the same custom JAUS component configuration file discussed in the WaitForComponents.vi definition and only include service connections vital to the proper operation of the PNVID component. More information on the custom component configuration file will be discussed later in Section 4.3.3.
4.3.2.6: MessageHandler.vi

The purpose of the MessageHandler VI is to receive and parse incoming JAUS messages and store the resulting values in the corresponding public variables. The VI is notified of incoming messages through the JTK component VIs and only handles messages that are not part of the core message group. Recalling from Section 4.3.1.2, core subgroup messages are internally handled by the JTK component VIs to simplify JAUS implementation for the developer. Thus, the PNVID Component is configured to only receive three types of JAUS messages. The first message, called Set Motion Profile, is sent by the Motion Planner at a rate of 2-8Hz and contains the most up-to-date motion profile. A complete definition of this message can be found in Section 2.3.2.2.3. The second message, called Report Velocity State, is received through a 25Hz service connection with the Velocity State Sensor component of the ByWire XGV™ and reports the current longitudinal velocity of the vehicle. The third message, called Report Wrench Efforts, is received through a 25Hz service connection with the Primitive Driver component of the ByWire XGV™ and reports the current steering wheel angle of the vehicle. Both the Report Velocity State and Report Wrench Efforts messages are fully defined in Section 2.3.2.1.4. The MessageHandler VI receives each of these three messages and stores the data in the public variables utilized by the PNVID algorithm definitions in Section 4.2. The full pseudocode definition of the MessageHandler VI is presented in Algorithm 4-15.

Algorithm 4-15: MessageHandler.vi Pseudocode

Subroutine MessageHandler.vi

While Stop = False and Reset = False

Select Case NewIncomingMessage

Case “Set Motion Profile (xE238)”

CurrentMotionProfile = ReadIncomingMessage()

NewMotionProfileFlag = True

Case “Report Velocity State (x4404)”

CurrentSpeed = ReadIncomingMessage()

Case “Report Wrench Efforts (x4405)”

CurrentSteeringAngle = ReadIncomingMessage()

Case “No Message”

% Do nothing.

End Select

End While

End Subroutine

The VI runs continuously through a while loop and only ceases execution when one of the Stop or Reset public variables has been set to true by the MonitorStopReset VI discussed in Section 4.3.2.7. Inside of the loop, a case selector block is utilized to determine which message, if any, has been received by the PNVID component. The first case occurs when the Set Motion Profile message is received. In this case, the VI reads the new motion profile from the Set Motion Profile message and stores it in the
CurrentMotionProfile public variable for use by the PNVID algorithm. The VI then sets the NewMotionProfileFlag public variable to true, also for use by the PNVID algorithm. The second and third cases occur when the Report Velocity State and Report Wrench Efforts messages are received, respectively. The VI then reads the current speed or current steering wheel angle from these messages and stores them in the CurrentSpeed and CurrentSteeringAngle public variables for use by the PNVID algorithm. Because the PNVID component receives these three messages periodically, the CurrentMotionProfile, CurrentSpeed, and CurrentSteeringAngle public variables are always kept up-to-date with the most current data and can be used by the PNVID algorithm to calculate the non-visual stimuli.

4.3.2.7: MonitorStopReset.vi
The purpose of the MonitorStopReset custom VI is to set the Stop, Reset, and StopOrReset public variables accordingly when the PNVID component is commanded to stop or reset. The stop and reset commands are generated by the JTK component VIs and can come from other software components within the JAUS system or directly from the component’s user interface. When these commands occur, this VI sets the Stop, Reset, and StopOrReset public variables accordingly for use by the PNVID algorithm and other VIs of the software implementation. It is through these public variables that the algorithm and other VIs are notified that a stop or reset has been commanded and that all of the nested threads must immediately perform the necessary closing procedures and stop execution.

4.3.3: Software Execution
As described in Sections 2.4.4.2 and 3.5, the Non-Visual Interface Computer (NVICPU) hosts the PNVID software and acts as a single computing node within the JAUS network of the Blind Driver Challenge™ platform. Because the PNVID software executes on the NVICPU, it is classified as a component within the NVICPU node. The PNVID Component is able to communicate with other software components within the platform’s JAUS network through the network connection and Node Manager of the NVICPU.

Each component within the JAUS network contains special properties that identify the component within the system and describe the services that the component offers. These properties are defined for the PNVID component in Table 4-8. The configuration in this table indicates that the PNVID component identifies itself with a component ID of 15. This particular ID was set to describe the PNVID service in coordination with TORC Robotics so that it may be identified by other software components within the platform JAUS network. The remainder of the PNVID JAUS address is dynamically determined by the NVICPU Node Manager; however the instance address is typically 1 as only one instance of the PNVID is executed at any given time.
The supported incoming messages of the PNVID Component include the Set Motion Profile, Report Velocity State, and Report Wrench Efforts messages. Recalling from Section 4.3.2.6, these messages provide the current motion profile, actual steering wheel angle, and actual speed data that is utilized by the PNVID algorithm. The supported incoming message specification configures the JTK component VIs to receive these messages internally and expose them to the MessageHandler VI of the PNVID software implementation. Additionally, the Motion Planner component uses the supported incoming message information to determine which components within the JAUS network require the Set Motion Profile message. This procedure must be taken as the Set Motion Profile message is a command class message and cannot be requested as a service connection by the PNVID component.

The required services and service connections are utilized by the WaitForComponent and EstablishServiceConnections VIs of the PNVID software implementation to ensure that the necessary message streams are prepared before normal operation begins. The configuration lists the Motion Planner, Velocity State Sensor, and Primitive driver as required services because they provide the Set Motion Profile, Report Velocity State, and Report Wrench Efforts messages that are vital to the operation of the PNVID software. The configuration also defines required service connections to the Report Velocity State and Report Wrench Efforts messages with a rate of 25Hz. These service connections ensure that constant streams of current velocity and steering angle data are provided to the PNVID software component.

---

**Table 4-8: PNVID Component Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>PNVID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification Type</td>
<td>30001</td>
</tr>
<tr>
<td>Component ID</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported Incoming Messages</th>
<th>Service ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Motion Profile</td>
<td>15</td>
<td>xE238</td>
</tr>
<tr>
<td>Report Velocity State</td>
<td>15</td>
<td>x4404</td>
</tr>
<tr>
<td>Report Wrench Efforts</td>
<td>15</td>
<td>x4405</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported Outgoing Messages</th>
<th>Service ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Services</th>
<th>Service ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Planner</td>
<td>22</td>
</tr>
<tr>
<td>Velocity State Sensor</td>
<td>42</td>
</tr>
<tr>
<td>Primitive Driver</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Service Connections</th>
<th>Service ID</th>
<th>Message</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Velocity State</td>
<td>42</td>
<td>x4404</td>
<td>25Hz</td>
</tr>
<tr>
<td>Report Wrench Efforts</td>
<td>33</td>
<td>x4405</td>
<td>25Hz</td>
</tr>
</tbody>
</table>
4.4: Performance Analysis

This subsection presents a series of experiments that investigate the performance of the PNVID and the comprehensive Blind Driver Challenge® system supplemented with the PNVID software. The discussions begin with an examination of the reference steering and speed signals generated by the PNVID. These examinations demonstrate the typical reference signals that result from the PNVID algorithm that is defined in Section 4.2. Next, an investigation of the driver’s ability to track the steering and speed reference signals is conducted to determine if the PNVID successfully generates non-visual stimuli that can be understood and reliably interacted with by the driver. Finally, the navigational abilities of the driver are analyzed to determine the overall performance of the comprehensive Blind Driver Challenge® system supplemented with the PNVID software.

4.4.1: Reference Signal Generation

The reference steering and speed signals are generated by the algorithms defined previously in Section 4.2. These algorithms perform a series of transformation steps to generate reference signals that are both accurate and in a form that can be understood and reliably interacted with by the driver. The following subsections provide typical steering and speed reference signals that result from these algorithms and demonstrate how the transformations solve the functional requirements of the PNVID that are defined in Section 4.1.

4.4.1.1: Reference Steering Wheel Angle Signal

Figure 4-15 depicts a typical reference steering wheel angle signal generated by the PNVID algorithm at a constant speed of 10m/s. The generated signal, called the Transformed Reference Steering (TRS), was derived from the Untransformed Reference Steering (URS) and the Current Steering (CS) using the algorithm defined in Section 4.2.5. Observation of this figure indicates that the PNVID algorithm is successful in removing the sawtooth patterns from the reference steering signal and creates a much smoother reference for the driver to track. The transformation process within the PNVID algorithm enabled the transformed reference signal to “ride” the peaks of the sawtooth pattern caused by the rapid regeneration of the motion profiles.

Although the transformed reference signal is relatively smooth when compared to the untransformed signal, the error calculated from the transformed signal and (3.1) exhibits a significant amount of “noise” due to the subtraction of two discontinuous signals. Close observation of the TRS and CS signals in Figure 4-15 shows that the two signals are discontinuous and are comprised of small, step-type increments. The subtraction of these two signals in (3.1) propagates discontinuity into the instantaneous error signal, as demonstrated in Figure 4-16. The ratio of the discontinuities caused by the step-type incrementing to the amplitude of the error signal is large and causes the discontinuities...
to straddle error thresholds that define which non-visual stimuli should be generated. This causes rapid changes in the stimuli communicated to the driver and significantly hinders his or her ability to track the reference steering signal.

Figure 4-15: Example of the transformed reference steering (TRS) signal generated by the PNVID algorithm at a constant speed of 10m/s. The TRS is derived from the untransformed reference steering (URS) signal and the current steering (CS) signal through the algorithms defined in Section 4.2.5.

Figure 4-16: Instantaneous steering wheel angle error calculated from the data shown in Figure 4-15 using (3.1) and the FFLT DOC defined in Table 3-2.

Recalling from Section 4.2, the definition of the PNVID algorithm includes a filtering step to decrease the discontinuity-to-signal ratio and rectify the instantaneous error signal. The low-pass filter applied in this step eradicates the discontinuities and prevents rapid changes in non-visual stimuli that may confuse the driver. Examination of Figure 4-16
indicates that the filtered error signal exhibits much smoother and lower frequency transitions between error ranges and ultimately non-visual stimuli. However, the rectification of the smooth instantaneous error signal comes at the cost of additional pure time delay injected into the signal itself. While the low-pass filter removes the discontinuities of the signal, it also creates a time delay in the form of a phase shift due to the frequency response of the low-pass filter used. This delay can be quantified by performing a time-shifted mean absolute error calculation. In this calculation, the filtered error is shifted in time across the unfiltered error and the mean absolute error (MAE) is calculated at each time shift. The shift at which the MAE is minimized indicates the delay induced by the second order Butterworth low-pass filter with 1Hz cutoff frequency utilized by the PNVID. This calculation has been performed on the error data shown in Figure 4-17. The results are plotted in Figure 4-17 and indicate that the low-pass filter causes 225ms of pure time delay between the unfiltered and filtered error signal.

![Figure 4-17: Time-shifted mean absolute error (MAE) comparison for determining the phase shift between the unfiltered and filtered instantaneous steering wheel angle error signal shown in Figure 4-16.](image)

When coupled with the pure time delay of the driver, the phase shift induced by the low-pass filter can cause overall driver reaction delays that effect the accuracy and stability of the closed loop system. Section 3.1.4 indicates that approximately 460ms of pure time delay can be expected from the driver’s response when tracking an arbitrary reference signal. When superimposed on the delay caused by the low-pass filter, the overall delay between the calculation of a reference and the driver’s associated reaction can be expected to be in the neighborhood of 685ms. Fortunately, this delay is offset by simply adjusting a parameter within the TORC AutonoNav™ Motion Planner. The planning software enables a “planning horizon” to be set that calculates future motion profiles and reports them as current motion profiles. The planner effectively uses
preview information from the environmental perception sensors to provide future data to
the driver in an attempt to offset the driver’s time delay. By setting the planning horizon
equal to the total delay of the system, the motion planning software is able to offset the
total delay as long as sufficient preview of the navigation environment is available.

4.4.1.2: Reference Speed Signal
Figure 4-18 depicts a typical speed reference signal generated by the PNVID. In this
experiment, the driver navigated a straightaway that proceeded into a slight turn with an
obstacle completely blocking the road. The reference signal generated by PNVID
simply exposes the most futuristic desired speed of the current TORC AutonoNav™
motion profile, which typically remains at a constant value designated by the speed limit
and curvature of the particular road section being navigated. However, small
discontinuities tend to appear in the reference signal as the AutonoNav™ system
makes slight updates in the current planned trajectory. These discontinuities propagate
into the instantaneous speed error through (3.9) and can cause rapid transitions
between SpeedStrip stimuli that may confuse the driver and hinder his or her ability to
track the reference speed.

![Figure 4-18: Typical reference speed signal generated by the PNVID and the
associated driver response utilizing the TE SOC defined in Table 3-7.](image)

Recalling from Section 4.2, the definition of the PNVID algorithm includes a filtering step
to decrease the discontinuity-to-signal ratio and rectify the instantaneous error signal.
The low-pass filter applied in this step eradicates the discontinuities and prevents rapid
changes in non-visual stimuli that may confuse the driver. Examination of Figure 4-19
indicates that the filtered error signal exhibits much smoother and lower frequency
transitions between error ranges and ultimately non-visual stimuli. However, the
rectification of the smooth instantaneous error signal comes at the cost of additional
pure time delay injected into the signal itself. While the low-pass filter removes the
discontinuities of the signal, it also creates a time delay in the form of a phase shift due to the frequency response of the low-pass filter used. Because the same exact low-pass filter was used to rectify the instantaneous steering wheel angle signal as the instantaneous speed signal, the same time delay can be expected. The time-shift MAE analysis performed in Section 4.4.1.1 indicates that this particular low-pass filter induces 225ms of pure time delay between the unfiltered and filtered speed error signal.

![Figure 4-19: Instantaneous speed error calculated from the data shown in Figure 4-18 using (3.1) and the TE SOC defined in Table 3-7.](image)

When coupled with the pure time delay of the driver, the phase shift induced by the low-pass filter can cause overall driver reaction delays that effect the accuracy and stability of the closed loop system. Section 3.2.4.3 indicates that approximately 2000ms of pure time delay can be expected from the driver’s response when tracking an arbitrary reference signal. When superimposed on the delay caused by the low-pass filter, the overall delay between the calculation of a reference and the driver’s associated reaction can be expected to be in the neighborhood of 2225ms. This overall delay, however, is mitigated with the combination of the “planning horizon” parameter of the TORC AutonoNav™ Motion Planner and the use of futuristic motion profile data do designate the reference speed. Recalling from Section 4.4.1.1, the “planning horizon” parameter is set to offset the ~700ms delay expected from the steering response of the driver. Because the entire motion profile is offset by this parameter, the desired speed signals are also offset by the same ~700ms and decreases the effective time delay of the driver speed response to ~1525ms. The remaining delay is mitigated by using the most futuristic desired speed of the motion profile as the instantaneous reference speed. As described in Section 2.3.2.2.3, the motion profiles typically contain 2-3 seconds of future steering and speed actions. This horizon ensures that the remaining effective time delay of the driver’s speed response is compensated for by offsetting the delay with future information.
4.4.2: Reference Signal Tracking

According to the functional requirements defined in Section 4.1, the ultimate purpose of the PNVID is to generate non-visual stimuli that can be understood and reliably interacted with by the driver. In simpler terms, the driver must be able to track each reference signal with a specified amount of accuracy. The following subsections examine the abilities of the driver to track the steering and speed reference signals that are calculated by the PNVID. These analyses aim to demonstrate that the PNVID does in fact generate non-visual stimuli that enable the driver to recreate the desired navigation actions defined by the TORC AutonoNav™ motion planner.

4.4.2.1: Steering Wheel Angle Tracking

A typical driver response to an arbitrary reference signal generated by the PNVID is shown in Figure 4-20. In this experiment, the driver utilized the FFLT DriveGrip operation configuration defined in Table 3-2 to navigate a sweeping left turn with varying curvature at a constant speed of 10m/s. Comparison of the driver response and the reference signal suggests that the driver is able to track the reference without any significant amount of steering wheel angle error. However, close examination of the response and reference signals indicates that the two signals are actually simultaneously interacting with each other. This occurs because the comprehensive Blind Driver Challenge® system implements closed-loop steering control nested within a closed-loop trajectory controller. While the driver closes the loop on the steering wheel angle, the TORC AutonoNav™ system closes the loop on the trajectory of the vehicle. Errors in steering wheel angle tracking by the driver are compensated for by the re-planning capabilities of the AutonoNav™ system and modify the reference steering wheel angle accordingly. The motion profiles generated by the AutonoNav™ system are superimposed on the current steering wheel angle, thus making the reference steering wheel angle signal completely dependent on the driver's response.

Figure 4-20: Typical driver response to a reference steering wheel angle signal generated by the PNVID.
The interaction of the reference signal with the driver response, when combined with the transformations performed by the PNVID, causes the reference signal to generally match the response during response ramps. The ramping of the response continuously forces Case 1 of the sawtooth filtering algorithm (Section 4.2.5) until the response ramp ends or the desired steering wheel angle is overshot, causing a Case 4 switch followed by Case 2. Thus, the driver is actually only tracking step signals superimposed on the current steering wheel angle. This is advantageous as it was proven in Section 3.1.4 that the driver exhibits superior tracking performance when the reference signal is composed of a step signal as opposed to a ramp signal. This provides the additional advantage of inadvertently mitigating ramp leading issues as long as the response generally follows the reference.

The instantaneous steering wheel angle shown in Figure 4-20 is re-plotted with more detail in Figure 4-21 below. This figure demonstrates that the driver is able to track the reference signal with relative ease as the error hardly surpasses the lowest error range designated by the FFLT DOC. Over the thirty second experiment, the driver is able to keep the instantaneous error within the acceptable error range of the FFLT DOC for approximately 73% of the time, making only 15 corrections over the entire timeframe. The mean absolute error (MAE) exhibited by the driver is 4.0° and agrees with the MAEs found in the DriveGrip Performance Analysis (Section 3.1.4) for the FFLT DOC. These quantitative results prove that the reference steering wheel angle signals generated by the PNVID can be tracked by the driver within the degree of accuracy required by the FFLT DOC and do not require significant controller effort by the driver.

*Figure 4-21: Instantaneous steering wheel angle error derived from the reference steering wheel angle signal and driver response signal shown in Figure 4-20.*
4.4.2.2: Speed Tracking

Examination of the driver’s response to the reference speed signal shown previously in Figure 4-18 strongly suggests that the driver is able to track the reference speed with accuracy and ease. The reference steering signal generated by the PNVID can be generalized as a series of step-type increments and decrements in reference speed over time. As discussed in Section 3.2.4.1, the driver is able to track step-type reference signals with steady state errors averaging at 0.28m/s and settling times averaging at 8586ms. These results are further demonstrated by the TE SOC driver response shown previously in Figure 4-18. This particular response yields steady state errors of approximately 0.525m/s and settling times in the neighborhood of 8475ms. While the average settling times are comparable, the average steady state errors differ slightly because the TE SOC only enforces steady-state errors within ±1m/s.

The instantaneous speed error signal shown in Figure 4-19 indicates that the driver is able to track the reference signal with relative ease as the error hardly surpasses the lowest error range designated by the TE SOC. Over the fifty second long experiment, the driver is able to keep the instantaneous error within the acceptable error range of the TE for approximately 77% of the time, making only 2 periods of correction over the entire timeframe. The driver is also able to recognize the full stop request generated by the TORC AutonoNav™ system at the 45 second mark and is able to bring the vehicle to a full stop from 6.3m/s (14mph) within 2.325s. These quantitative results prove that the reference speed signals generated by the PNVID can be tracked by the driver within the degree of accuracy required by the TE SOC and do not require significant controller effort by the driver.

4.4.3: Comprehensive System Performance

At this point, it has been determined that the PNVID is able to supply the driver with reference steering and speed signals that can be tracked within a certain degree of accuracy. However, the tracking performance does not completely describe the abilities of the comprehensive Blind Driver Challenge® system. This subsection aims to characterize the performance of the BDC system as a whole by investigating the driver’s ability to navigate a complex road course under complete blindness. The results from this investigation reflect not only on the abilities of the PNVID, but also on the perception and motion planning abilities of the Blind Driver Challenge® research platform.
4.4.3.1: **Experimental Setup**

In this experiment, fully blind drivers were asked to navigate a complex road course utilizing the complete Blind Driver Challenge® system with PNVID software. The purpose of the experiment was to record the drivers' trajectories along the road course and analyze their lane keeping, speed tracking, and obstacle avoidance abilities using the non-visual stimuli provided by the BDC system. While the drivers were navigating the road course, the trajectory of the vehicle was recorded through the logging of global position points from the Inertial Navigation System described in Section 2.2.1. The reference steering and speed signals as well as the drivers' associated responses were also recorded to match with the vehicle trajectories for a more detailed analysis.

The road course utilized in this experiment was the Patriot Course at the Virginia International Raceway (VIR) near Danville, Virginia. The Patriot Course, pictured in Figure 4-22, is a 1.10mi long paved course composed of various straightaways and curvatures. This road course was chosen for the experiment because its diverse composition of roadways enabled a wide array of driving scenarios to be tested. While the actual width of the course is approximately 30ft, the RNDF (Section 2.6.1) created for the Patriot Course superimposes virtual 18ft lanes along the centerline of the track.

![Road Course Utilized](image)

**Figure 4-22:** Road course utilized for comprehensive Blind Driver Challenge® system testing. Course driving direction is clockwise.
Figure 4-22 additionally describes the speed limits of each course section and identifies the locations of obstacle fields and stop points. The course begins in a 15mph zone that snakes into a long straightaway where the vehicle can safely reach speeds of up to 25mph. A series of sharp turns follow the straightaway with a decreased speed limit of 10mph. Next, the course enters the first obstacle field in which four large boxes are dynamically placed in front of the vehicle by a separate lead vehicle. After passing through the obstacle field, the driver must navigate through an off-centered gate with a 12ft opening. Once through the gate, the driver enters a tight radius snake that must be navigated at slower speeds of 5mph. In the final stretch, the driver navigates a 15mph sweeping turn into the final obstacle field. In this field, a static 100-ft, off-center slalom course is created using four traffic barrels. Finally, the course ends with a stop point at the finish line at which the driver must accurately bring the vehicle to a complete stop.

The navigation experiment was conducted with several fully blind drivers over the course of several days as part of a selection process for which person would serve as the blind driver for the Rolex 24 public demonstration discussed in Section 4.5. The selection process began with 5 different blind candidates from the National Federation of the Blind and a decision was made by a committee based on both skill and the ability to serve as a leader for the Blind Driver Challenge® effort. The results that are presented in this section are taken from the course navigations of the selected driver after the extensive selection process. This particular set of results has been chosen because the selected driver best demonstrates the capabilities of the comprehensive Blind Driver Challenge® system. The data collected and presented in these results cooperates with the Virginia Tech Institutional Review Board (IRB) Human Subject Research Protocol.

4.4.3.2: Results

The results obtained from the comprehensive system performance experiment will be systematically presented and discussed in this subsection. First, a high level analysis of the overall performance will be conducted for the entire length of the track. Following this discussion will be a series of further analyses that break down the track into separate sections to perform a more thorough investigation of the driver’s navigation performance. Each analysis will examine not only the vehicle trajectory in relation to the course, but will also examine lateral lane and speed deviations with correspondences in the steering and speed tracking responses.

Before these discussions begin, it is important to understand how the lateral lane and speed deviations are calculated. The speed deviation calculation is the simplest in that it is just the difference between the actual vehicle speed and the speed limit set for the course section as defined in Figure 4-22. The lateral lane deviation, however, is calculated in a more complex manner. Ultimately, the lateral deviation is found by finding the distance between the center of the vehicle’s rear axle and the center of the
driving lane. This can be calculated by finding the distance between the center of the rear axle reported by the Inertial Navigation System (Section 2.2.1) and the closest driving lane center point. However, the points that define the center of the driving lane are sparse relative to the trajectory points of the rear axle. Therefore, vector projection is used to find the perpendicular distance of the rear axle to a line that connects the nearest two driving lane center points. This is exemplified in Figure 4-23 and defined by the following equations:

\[
P = \begin{bmatrix} - (y_2 - y_1) \\ x_2 - x_1 \end{bmatrix}
\]  \hspace{1cm} (4.5)

\[
R = \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix}
\]  \hspace{1cm} (4.6)

\[
D = \frac{R \cdot P}{|P|}
\]  \hspace{1cm} (4.7)

where \( x_1, y_1 \) and \( x_2, y_2 \) are the nearest lane center points before and after the current vehicle location along its forward trajectory, respectively, \( x_0 \) and \( y_0 \) is the current vehicle location, \( P \) is the vector perpendicular to the vector from the nearest “before” lane center point to the nearest “after” lane center point, \( R \) is the vector from the current vehicle location to the nearest “before” lane center point, and \( D \) is the perpendicular distance between the center of the vehicle rear axle and the center of the driving lane. \( P \) is defined so that positive distances \( D \) indicate deviations to the right of the lane center, as shown in Figure 4-23.

![Figure 4-23: Example of a lateral lane deviation calculation using (4.5)-(4.7).](image-url)
It is important to note that, although equations (4.5)-(4.7) enable the lateral lane deviation to be calculated with sparse lane center points, the sparseness still causes minor discontinuities to appear in the lateral deviation signal over time. These discontinuities occur whenever the vehicle passes the closest “after” lane center point and a new set of “before” and “after” center points are found.

The results presented in this subsection will include a plot that portrays both the lateral lane and speed deviations over time through each course section. These plots additionally show the maximum allowable lateral deviation for the particular course section that the driver is navigating in. The maximum allowable values, shown with a dashed line, are determined by the width of the driving lane and the width of the vehicle itself. These values show the maximum lateral deviation that may occur before the wheels of the vehicle travel outside of the driving lane.

Now that the deviation calculations used in this subsection’s analyses has been defined, the presentation and discussion of the experiment results may begin. Figure 4-24 depicts the trajectory navigated by the driver over the entire length of the VIR Patriot Course. A quick observation of this figure indicates that the driver is generally able to navigate the full course without any significant lane deviations. The lateral lane deviations are examined in much more detail through Figure 4-25. This figure proves that the driver never exceeds the maximum allowable lateral deviation and stays completely within the driving lane over the entire length of the course. The driver comes as close as 1ft from laterally exiting the lane in two different course locations, each of which will be described in more detail in later subsections. The mean absolute lateral deviation (MALD) over the entire course is only 1.76ft, proving that the driver is able to utilize the comprehensive Blind Driver Challenge® system to navigate a road with satisfactory accuracy.

The minimal lateral deviations can be partially attributed to the driver’s ability to accurately track the reference steering wheel angle signal, as shown in Figure 4-26. Over the length of the course, the motions planned by the TORC AutonoNav™ system required many complex steering maneuvers to not only stay within the driving lane, but also avoid static and dynamic obstacles. Keeping in mind the time scale, Figure 4-26 shows that the PNVID successfully transformed the complex steering maneuvers into reference steering signals that the driver could interact with. Figure 4-27 indicates that the driver was able to track the reference signal with relatively small error throughout the entire experiment using the FFLT DOC defined in Table 3-2 and chosen in Section 3.1.4.3. The maximum error exhibited by the driver was approximately 50°, while the mean absolute error was only 5.11°. The combination of the planning abilities of the TORC AutonoNav™ system and the low MAE of the driver’s steering wheel angle response is what enabled the driver to navigate the entire course with only 1.76ft of average lateral lane deviation.
Figure 4-24: Trajectory navigated by a blind driver on the VIR Patriot Course characterized in Figure 4-22. The vehicle trajectory (blue) starts at 0,0 and makes a full clockwise loop. Lane edges (black) and obstacles (red) are also shown.

Figure 4-25: Lateral lane and speed deviations over the length of the entire VIR Patriot course.
Figure 4-26: Reference steering wheel angle generated by the PNVID and the associated blind driver response over the entire VIR Patriot course.

Figure 4-27: Instantaneous steering wheel angle error exhibited by the blind driver over the entire VIR Patriot course.

The speed deviation, also shown in Figure 4-25, indicates that the driver is generally able to track the speed limits of each course section with maximum deviations of 15mph. It is important to recognize that the large deviations occur during transitions between course sections with different speed limits, as will be discussed in later subsections. Even with the transition deviations, the mean absolute speed deviation (MASD) is 1.79mph. This low speed deviation can be attributed to the driver’s ability to track the reference speed signals generated by the PNVID in coordination with the desired speeds specified by the TORC AutonoNav™ system. Figure 4-28 portrays the reference speed signal generated by the PNVID and how well the driver is able to track it using the TE SOC defined in Table 3-7. It is very easy to see the different speed
limits of each course section throughout the reference speed signal and how the driver is able to transition to each new speed limit and continuously track it. The tracking ability is further proven by the plot of speed error over time in Figure 4-29. This plot indicates that the driver exhibited a maximum speed error of only 13mph and a mean absolute error of only 1.38mph. The MAE of the speed response differs from the MASD because the MASD relates the deviations to the section speed limit while the MAE relates the error to the reference speed, which is not always equal to the speed limit.

Figure 4-28: Reference speed generated by the PNVID and the associated blind driver response over the entire VIR Patriot course.

Figure 4-29: Instantaneous speed error exhibited by the blind driver over the entire VIR Patriot course.
These results prove that the blind driver is able to navigate the complex VIR Patriot road course with acceptable levels of lateral lane and speed deviation. However, the results generalize over the entire course and do not capture particular periods of interest that would help provide a more in-depth analysis of the driver’s performance. Therefore, the course will be broken down into sections that can be investigated with more detail. The results of each course section are presented and discussed in the following sub-sections.

4.4.3.2.1: Low-Curvature Snake

The low-curvature snake section, shown in Figure 4-30, includes the starting line of the course as well as a four-turn, low curvature snake. The maximum curvature of the snake is 0.023rad, which is the equivalent of a 140ft turning radius. As indicated by Figure 4-22, the speed limit for this particular course section is set at 15mph. The trajectory navigated by the blind driver can be seen in the section map shown in Figure 4-30. A quick observation of the blue trajectory line proves that the driver is generally able to navigate the low curvature snake with a smooth trajectory that follows the dotted lane centerline.

![Figure 4-30: Trajectory navigated by a blind driver on the low-curvature snake section of the VIR Patriot Course. Timing is indicated by the green crosses.](image)

A more detailed analysis of the driver’s navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 4-31. Observation of the speed deviation shows that the driver is able to start the vehicle from standstill and reach the 15mph speed limit within approximately 8 seconds with negligible overshoot. This is also clearly exemplified in the speed tracking response and error shown in Figure 4-32 and Figure 4-33, respectively. Upon startup, the instantaneous speed error quickly changed from 0mph to almost 15mph. The driver was able to recognize the error through the vibrations generated by the SpeedStrip interface on his middle thighs.
and knees and make the appropriate corrections with tracking performance similar to that presented in Section 3.2.4.

Figure 4-31: Lateral lane and speed deviations over the low-curvature snake section of the VIR Patriot course.

Figure 4-32: Reference speed generated by the PNVID and the associated blind driver response over the low-curvature snake section of the VIR Patriot course.

Figure 4-33: Instantaneous speed error exhibited by the blind driver over the low-curvature snake section of the VIR Patriot course.
Throughout the entire course section, the driver is able to keep the lateral lane deviation within the maximum limits and thus keep the tires on both sides of the vehicle fully within the lane boundaries. The maximum lane deviation occurs at approximately 33.5s and can be seen in the trajectory plot on Figure 4-30. Although the deviation at this point is as much as 5ft, the vehicle still remains within the driving lane with 1ft of clearance between the left wheels and the left lane boundary. The deviation occurs due to the combination of the complex steering actions required and the length of continuous steering wheel angle error leading up to the 33.5s marker. Figure 4-34 shows that the driver must make a 150° counter-clockwise turn with the wheel followed immediately by a 175° clockwise turn in a span of 7.75s. During this complex action, the driver exhibits continuous error larger than ±5° for approximately 7.6s, starting at 25.7s and ending at 33s in Figure 4-35. It is no coincidence that the largest deviation occurs at 33s. The amount of lateral deviation can be loosely related to the integral of the steering wheel angle error over the past couple of seconds in time. The larger the error and the longer it remains constant, the larger the lateral deviation will become.

Other than the 5ft deviation observed at 33s, the remainder of the lateral deviations fall within ±3ft from the lane centerline. The mean absolute steering deviation is calculated to be 1.26ft, while the mean absolute steering wheel angle error is found to be 5.19°. The MAE of the steering wheel angle is relatively large compared to the typical MAEs seen in Section 3.1.4.3 due to the interaction between the driver and the TORC AutonoNav™ motion planner of the dual closed-loop system. This combination causes the reference signal to continuously oscillate with low frequency when compared to the non-periodic arbitrary signals used in the experiments of Section 3.1.4.3. These oscillations require continuous corrective action from the driver and thus increase the MAE over the length of the timeframe.

Figure 4-34: Reference steering wheel angle generated by the PNVID and the associated blind driver response over the low-curvature snake course section.
Figure 4-35: Instantaneous steering wheel angle error exhibited by the blind driver over the low-curvature snake section of the VIR Patriot course.

4.4.3.2.2: High-Speed Straightaway

The high-speed straightaway section, shown in Figure 4-36, is a straightaway with a slight bend (0.006 rad) and holds a 25 mph speed limit as indicated by Figure 4-22. The purpose of this section of the course is to examine how well the blind driver steers the vehicle using the FFLT DOC discussed in Section 3.1.4.3 at higher speeds. The trajectory navigated by the blind driver can be seen in the section map shown in Figure 4-36. A quick observation of the vehicle trajectory shown by the blue line indicates that the driver is able to navigate this high-speed course section with generally low lateral deviation from the dotted lane centerline.

Figure 4-36: Trajectory navigated by a blind driver on the high-speed straightaway section of the VIR Patriot course. Timing is indicated by the green crosses and labeled accordingly.
A more detailed analysis of the driver’s navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 4-37. Observation of the speed deviation indicates that the driver is able to transition to the new 25mph speed limit from the previous 15mph limit in approximately 5s with no overshoot. The speed tracking and error plots given in Figure 4-38 and Figure 4-39 show that the driver is able recognize that he is approaching the correct speed through the transitions across the TE SOC error thresholds and was prepared to hold constant speed once the “no-vibration” stimulus was activated. The driver was able to maintain satisfactory speed error for the remainder of the course section using only the physical speed feedback discussed in Section 3.2.4.

![Lateral Lane and Speed Deviations](image)

**Figure 4-37:** Lateral lane and speed deviations over the high-speed straightaway section of the VIR Patriot course.

![Reference Speed](image)

**Figure 4-38:** Reference speed generated by the PNVID and the associated blind driver response on the high-speed straightaway section of the VIR Patriot course.
Figure 4-39: Instantaneous speed error exhibited by the blind driver over the high-speed straightaway section of the VIR Patriot course.

The lateral lane deviation plotted in Figure 4-37 indicates that the driver was certainly able to keep the vehicle well within the lane boundaries while using the FFLT DriveGrip operation configuration at higher speeds. The lateral lane deviations remain within ±2.5ft over the entire course section and exhibit an absolute mean of 0.97ft. However, the reference signal generated by the TORC AutonoNav™ and the PNVID required very little changes in the steering wheel angle, as shown in Figure 4-40. As the blind driver entered this course section, he had to make a slight counter-clockwise steering adjustment to enter the straightaway at a parallel angle. For the next 7s, the planner held the reference steering wheel angle constant at 0° even though the vehicle was not perfectly centered in the lane. Once the driver enters the slight right hand turn at 55s, the reference steering wheel angle is changed to a nonzero value in an effort to guide the driver through the curve. The driver’s reaction to the latter change in the reference signal does not respond with a steering wheel angle change until 56s. This 8s total timeframe accounts for the constant decreasing change in lateral deviation seen between 48s and 56s on Figure 4-37.

Because the reference steering wheel angle remained at 0° throughout the straightaway, this experiment unfortunately does not offer any conclusions as to whether or not the lenient tolerances of the FFLT DOC enable satisfactory lane center tracking at higher speeds. This is exemplified by the instantaneous steering error plot provided in Figure 4-41. The blind driver is able to make all of the necessary corrections using the DriveGrip interface with the FFLT configuration; however all of the required corrections are relatively large. The purpose of this course section was to see if minor corrections required by the motion planner at higher speeds would be ignored by the lenient tolerances of the FFLT and cause large lateral deviations or tracking instability. Although the experiment on this course section did not serve its main
purpose, it demonstrated that the driver is still able to navigate roads at speeds of 25mph even while using the FFLT DOC. This same result, including the constant reference steering value, was found throughout all navigation tests through the high-speed straightaway road section over the total 24 hours of track time at the VIR Patriot course. This suggests that the TORC AutonoNav™ system is configured to implement a trade-off between seeking the lane centerline and other planning parameters.

Figure 4-40: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the high-speed straightaway course section.

Figure 4-41: Instantaneous steering wheel angle error exhibited by the blind driver over the high-speed straightaway section of the VIR Patriot course.
One final note to consider is that the mean absolute error for the blind driver's steering angle tracking is $2.91^\circ$. When compared with the low-curvature snake section, the steering MAE and the mean absolute lateral deviation seem to decrease together. Therefore, the steering angle MAE and MALD will be recorded for each of the remaining course sections and any relationships that may exist will be investigated.

4.4.3.2.3: Medium-Curvature Snake

The medium-curvature snake section, shown in Figure 4-42, is a three-turn snake with a maximum curvature of 0.041 rad and equivalent turning radius of 81 ft. The course layout in Figure 4-22 indicates that this course section sets a 10 mph speed limit due to the increased curvature of the lanes. The trajectory navigated by the blind driver can be seen in the section map shown in Figure 4-42. A quick observation of the blue trajectory line proves that the driver is generally able to navigate the medium-curvature snake with a smooth trajectory that follows the dotted lane centerline closely.

A more detailed analysis of the driver's navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 4-43. Observation of the speed deviation shows that the driver is able to decrease from 25 mph to 10 mph in approximately 4 s with a small overshoot. Compared to accelerations, decelerations by the driver tend to exhibit overshoot as the driver is more attentive to decreasing speed as a cautious measure. However, examination of the speed tracking and error plots in Figure 4-44 and Figure 4-45 indicate that the overshoot does not exceed the lowest error threshold and thus the driver corrects the overshoot using his own physical perception of the vehicle's inertia as feedback. Once the driver settled to the 10 mph speed limit, he tended to travel at speeds slightly faster than the speed limit but still maintained a satisfactory speed for most of the course section. There were two instances where the driver temporarily exceeded the first error threshold and activated the middle back SpeedStrip stimulus. The driver responded to the stimuli both times with a slight deceleration and lowered the speed to within the acceptable limits.

The lateral deviation also shown in Figure 4-43 indicates that the driver was able maintain at least 3 ft of clearance between the lane boundaries and the vehicle tires for the entire duration of the course section. The maximum deviation from the lane centerline never surpassed $\pm 3$ ft of clearance and exhibited a mean absolute deviation of 1.73 ft. Closer examination of the lateral deviation and the vehicle trajectory shows that the driver tends to drive inside of the lane centerline over the length of each turn. Because these trends occur for extended periods of time, it is suggested that the TORC AutonoNav™ system is responsible as it plans trajectories that undercut the lane centerline.
The steering wheel angle tracking and error plots given in Figure 4-46 and Figure 4-47 indicate that the blind driver actually exhibits a significant amount of control effort throughout the medium-curvature snake. The reference steering signal generated by the AutonoNav™ and the PNVID hardly remains constant over the length of the course section. It seems that the interaction between the motion planner and the driver's steering response causes the reference signal to exhibit large steering wheel angles to match the curvature of each turn that are superimposed with smaller, higher frequency adjustments that are meant to make slight curvature corrections while in the turn. It is
these small adjustments that enable the driver to accurately stay within the lane boundaries even while navigating turns with considerable curvature. Although there was a significant amount of control effort required from the blind driver, he was still able to ensure that the steering wheel angle error never exceeded $\pm 16.5^\circ$. This limitation on the error means that the driver navigated the entire course section using only his index and middle fingers. The driver was able to achieve a mean absolute error of $4.48^\circ$ which, when related to the 1.73ft MALD, continues to match the trend of decreasing MAE with MALD over the entire course.

![Figure 4-43: Lateral lane and speed deviations over the medium-curvature snake section of the VIR Patriot course.](image)

![Figure 4-44: Reference speed generated by the PNVID and the associated blind driver response on the medium-curvature snake section of the VIR Patriot course.](image)
Figure 4-45: Instantaneous speed error exhibited by the blind driver over the medium-curvature snake section of the VIR Patriot course.

Figure 4-46: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the medium-curvature snake course section.

Figure 4-47: Instantaneous steering wheel angle error exhibited by the blind driver over the medium-curvature snake section of the VIR Patriot course.
4.4.3.2.4: Dynamic Obstacle Field
The dynamic obstacle field is a long, relatively straight section of the VIR Patriot course where obstacles are dynamically placed in front of the Blind Driver Challenge® vehicle to demonstrate the real-time perception and re-planning features of the research platform. The dynamic obstacles utilized in this experiment were cardboard boxes that were at least 36in tall and 24in wide on both sides. The boxes were dynamically placed by being thrown from the rear of a lead vehicle driving approximately 200ft in front of the Blind Driver Challenge® vehicle. Each box was weighted at the bottom to ensure that the tallest side stood upright upon landing and to prevent wind from moving the box after it was thrown from the lead vehicle. The boxes were dynamically placed approximately 150ft apart along the course section and were alternated on each side of the centerline to force the blind driver to slalom.

The course section itself, shown in Figure 4-48, is approximately 240m (787ft) long and uses slightly wider 22ft virtual lanes to provide the blind driver with more room to avoid the dynamic obstacles. As indicated by Figure 4-22, the speed limit is set to 15mph over the entire duration of the course section. The section map in Figure 4-48 additionally shows the position of the dynamically placed obstacles as detected by the perception system of the research vehicle. Each obstacle is denoted by a small red rectangle that encompasses the dynamic obstacle after it was thrown from the lead vehicle. Close examination of the blue trajectory line navigated by the driver generally indicates that the driver was able to avoid each obstacle while staying within the lane boundaries. Each encounter with an obstacle demonstrates deliberate changes in vehicle heading to avoid the obstacle and subsequently return to the lane center.

![Figure 4-48: Trajectory navigated by a blind driver on the dynamic obstacle field section of the VIR Patriot course. Timing is indicated by the green crosses.](image-url)
The lateral lane and speed deviations for this course section are plotted in Figure 4-49. The speed deviation in this figure indicates that the driver is able to transition from the 10mph speed limit from the previous section to the new 15mph speed limit in approximately 3s with no overshoot. As discussed earlier, the driver typically only exhibits overshoot when decreasing speed to a lower reference value. While the driver progressed through the dynamic obstacle field, the reference speed generated by the AutonoNav™ and PNVID was held constant as indicated in Figure 4-50. The presence of obstacles seems to have caused an irregular number of discontinuities in the reference signal; however these are all filtered out in the instantaneous error signal provided in Figure 4-51. The driver was able to maintain a constant speed within the acceptable speed limits for the majority of the course section and only had to make one small deceleration correction while approaching the final dynamic obstacle at ~152s.

**Figure 4-49:** Lateral lane and speed deviations over the dynamic obstacle field section of the VIR Patriot course.

**Figure 4-50:** Reference speed generated by the PNVID and the associated blind driver response on the dynamic obstacle field section of the VIR Patriot course.
Figure 4-51: Instantaneous speed error exhibited by the blind driver over the dynamic obstacle field section of the VIR Patriot course.

The lateral lane deviation plot also shown in Figure 4-49 differs from the deviation plots of previous course sections due to the presence of the dynamic obstacles. While the dashed blue line still represents the maximum deviations permissible before the vehicle crosses the lane boundary, the red lines indicate the maximum deviations permissible before the vehicle strikes a dynamic object in the lane. The red lines are actually very thin rectangles that horizontally represent the short length of time during which the vehicle was passing the obstacle. It is important to remember that this plot only indicates lateral distance information along the driving lane and does not include any longitudinal information. Therefore, only the distances along the vertical axis of Figure 4-49 should be considered as the horizontal axis indicates time and not distance. This plotting strategy helps visualize the lateral clearances achieved by the blind driver while passing each dynamic obstacle in the course section.

The lateral deviations shown in Figure 4-49 indicate that the driver is able to simultaneously avoid the dynamic obstacles and the lane boundaries over the entire course section. Each obstacle is avoided with at least 5ft of clearance, proving that the Blind Driver Challenge® system is capable of enabling a blind driver to avoid obstacles in real time. The lane boundary clearance at the passing of each obstacle is consistently less than the associated obstacle clearance, which suggests that the AutonoNav™ trajectory planner places a higher importance on clearing obstacles in comparison to avoiding lane boundaries. This seems to be a valid planning strategy as striking an obstacle will always cause an emergency while temporarily exceeding the lane boundary may not.

In general, the driver exhibits lateral deviations less than 7ft with a mean absolute deviation of 2.49ft even in the presence of the dynamic obstacles. The low MASD
suggests that the driver normally tracks the lane centerline except for quick deviations to avoid the dynamic obstacles where necessary. Figure 4-52 demonstrates that the considerably low MASD exhibited by the driver in this complex course section came at the cost of relatively complex control actions. During the 128s-132s timeframe, the TORC AutonoNav™ planned trajectories that significantly differed between each regeneration while it was attempting to find a suitable path around the first dynamic obstacle. The driver was able to generally follow this complex, oscillatory signal until the vehicle passed the first obstacle and the reference signal resumed normal behavior. A significant tracking error by the driver can be seen at this point during the 132s-134s timeframe, wherein the driver suddenly turns the wheel in the wrong direction against the instruction of the DriveGrip interface. This is exemplified by the increase in instantaneous steering wheel angle error shown in Figure 4-53 even though the reference remains constant and the interface is communicating a need for a right turn rather than a left. The driver was able to recognize this mistake and make the necessary correction quickly enough to have negligible adverse effects on the vehicle trajectory and lateral deviation.

The remainder of the reference signal was partially complex in that it required significant changes in steering wheel angle (up to 140°) mixed with smaller oscillations to re-acquire lane centering. The steering error plot in Figure 4-53 indicates that the driver was able to track the remaining partially complex reference with error less than ±23°. Over the entire course section, the blind driver exhibited a mean absolute steering error of 8.63°. This MAE and the corresponding MASD follow the typical trend seen in previous course sections; however the presence of obstacles makes this particular data set misleading. The motion planner and PNVID specifically choose reference steering signals that force the driver to increase lateral lane deviation from the lane centerline to

![Figure 4-52: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the dynamic obstacle field course section.](image-url)
avoid the obstacles. In this scenario, even if the driver tracks the reference perfectly and exhibits an MAE of 0°, the lateral lane deviation will still be high due to the avoidance of the obstacles. Therefore, the relationship between MAE and MASD is nullified for this particular course section.

![Graph showing steering wheel angle error](image)

**Figure 4-53:** Instantaneous steering wheel angle error exhibited by the blind driver over the dynamic obstacle field section of the VIR Patriot course.

### 4.4.3.2.5: Gate

The gate course section of this experiment, shown in Figure 4-54 utilizes a wall that partially obscures at least one half of the driving lane. The purpose of the gate is to force the driver to deviate from the lane centerline and navigate through a small opening placed off-center within the driving lane. The gate is constructed of 2.5ft diameter, 4ft tall barrels that are placed side-by-side to create a wall effect. These barrels can be visualized as the small red bounding rectangles on the course section map in Figure 4-54. The opening of the gate is located 5ft to the right of the lane centerline and is 14ft wide, giving the driver a maximum of 4.5ft of clearance on each side of the vehicle. Similarly to the previous dynamic obstacle field section, this section utilizes a 15mph speed limit. A quick observation of the blue trajectory line on the course section map indicates that the driver is able to not only navigate through the gate, but also return to the lane center shortly afterwards.

The lateral lane and speed deviations are pictured in Figure 4-55. The speed deviation for this course section remains at a constant value near zero because the speed limit is continuous from the previous course section, as shown in Figure 4-56. The driver was able to avoid the gate well enough that the motion planner did not have to request any temporary decreases in speed to avoid danger. The speed error plot in Figure 4-57 demonstrates that the driver maintains the constant speed exceptionally well and keeps the instantaneous speed error well below the lowest error threshold for the entire course.
section. The reference speed falls to 5mph at the end of each of these two figures in preparation for entering the following course section and should not be considered in the analysis for the gate course section.

![Graph of trajectory and speed deviations](image1)

**Figure 4-54:** Trajectory navigated by a blind driver on the gate section of the VIR Patriot course. Timing is indicated by the green crosses and labeled accordingly.

![Graph of lateral lane and speed deviations](image2)

**Figure 4-55:** Lateral lane and speed deviations over the gate section of the VIR Patriot course.

Similarly to the previous course section, the lateral lane deviation plot in Figure 4-55 includes the position of the gate wall as an obstacle in reference to the perpendicular distance from the lane centerline and time along the vehicle trajectory. Once again, distance is only communicated on the vertical axis of this figure while the horizontal axis communicates time. Thus, the longitudinal clearance of the gate should only be observed in the actual trajectory plot in Figure 4-54. The lateral deviations shown in Figure 4-55 indicate that the driver clears the gate with 3.5ft of clearance on the left side of the vehicle, but continues to increase the deviation to 7ft after passing through the
gate and only clears the right lane boundary by 1ft. The lateral deviation and trajectory plots suggest that the driver began the corrections to avoid the gate and right lane boundary with a small amount of delay. This can be confirmed by observing the steering wheel angle tracking and error plots given in Figure 4-58 and Figure 4-59, respectively. The tracking plot demonstrates that the driver response to the initial change in reference steering lagged by approximately 1100ms and induced error near 40° during the entire interval. Although the TORC AutonoNav™ motion planner partially offsets this lag with the planning horizon parameter, the lag is still too great and causes the trajectory to shift forward in time as exemplified in Figure 4-54. Fortunately, the comprehensive system is robust enough to still safely operate even in situations where the driver performance temporarily drops. Although the driver exhibited a considerably larger than normal lag time, he was still able to steer through the gate and return back to the center of the lane with acceptable clearance levels.

![Graph](image1)

**Figure 4-56**: Reference speed generated by the PNVID and the associated blind driver response on the gate section of the VIR Patriot course.

![Graph](image2)

**Figure 4-57**: Instantaneous speed error exhibited by the blind driver over the gate section of the VIR Patriot course.
204

Figure 4-58: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the gate course section.

Figure 4-59: Instantaneous steering wheel angle error exhibited by the blind driver over the gate section of the VIR Patriot course.

The mean absolute error and mean absolute lateral deviation exhibited by the driver is 7.99° and 2.73ft, respectively. Similarly to the dynamic obstacle field course section, the relationship between the MAE and MALD in this course section is nullified due to the presence of obstacles.

4.4.3.2.6: High-Curvature Snake
The high-curvature snake, shown in Figure 4-60, is a four-turn snake with 18ft wide lanes and curvatures as high as 0.0535rad (61ft turning radius). The turns in this course section alternate in direction and require the blind driver to track significant changes in steering wheel angle in order to be successfully navigated. As displayed in Figure 4-22, the speed limit for this section is set to 5mph due to the significantly low turning radii of the turns. The trajectory navigated by the blind driver can be seen in the section map shown in Figure 4-60. A quick observation of the blue trajectory line proves that the driver is able to navigate the high-curvature snake with a smooth trajectory that generally follows the dotted lane centerline.
Figure 4-60: Trajectory navigated by a blind driver on the high-curvature snake section of the VIR Patriot course. Timing is indicated by the green crosses.

The lateral lane and speed deviation exhibited by the driver over this course section is displayed in Figure 4-61. The speed deviation indicates that the driver was able to decrease the speed from 15mph to 5mph in 3.57s with only 0.71mph of overshoot. After initially reaching the correct speed, the driver demonstrated a tendency to travel at a slightly higher rate of speed and was forced to make several corrections to decelerate the vehicle and return to an acceptable speed. This is clearly demonstrated in the speed tracking and instantaneous error plots provided in Figure 4-62 and Figure 4-63, respectively. Although several corrections were required, the driver was able to quickly make the necessary decelerations and never exhibited speed error larger than ±3.5mph.

Figure 4-61: Lateral lane and speed deviations over the high-curvature snake section of the VIR Patriot course.
Figure 4-62: Reference speed generated by the PNVID and the associated blind driver response on the high-curvature snake section of the VIR Patriot course.

Figure 4-63: Instantaneous speed error exhibited by the blind driver over the high-curvature snake section of the VIR Patriot course.

The lateral lane deviations also shown in Figure 4-61 demonstrate that the driver was able to maintain deviations from the centerline within ±4ft over the entire high-curvature snake section. The largest deviations occur at the 227s, 242s, and the 254s mark and can be observed in the trajectory plotted in Figure 4-60. Examination of the steering tracking and instantaneous error plots in Figure 4-64 and Figure 4-65 suggest that the 227s deviation is a result of driver tracking error while the 242s and 254s deviations are not. At 225s, the driver began to exhibit a distinct loss in tracking performance that induced absolute errors as much as 40° and lasted for approximately 5s. Close examination of this time period in Figure 4-64 and Figure 4-65 shows that the driver was generally not turning the wheel counter-clockwise enough. Interestingly, the trajectory plot in Figure 4-60 indicates that the driver was overcutting the left turn and thus was steering too far to the left. This contradicts with what is shown in the steering tracking and error data and suggests that the AutonoNav™ was planning a path that purposefully overcut the left turn.
During the time leading up to the deviations at 242s and 254s, the driver does not seem to exhibit any significant amount of error in either size or duration. Figure 4-65 suggests that the driver was tracking the reference with a slightly elevated mean absolute error at the time of the deviations; however this does not necessarily act as the root cause of the deviations themselves. The 5s timeframe in Figure 4-65 just before the 242s deviation indicates that the system was requesting the driver to turn the wheel clockwise and increase the vehicle curvature in the right-hand direction. However, the trajectory in Figure 4-60 once again shows that the driver was already overcutting the corner in the right-hand direction during this timeframe. Once again, this suggests that the AutonoNav™ was planning trajectories that cut the corner of the turn.
The 5s timeframe leading up to the final significant deviation at 254s does not demonstrate any significant steering wheel angle errors in size or duration. This suggests that the driver was tracking the reference signal relatively well and that the deviations were once again caused by the AutonoNav™ motion planner. It seems that all three of the significant deviations over the high-curvature snake course section were caused by the TORC AutonoNav™ system. The same type of behavior has also been noticed in the medium-curvature snake course section. However, these particular deviations cause no particular harm as the driver and vehicle maintain at least 2ft of clearance from the lane boundaries at all times. It is important to remember that the AutonoNav™ system is not just a simple lane centerline-seeking planner and uses additional costs unrelated to centerline to calculate each trajectory [10].

Over the entire course section, the driver exhibits a mean absolute lateral deviation of 1.53ft and a mean absolute steering error of 4.64°. Despite the large deviations previously discussed, the driver demonstrates the ability to track the reference steering wheel angle with satisfactory accuracy and is generally able to navigate the center of the driving lane. Similarly to the object fields, it is debatable if the relationship between the MALD and the MAE is considered valid for this course section because of how the AutonoNav™ system forces the vehicle off the lane centerline. Even though the driver may track the reference very well and achieve low MAE, the MALD may be high and can distort the relationship observed.

4.4.3.2.7: Sweeping Turn
The sweeping turn section of the VIR Patriot course in this experiment is pictured in Figure 4-66. This particular section consists of a 600ft long sweeping right turn with 18ft lanes and a maximum curvature of 0.0075rad (436ft turning radius). The speed limit is set at 15mph, as dictated in the full course map in Figure 4-22. The trajectory navigated by the blind driver can also be seen in the section map in Figure 4-66. A quick observation of the blue trajectory line indicates that the driver is able to navigate the sweeping turn and is generally able to track the lane centerline. However, slight deviations that cut the inside of the right turn can be observed and will be discussed in the following paragraphs.

The lateral lane and speed deviation data for this course section is presented in Figure 4-67. The speed deviation plot indicates that the blind driver was able to increase the vehicle speed from 5mph to 15mph in 4.3s with no overshoot. Similarly to the discussions in past course sections, the driver typically exhibits lower rise times that result in no overshoot for increases in speed. This occurs because the accuracy of the speed increase is not considered to be vital to the safety of the driver when compared to speed decreases. Figure 4-68 and Figure 4-69 show that the driver was able to maintain the 15mph speed limit over the remainder of the course section except for a brief 0.12mph over-speed at 276.5s that was quickly corrected in fewer than 2s.
Figure 4-66: Trajectory navigated by a blind driver on the sweeping turn section of the VIR Patriot course. Timing is indicated by the green crosses.

Figure 4-67: Lateral lane and speed deviations over the sweeping turn section of the VIR Patriot course.
Figure 4-68: Reference speed generated by the PNVID and the associated blind driver response on the sweeping turn section of the VIR Patriot course.

Figure 4-69: Instantaneous speed error exhibited by the blind driver over the sweeping turn section of the VIR Patriot course.

The lateral lane deviation plot also provided in Figure 4-67 indicates that the driver was able to maintain a mean absolute lateral deviation of 1.39ft over the sweeping turns with a maximum deviation of 3.64ft. This maximum deviation occurs shortly after the driver enters the course section and peaks at 266s, as seen in both Figure 4-67 and Figure 4-66. The steering wheel angle tracking and error plots in Figure 4-70 and Figure 4-71 indicate that, during this timeframe, the driver was accurately tracking the reference signal. This suggests that the 3.64ft deviation was planned as part of the trajectory generated by the TORC AutonoNav™ system. The AutonoNav™ system generated this particular deviated trajectory because of a discontinuous connection between the previous high-curvature snake course section and the current sweeping turn course.
section. This discontinuous connection can be seen by closely observing the section connection point on the full course map in Figure 4-24. The discontinuity is not caused by the Blind Driver Challenge™ system itself but by a poor definition of the course map in the RNDF file (Section 2.6) that is used by the system. Fortunately, the system was robust enough to enable the driver to smoothly navigate the discontinuous connection with an acceptable 2.6ft clearance from the right lane boundary.

Figure 4-70: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the sweeping turn course section.

Figure 4-71: Instantaneous steering wheel angle error exhibited by the blind driver over the sweeping turn section of the VIR Patriot course.
A second large lateral deviation of 2.9ft is seen at 286s where the driver overcuts the turn and navigates inside of the lane centerline. The steering tracking and error data in Figure 4-70 and Figure 4-71 indicate no significant tracking errors, thus demonstrating once more how the AutonoNav™ motion planner tends to generate trajectories that overcut turns. Aside from the 3.64ft and 2.9ft deviations, the driver was able to keep within ±2ft of the lane centerline over the length of the sweeping turn course section. The mean absolute lateral deviation, including the significant deviations discussed earlier, was calculated to be only 1.39ft. In comparison, the driver maintained a mean absolute steering error of 4.5° and never surpassed ±17° of error. Similarly to the previous course sections, the relationship between the MAE and MALD in this section is partially corrupted by the two areas in which the AutonoNav™ motion planner deliberately deviated from the lane centerline.

4.4.3.2.8: Static Obstacle Field
The static obstacle field section of the VIR Patriot course in this experiment is pictured in Figure 4-72. This section is composed of four static obstacles that are placed in a slalom configuration and enforces a 15mph speed limit as shown in Figure 4-22. The static obstacles are made from the same 2.5ft diameter, 4ft tall barrels utilized in the gate course section and are placed on alternating sides of the lane centerline. The barrels are spaced approximately 100ft apart and are centered approximately 4.5ft from the lane centerline. The actual positioning of each barrel on the course can be seen by the red bounding rectangles placed in Figure 4-72. This particular configuration forces the driver to slalom around the static obstacles and thus perform relatively complex driving maneuvers. Figure 4-72 additionally shows the trajectory navigated through the course section by the driver as the blue line. It is easy to see that the driver simultaneously avoids the static obstacles and the lane boundaries as he travels through the slalom course.

The lateral lane and speed deviation data for this course section is supplied in Figure 4-73. The speed deviation plot demonstrates that the driver is able to continue tracking the 15mph speed limit as he transitions into the course section and is able to maintain suitable speed until the vehicle begins to approach the finish line at the end of the section. This behavior is further exemplified by the speed tracking and instantaneous error plots given in Figure 4-74 and Figure 4-75. Unfortunately, the decreases in speed due to the upcoming finish line could not be isolated to that particular course section as the AutonoNav™ system begins to request a decrease in speed while the driver is still passing the final obstacle at 318s.

Similarly to the dynamic obstacle field, the lateral lane deviation plot in Figure 4-73 includes the position of the static obstacles in reference to the perpendicular distance from the lane centerline and time along the vehicle trajectory. Once again, distance is only communicated on the vertical axis of this figure while the horizontal axis
communicates time. Thus, the longitudinal clearance of the gate should only be observed in the actual trajectory plot in Figure 4-72. The lateral lane deviation plotted in Figure 4-73 indicates that the blind driver was able to avoid each static obstacle with at least 5ft of clearance while maintaining at least 1.5ft of clearance from the lane boundaries. Once again, the driver tends to navigate around the obstacles with larger clearance between the obstacles as compared to the clearance between the lane boundaries. Since the steering wheel angle tracking and error plots in Figure 4-76 and Figure 4-77 indicate no significant steering error over the entire course section, this once again suggests that the TORC AutonoNav™ system tends to find higher costs in distance from an obstacle when compared to distance from a lane boundary.

Figure 4-72: Trajectory navigated by a blind driver on the static obstacle field section of the VIR Patriot course. Timing is indicated by the green crosses.
Figure 4-73: Lateral lane and speed deviations over the static obstacle field section of the VIR Patriot course.

Figure 4-74: Reference speed generated by the PNVID and the associated blind driver response on the sweeping turn section of the VIR Patriot course.

Figure 4-75: Instantaneous speed error exhibited by the blind driver over the sweeping turn section of the VIR Patriot course.
Figure 4-76: Reference steering wheel angle generated by the PNVID and the associated blind driver response on the static obstacle field course section.

Figure 4-77: Instantaneous steering wheel angle error exhibited by the blind driver over the static obstacle field section of the VIR Patriot course.

Although the trajectory navigated by the blind driver in Figure 4-72 is quite smooth, the driver was forced to track a relatively complex reference signal. The reference signal generated by the AutonoNav™ and the PNVID in Figure 4-76 includes steering angle changes of up to 120° and only holds a constant reference angle for a few short moments. Figure 4-77 shows that the driver is able to track the complex reference with absolute error less than 24°; however much control effort was required. The mean absolute steering error of this particular course section is 8.83° and is comparable to the 8.63° and 7.99° MAEs found in the dynamic obstacle field and gate course sections. The mean absolute lateral deviation of this course section is 2.27ft and can be compared with the 2.49ft and 2.73ft MALDs also found in the dynamic obstacle and gate course sections. It seems that course sections containing obstacles typically exhibit MAEs and MALDs in the neighborhood of 8.5° and 2.5ft, respectively.
4.4.3.2.9: Finish Line
The final section of the VIR Patriot course, shown in Figure 4-78, is a stop point located at the end of one full lap around the track. The purpose of this relatively small course section is to investigate the driver's ability to come to a complete stop at or before a stop point designated in the RNDF map file. It is important to note that this course section partially overlaps with the static obstacle field section because the TORC AutonoNav™ system began to decrease the reference speed in preparation for the stop point just after passing the final static obstacle. Therefore, lateral lane deviations can be expected in the first 3s of this section while the driver passes the final obstacle and returns to the lane centerline. The location of the stop point is indicated on the course map by a red point located at 0m, 0m.

![Figure 4-78: Trajectory navigated by a blind driver at the finish line section of the VIR Patriot course.](image)

The blue trajectory line shown in Figure 4-78 demonstrates that the driver was able to bring the vehicle to a complete stop before the stop point was reached. However, the trajectory data shown in this figure represents the position of the rear axle of the vehicle and does not communicate the distance between the front of the vehicle and the stop point. Figure 4-79 provides this information by plotting the distance between the front of the vehicle and the stop point as a function of time. The figure shows this same distance function at two different scales to display finer results near the actual stop point. Observation of this figure indicates that the driver does not begin to decelerate the vehicle from the initial 15mph speed until approximately 40ft from the stop point. After this initial deceleration, the driver approached the stop point at a relatively constant lower speed until a satisfactory distance was obtained. The driver
subsequently brought the vehicle to a complete stop with 1.21ft of distance between the front bumper and the stop point. It is important for the driver to be able to stop at or before the stop point to ensure that the vehicle does not overshoot stop signs or traffic lights and endanger the driver and other traffic.

![Graph](image)

**Figure 4-79**: Distance between the front of the Blind Driver Challenge® vehicle and the finish line stop point as a function of time at the finish line section of the VIR Patriot course.

The speed tracking data provided in Figure 4-80 demonstrates how the TORC AutonoNav™ system planned the vehicle stop as well as the driver's associated speed response. At approximately 130ft from the stop point, the motion planner and PNVID began to constantly decrease the reference speed until 30ft from the stop point. At this point, the reference decreased at a slower rate until the appropriate stopping distance was reached and a full stop was requested by the planner. The driver response also shown in Figure 4-80 and the instantaneous error plot in Figure 4-81 indicate that the driver performed two decelerations as commanded by the SpeedStrip interface and brought the vehicle to a smooth, controlled stop with low instantaneous error. The interaction between the driver and the AutonoNav™ motion planning actually enabled the driver to bring the vehicle to a smooth stop without the need to communicate a full stop stimulus on the SpeedStrip interface. The instantaneous error plot in Figure 4-81 shows that the instantaneous error exhibited by the driver never reached the Full Stop vibration zone because he led the reference and stopped the vehicle 933ms before it was necessary. Because the driver stopped with 1.21ft of distance between the front of the vehicle and the stop point, this suggests that the AutonoNav™ system was originally planning a trajectory that would result in the vehicle partially overshooting the stop point.
The lateral deviation plot shown in Figure 4-82 shows that, after returning to the lane centerline, the driver exhibits deviation that increases at a low constant rate until the vehicle is stopped and the stop point is achieved. The reference steering wheel angle shown in Figure 4-83 proves that this constant increase in lateral deviation occurs because the AutonoNav™ commands a completely straight trajectory until the stop point is reached. It seems that the motion planner simply attempts to reach a goal zone centered at the stop point and ignores tracking the lane centerline position while attempting to guide the vehicle into the goal zone itself. The instantaneous error plot in Figure 4-84 further proves that the driver tracks the reference signal accurately and that the AutonoNav™ motion planner is responsible for the resulting lateral deviations.
Figure 4-82: Lateral lane deviation throughout the finish line section of the VIR Patriot course.

Figure 4-83: Reference steering wheel angle generated by the PNVID and the associated blind driver response at the finish line course section.

Figure 4-84: Instantaneous steering wheel angle error exhibited by the blind driver over the finish line section of the VIR Patriot course.
4.5: Rolex 24 Public Demonstration

After successfully developing the Blind Driver Challenge® technologies discussed thus far in this work, the National Federation of the Blind, Virginia Tech, and TORC Robotics cooperated to show the world that a blind person could successfully navigate a full vehicle on a complex road course using the Blind Driver Challenge® system with the PNVID software. The demonstration was held at the Daytona International Speedway just prior to the annual Rolex 24 GRAND-AM race and was publicized by various media organizations. This subsection documents the plans, preparation, and results of this high-profile public demonstration.

4.5.1: Demonstration Plan

The Rolex 24 public demonstration was planned to be held on January 29th, 2011 at the Daytona International Speedway (DIS) as a pre-race activity for the annual Rolex 24 GRAND-AM race. The demonstration would include a completely blind driver independently navigating the Infield course of the DIS and performing several complex navigation tasks to demonstrate the capabilities of the driver and the Blind Driver Challenge® system. Figure 4-85 provides an image of the course map complete with navigation events and speed limits.

![Course Map](image)

Figure 4-85: Road course utilized at the Daytona International Speedway for the Rolex 24 Blind Driver Challenge® public demonstration.

The demonstration course began at the DIS start line with a 20mph speed limit. The blind driver would have to navigate the vehicle up onto the 18° bank and still maintain proper heading control of the vehicle. Upon entering the Infield course of the DIS, the speed limit drops down to 10mph so that the driver may avoid three static obstacles configured in a slalom formation. Each static object would be 100ft apart and centered
6ft from the lane centerline. After successfully avoiding the static obstacles, the driver would enter a 15mph zone through a chicane as he approaches the first hairpin turn. The hairpin turn utilizes a 10mph speed limit and ends at the location of the Infield National Federation of the Blind grandstands placed for this particular demonstration. The next course section would be a 25mph high-speed straightaway that displays the combined speed capabilities of the blind driver and the Blind Driver Challenge® system. After the high-speed straightaway, the driver would enter the dynamic obstacle field. In this 10mph course section, a lead vehicle would throw four 36” boxes in the path of the blind driver from approximately 150ft away. Figure 4-86 pictures the deployment of the dynamic obstacles during one of the practice runs at the VIR Patriot course. The purpose of this exercise would be to demonstrate that the system actively plans trajectories and enables the blind driver to safely navigate a dynamic environment. After one final hairpin turn, the blind driver overtakes the same lead vehicle at 20mph on the left side and enters a sharp, 5mph left-hand turn. Upon exiting the final turn, the blind driver would navigate through a series of walls configured to form an off-center, 15ft wide gate within the driving lane. After successfully passing through the gate, the driver would subsequently reach the finish line of the road course and bring the vehicle to a complete stop.

Figure 4-86: Deployment of dynamic obstacles exemplified during a practice session at the Virginia International Raceway Patriot course.
4.5.2: Preparation
The Rolex 24 public demonstration was a complex and high-profile event that required significant amounts of preparation to ensure perfect execution. The demonstration depended not only on fully operational and robust Blind Driver Challenge® technology, but also on fully trained blind drivers that were capable of utilizing the technology itself. This subsection documents the timeline of the preparation process as well as the practice runs utilized to select and prepare a blind driver for the demonstration.

4.5.2.1: Timeline
The following bullet points provide a timeline of the development, testing, training and practice steps taken in preparation for the Rolex 24 public demonstration.

- **June 1, 2010**: Two stock 2010 Ford Escape Hybrids (Section 2.1.1) were acquired.
- **June 30, 2010**: TORC Robotics ByWire XGV™ conversion (Section 2.1.2) completed on both vehicles.
- **July 15, 2010**: Meeting with the NFB, Virginia Tech, and TORC to determine project milestones leading up to the Rolex 24 public demonstration.
- **August 16, 2010**: Environmental perception sensor (Section 2.2) selection process complete. Sensor ordering process begins.
- **August 24, 2010**: Non-Visual Interface System (Section 3, Section 4) development begins.
- **October 31, 2010**: First vehicle completely outfitted with environmental perception sensors.
- **November 19, 2010**: Delivered Blind Driver Challenge™ simulator package (Section 2.5) to the NFB for driver training.
- **December 1, 2010**: Performed system testing and driver training on the Virginia International Raceway (VIR) Patriot course for 8 hours.
- **December 15, 2010**: Performed system testing and driver training on the VIR North course for 8 hours. Also, the final selection of the driver and backup driver for the Rolex 24 demonstration was made. See Section 4.5.2.2 for more detail on the selection process.
- **January 1, 2011**: Second vehicle completely outfitted with environmental perception sensors.
• **January 10-11, 2011:** Performed system testing and driver training at the Virginia Tech Transportation Institute (VTTI) Smart Road for 16hrs.

• **January 17-19, 2011:** Finalized system software/hardware and performed full demonstration practice for 24hrs with selected drivers as shown in Section 4.4.3.

• **January 25-28, 2011:** Arrived at the Daytona International Speedway and performed 2hrs of on-track demonstration practice. Additionally performed 20hrs of dynamic obstacle and general navigation practice in a local, closed parking lot.

• **January 29, 2011:** Completed the full Rolex 24 public demonstration with perfect execution.

4.5.2.2: **Driver Selection**

Early in the preparation process, the National Federation of the Blind internally selected five candidates to serve as the blind driver for the Rolex 24 public demonstration. The candidates varied in age, degree of blindness, and in previous experience with driving an automobile. A committee containing representatives from the NFB, Virginia Tech, and TORC Robotics was formed to judge the performance of each candidate based on a variety of metrics and ultimately choose one candidate to act as the official demonstration blind driver. The judging process was conducted during the first 16hrs of full system testing at the Virginia International Raceway on December 1st and 15th of 2010. During this timeframe, only one vehicle was complete with the environmental perception sensors (Section 2.2) and possessed the full Blind Driver Challenge® system functionality. In order to provide the five candidates with more driving time for judgment, a second 2010 Ford Escape Hybrid was specially outfitted with the Non-Visual Interface System (Section 3) and a remote control. The remote could be used by a sighted passenger to communicate non-visual stimuli to the blind driver in a manner similar to the complete Blind Driver Challenge® system. Utilizing this specialized secondary vehicle along with the original complete vehicle doubled the amount of driving time each candidate received during the 16hr judgment period. Each candidate would navigate one full lap around course and then switch out with another candidate. The selection committee was evenly dispersed in each vehicle and also rotated between vehicles at random. During each lap, the committee would observe the performance of each candidate and take note of their skills and shortcomings. The process repeated continuously until the allotted track time for each day expired.

At the end of the second testing period on December 15th, 2010, the selection committee met in private to discuss the performance of each candidate and reach a unanimous selection of the official demonstration blind driver. Initial discussions in the selection process brought forward two particular candidates who undoubtedly demonstrated the best navigation skills using the Non-Visual Interface system. These
two candidates performed equally well in reference signal tracking for steering and speed and were able to completely navigate the testing course with the highest performance. The candidates also demonstrated the most understanding of the underlying technology and were able to use this knowledge to partially maintain navigation performance even in situations where a lapse in technology performance presented itself. The final metric that was utilized to select the official blind driver was based on the candidate’s ability to act as the face of the Blind Driver Challenge® to the general public after the demonstration was conducted. As a director within the NFB Jernigan Institute and a long-time leader within the Blind Driver Challenge®, Mark Riccobono was therefore chosen as the official blind driver for the Rolex 24 public demonstration. The runner-up candidate, Anil Lewis, was subsequently selected as the backup blind driver for the demonstration should Mark Riccobono suddenly become unable to fulfill his duties.

4.5.3: Demonstration Results

The significant amount of work performed in preparation for the Rolex 24 demonstration enabled Mark Riccobono to navigate around the DIS demonstration course described in Section 4.5.1 with perfect execution. He was able to stay well within the lane boundaries of the predefined DIS RNDF map file over the entire course and reached speeds up to 27mph on the high-speed straightaway. Mark avoided both the static and dynamic obstacles on the course with acceptable clearances and subsequently returned to the lane centerline after each obstacle field. He also successfully passed the lead vehicle at 20mph, navigated through the 15ft gate, and brought the vehicle to a complete stop upon reaching the finish line. Aside from the pre-planned navigation tasks, Mark also interacted with the on-looking crowd while driving by honking the horn at the location of the NFB grandstands and while passing the lead vehicle. The successful demonstration was publicized internationally in both print and video by various media organizations, proving to the world that blind people can independently drive a vehicle with the specially designed technologies presented and discussed in this work. Photos demonstrating the success of the Rolex 24 public demonstration can be seen in Figure 4-87 through Figure 4-90 below.
Figure 4-87: Florida Congressman John Mica (right) hands the keys to the Blind Driver Challenge® vehicle to Mark Riccobono (middle) and Dr. Marc Maurer (left).

Figure 4-88: Mark Riccobono driving the Blind Driver Challenge® vehicle on the 18° bank of the Daytona International Speedway shortly after the starting line.
Figure 4-89: Mark Riccobono approaching the gate course section near the finish line.

Figure 4-90: Mark Riccobono (second from left) and Dr. Dennis Hong (right) celebrating the successful demonstration.
Section 5: The Adaptive Non-Visual Interface Driver (ANVID)

The Adaptive Non-Visual Interface Driver (ANVID) is a specialized software component designed to adaptively generate non-visual stimuli based on trajectories planned by the TORC Robotics AutonoNav™ system. This software acts as a reference transformation technology and solves the third issue of the Problem Statement (Section 1.3) by converting the planned trajectory information into non-visual stimuli that can be reliably understood and interacted with by the driver. Additionally, the non-visual stimuli generated by the ANVID are adaptive in that the stimuli are optimized for each particular driver through real-time driver modeling and prediction. This technique implements Model Predictive Control (MPC) to determine non-visual stimuli that maximize the driver's performance based on the ability to predict the driver’s actions for a given set of stimuli. The now complete and adaptive instructional solution to the Blind Driver Challenge® utilizing the ANVID as a reference transformation technology is shown in Figure 5-1.

![Diagram](image-url)

Figure 5-1: Complete instructional solution to the Blind Driver Challenge® utilizing the ANVID software. The Research Platform refers to the Blind Driver Challenge® Research Platform and is described in detail in Section 2. The NVIS refers to the Non-Visual Interface System and is presented in detail in Section 3.
This section focuses on providing a complete and detailed description of the ANVID software as well as a performance analysis that investigates the comprehensive abilities of the software. First, a list of functional requirements will be defined that will explain what issues must be solved by the ANVID software. Next, the algorithms within the ANVID software will be fully defined. Finally, a detailed description and performance analysis of the full software implementation is provided at the conclusion of this section.

5.1: Functional Requirements

The primary function of the ANVID software is to transform planned driving actions from the TORC Robotics AutonoNav™ system into non-visual stimuli that can be communicated over the Non-Visual Interface System. As described in Section 2.3.2.2, the Motion Planner of the AutonoNav™ generates desired trajectories for the vehicle to follow at a variable rate of 2-8Hz and encodes them into motion profiles through a trajectory model. The motion profiles, defined in Section 2.3.2.2.3, contain sequential curvature and velocity data that can be transformed into non-visual stimuli and communicated to the driver so that he or she physically realizes the desired trajectory of the AutonoNav™ motion planner. The purpose of the ANVID software is to perform this transformation adaptively and generate optimal stimuli that are communicated over the Non-Visual Interface System described in Section 3. This primary functional requirement can be broken down into several sub-problems that are described in the following subsections.

5.1.1: Motion Profile Sawtooth Patterns

The first issue that must be mitigated by the ANVID software is the motion profile sawtooth patterns that are caused by the rapid regeneration of the motion profiles themselves. This phenomenon has been previously explained in detail in Section 4.1.1 and still applies to the ANVID software because the motion profiles ultimately supply the system with a desired trajectory that the driver must follow. The ANVID must transform the sawtoothed curvature and speed reference signals encoded in the motion profiles into optimal non-visual stimuli that are smooth. Since the sawtooth waves dramatically interfere with the driver’s ability to track the reference steering signal, the ANVID software must eradicate the waves from the reference and ensure that the resulting non-visual stimuli can be reliably understood and interacted with by the driver. The software must also perform this transformation in a way that minimizes additional time lag injected into the system to avoid further increases in the ultimate latency of the driver’s responses.

5.1.2: Driver Time Delay

Section 3.1.4 and Section 3.2.4 demonstrated that driver responses to reference steering and speed signals exhibit varying amounts of pure time delay that limit his or her ability to accurately recreate vehicle trajectories planned by the TORC Robotics
AutonoNav™ system. Not only do these time delays vary from driver to driver, but they also vary for each driver in time. The time delays are the result of the driver’s perception, processing, and neuromuscular dynamics and have long been a major factor to consider in driver modeling [13] [20] [21] [22] [28] [29]. The PNVID software presented in Section 4 implemented a single average time delay calculated from driver data to offset the steering and speed response time delays of all drivers utilizing the system at all times. While this method generally proved robust, delays were still found in the driver’s navigated trajectory in Section 4.4 that were the result of insufficient delay offset by the PNVID system. The ANVID software must be able to adapt to each driver’s time delay and continuously provide accurate delay offset to prevent trajectory delays from occurring as found with the PNVID software. This adaptive method will improve the driver’s accuracy in recreating the planned trajectories and increase the overall stability of the closed loop system.

5.1.3: Driver Tracking Characteristics

Even without the consideration of time delay, drivers still exhibit significantly different tendencies when tracking reference steering and speed signals using the DriveGrip and SpeedStrip non-visual interfaces. Much of the initial blind driver testing discussed in Section 4.5 showed that drivers reacted to the non-visual stimuli in many different ways. One group of drivers tended to make slower corrections in response to a particular non-visual stimulus, leading to overdamped-like responses that took a significant amount of time to converge to the reference value. Other drivers tended to make faster corrections and exhibited underdamped-like responses that induced overshoot and oscillations centered on the reference values. These variations occurred across all drivers and demonstrated significant differences in reference signal tracking abilities and trajectory recreation.

The ANVID software must therefore be able to adapt to each driver’s particular tendencies in responding to non-visual stimuli communicated by the Non-Visual Interface System. Such adaptivity will ensure that each driver is provided with an optimal set of non-visual stimuli that will induce optimal trajectory recreation from the driver. It can be considered that the ANVID will manipulate the driver based on his or her particular response tendencies to ensure that the reference steering and speed signals are tracked with higher accuracy. This method will improve the overall navigation abilities of the driver regardless of his or her particular interaction tendencies with the DriveGrip and SpeedStrip interfaces.
5.2: Functional Overview

The ANVID software presented in this section uses Model Predictive Control (MPC) and real-time driver modeling to generate non-visual stimuli that adapt to each driver’s tracking tendencies. MPC is a technique that seeks suboptimal control inputs to a dynamical system that are determined by predicting the system’s output using a model of the system. The ANVID software uses MPC to determine suboptimal reference steering wheel angle and speed signals that are predicted to induce the closest match between the trajectory planned by the TORC AutonoNav™ system and the trajectory navigated by the driver. By creating a different model for each driver that utilizes the system, the ANVID can tailor the non-visual stimuli to elicit higher accuracy trajectory recreation regardless of the driver and his or her tracking tendencies. Furthermore, use of a continuously online training model will allow the ANVID to adjust the stimuli as the driver’s tracking tendencies change based on the particular driving scenario or as tracking skill improves over time.

The high-level functionality of the ANVID software is defined in Figure 5-2. The processes in this diagram all run in parallel and share necessary information through shared memory. The first process, called the Message Handler, simply provides the most current motion profile sent by the TORC AutonoNav™ system as well as the current steering wheel angle and speed of the vehicle. The current steering and speed values are ultimately controlled by the driver’s actions and thus represent the driver’s response. The Model Predictive Controller uses the current motion profile and the continuously trained driver model to calculate a set of suboptimal reference steering and speed signals that the driver must track to recreate the planned trajectory. The Non-Visual Stimuli Generator uses the reference signals and the current steering and speed data induced by the driver to calculate and generate the non-visual stimuli over the Non-Visual Interface System. While this process occurs, Driver Modeling software running in the background continuously monitors the reference signals and the current steering and speed signals. The modeling software uses this information to create a model that can predict what the driver’s steering and speed responses will be to a particular set of reference signals. It is this prediction model of the driver that is used by the Model Predictive Controller to determine a suboptimal set of reference signals.

With this process, the ANVID software generates an entirely new horizon of future reference steering and speed data each time a new motion profile is received from the AutonoNav™ system. For example, a motion profile may contain commanded steering wheel angle and speed actions for the next 3 seconds. The ANVID software would pass this 3s of future commanded actions through the MPC process to calculate 3s of future reference steering and speed signals. These reference signals would be sub-optimally chosen to ensure that the driver’s associated tracking response over the next 3s would most closely match the original 3s of commanded actions from the motion.
profile. However, a special feature of the ANVID is that it turns the previous disadvantage of rapidly-regenerated motion profiles (Section 4.1.1) into an advantage. Because the motion profiles are regenerated between 2Hz and 8Hz, the ANVID software only needs to generate a maximum of 0.5s of future reference steering and speed signals even though the motion profile may contain more than 0.5s of future steering and speed commands. The minimal 2Hz rate replaces the current motion profile with a new one at a maximum of \((2Hz)^{-1} = 0.5s\). Any reference signals calculated by the ANVID software that have a horizon larger than 0.5s will never be fully completed and waste valuable computation cycles to calculate. This means that the ANVID software must only utilize the MPC and driver modeling processes to calculate data with a 0.5s-long horizon at maximum. Such a small horizon greatly reduces the amount of computation time and makes the complex ANVID implementation possible.

![Figure 5-2](image)

**Figure 5-2: High-level functionality diagram of the ANVID software.**

The ANVID software uses particular algorithms for the driver modeling and optimization processes that provide the most significant performance advantages in terms of implementation and driver trajectory recreation. The Driver Modeling process uses Artificial Neural Network (ANN)-based models to predict what the driver’s steering and speed responses will be to a particular set of reference steering and speed signals. The ANN models are trained with online supervised learning with a Multiple Extended Kalman Algorithm (MEKA) trainer. The optimization process implements the Quasi-Newton method to decrease optimization time using Hessian information. The primary advantage of this method is that the Hessian can be approximated and does not require explicit calculation. While a brief description of these modeling and optimization processes have been provided in this subsection, the detailed theoretical definitions and algorithmic implementations are given in Section 5.3 and 0.
5.3: Theoretical Definitions

As discussed in the Functional Overview in the previous section, the ANVID software utilizes Model Predictive Control (MPC) to ultimately determine adaptive the non-visual stimuli that are communicated to the driver. It was also briefly mentioned that the MPC method implemented by the ANVID software internally uses Artificial Neural Network (ANN) based driver models and Quasi-Newton optimization. This subsection focuses on providing the theoretical definitions of these processes before the actual algorithmic implementation is defined in Section 5.4.

5.3.1: Model Predictive Control (MPC)

Model Predictive Control (MPC) is a method that seeks optimal control inputs to a dynamical system that are determined by predicting the system’s output [85]. Essentially, MPC is an optimization process that incorporates a model of the system within the objective function. New candidate solutions for the control inputs are calculated and passed through the system model until the model-predicted output of the system minimizes a user-defined objective function. As discussed in the literature review of this work (Section 1.4), MPC has been used by many researchers and engineers to control various types of dynamical systems. One of the major advantages of MPC that is demonstrated by the discussed literature is the ability to implement many different combinations of modeling and optimization schemes. For example, [89] [90] utilized a linear finite impulse response (FIR) model with a heuristics based iterative algorithm optimizer while [97] utilized an artificial neural network based model with a reduced hessian SQP optimization scheme. This configurable ability enables the MPC method to be applied to many different types of complex dynamical systems with satisfactory results. The works discussed in Section 1.4.4 also reported significantly improved controller performance over standard PID-type controllers and were even able to be implemented on nonlinear, time-invariant systems.

The theoretical MPC process used in the ANVID software is defined in Figure 5-3. The process begins by supplying the MPC controller with a reference value \( r \) that is desired for the dynamic system to track. The controller then uses an optimization process to determine a control input solution that is predicted to exhibit the lowest performance cost. The cost of the system response is completely customizable and can include metrics such as how well it tracks the reference signal or how much control effort is required. The performance cost is able to be predicted by the optimization process through the use of a model of the dynamical system being controlled. At each iteration of the optimization process, the current control input candidate \( \hat{u} \) is passed through the system model and provides the predicted output of the system \( \hat{y} \) for that particular candidate input. The predicted output and the candidate control input can then be passed through an objective function that determines the overall performance cost \( J \) associated with the candidate control input solution. The cost is then compared
to a maximum threshold and the entire process is repeated until the cost falls below the threshold and a certain level of predicted system performance is reached. The candidate control input ($\hat{u}$) that induced this desired level of predicted performance is then declared the optimal control input ($u$). The MPC controller then applies this optimal control input ($u$) to the dynamical system and repeats the entire process in the next controller iteration.

**Figure 5-3: General process flowchart for the Model Predictive Control method.**

The ANVID software supplements the Model Predictive Controller (MPCr) shown in Figure 5-3 with a system model that is continuously trained online in real-time using a parallel training process. This provides the MPCr with an up-to-date model of the dynamical system and allows the ANVID to adapt in time with a time-variant system. Since the human driver is certainly a time-variant system, the ANVID must adopt this technique to account for changes in the driver’s input/output relationship during different driving scenarios and as the driver’s skill increases.

The Functional Overview in Section 5.2 describes that the ANVID utilizes MPC to find a finite horizon of suboptimal future reference steering and speed signals based on the horizon of future commanded steering and speed actions specified by the most current motion profile. The reason that the MPCr calculates these suboptimal reference signals is to essentially “trick” the driver into accurately recreating the commanded actions from the motion profile. The analyses of driver responses to steering and speed reference signals using the Non-Visual Interface system discussed in Sections 3.1.4 and 3.2.4 gave a clear indication that drivers exhibit poor tracking characteristics such as pure time delay, overshoot, and oscillatory behavior. If the commanded actions from a motion profile were directly fed as a reference to the driver, one could not expect the driver to perfectly track the commanded actions and recreate the planned trajectory with high accuracy. Although Section 4.4 demonstrated that the system is robust enough to still function properly, the ANVID algorithm seeks to improve the driver’s navigation performance by indirectly improving the driver’s reference tracking abilities. The ANVID uses MPC to find what particular reference signals the driver must track so that his or her associated response actually recreates the commanded actions.
An example of this process is pictured in Figure 5-4. In this case, the untransformed reference signal (UREF) is the steering signal commanded by the current motion profile. The driver must track the commanded steering signal (UREF) accurately in order to recreate the trajectory planned by the AutonoNav™ and safely navigate the vehicle. However, the driver’s response to the untransformed reference (URESP) exhibits pure time delay with overshoot and does not track the commanded steering signal accurately at all. MPC is used to mitigate this problem by finding a suboptimal transformed reference signal (TREF) that the driver will track instead of the commanded signal (UREF). The driver’s response to the transformed reference (TRESP) still exhibits the same time delay and overshoot compared to the transformed reference (TREF); however the response almost perfectly recreates the commanded reference (UREF) that the driver was required to recreate in the first place. The MPC process essentially performs the calculations for Figure 5-4 repeatedly by experimenting with different transformed reference signals (TREFs) until the associated model-predicted response (TRESP) closely matches the commanded signal (UREF).

![Figure 5-4: Example of how MPC is used to control the human driver with fictional data. UREF is the untransformed reference steering signal and URESP is the driver’s response to UREF. TREF is the suboptimal transformed reference steering signal calculated by MPC and TRESP is the driver’s response to TREF. Note that the TRESP accurately recreates UREF.](image)

The theoretical definitions of the driver modeling and optimization processes utilized by the ANVID MPCr are provided in the following subsections. These definitions will discuss how the modeling is performed as well as how the optimization process actually determines a suboptimal set of reference signals for the driver to track. Once the theoretical definitions have been discussed, the actual algorithm that implements the ANVID can be defined.
5.3.2: Driver Modeling

The model of the driver itself is one of the most important components of the MPC process utilized by the ANVID software presented in this section. The driver model is what enables the ANVID to adapt to drivers with different and time-varying tracking tendencies. Therefore, the model of the driver must be not only accurate but also continuously updated. The difficulty of this problem is increased by the fact that the human driver is considered as a dynamical system that is highly nonlinear, time variant, and stochastic in nature [34] [35]. Many publications have focused on creating models for such a complex system with varying degrees of success, such as [125] [126] [127] [128] [129] and the literature discussed in Section 1.4.1. More recent attempts have started to implement the driver models with Artificial Neural Networks (ANNs) due to their ability to model nonlinear systems. It was particularly noted in [40] and [41] that the inherent nonlinearity and adaptivity of ANNs can create accurate models of the nonlinear driver-vehicle system over a wide range of operating conditions and scenarios. For these reasons, this work utilizes ANNs to model the driver as part of the ANVID MPC process. The following subsections describe which particular ANN is used to model the driver, how it is applied to the ANVID, and how it is able to be trained in real time.

5.3.2.1: Time Series Prediction Neural Networks (TSPNNs)

A dynamical system can be defined as a system with time-dependent transitions from one state to another state. The system contains state memory, meaning that each state created is partially a result of the previous states. To accurately model such a system, the model must incorporate dynamics of its own that hold some sort of state memory. Conditions of the past states must be considered by the model when predicting the future states of any dynamical system. For this reason, the driver models presented in this work use Time Series Prediction Neural Networks (TSPNNs) to model the driver dynamics and predict future outputs from the driver.

A TSPNN is a particular neural network architecture that uses past outputs of the actual dynamical system as network inputs to help predict future system outputs [130]. The TSPNN is, in its simplest form, implements a focused time delay neural network (TDNN) as a standard multi-layer perceptron (MLP) with memory external to the network [131]. The advantage of this system compared to a recurrent neural network is that the training can be performed online in a rapid manner due to its static nature. Recurrent networks are inherently dynamic due to their internal use of feedback and thus each training iteration must converge to a fixed-point solution of updated weights. This takes significantly more time than the single-step training iterations used by the MLP and cannot be used online to quickly adapt to time-variant systems. The TPSNN is able to model the system dynamics similarly to recurrent neural networks and also offers the ability to rapidly adapt online.
An example of a typical TSPNN is provided below in Figure 5-5. The TSPNN is simply a multi-layer, feed-forward perceptron with network inputs that include a finite horizon of past actual system outputs ($y$) and future system inputs ($u$) to predict a finite horizon of future system outputs ($\hat{y}$). The window of past outputs acts as a state memory that enables the network to predict future the states of a dynamical system. The TSPNN may be configured for any amount of inputs and outputs for any size horizon, making it applicable to all types of dynamical systems. One of the major advantages of such a model is that the model designer must only consider what particular inputs influence the output of the dynamical system in question. The designer does not need to make any attempt at understanding the complex underlying dynamics of the system because neural networks are a pure black-box system identification process.

Figure 5-5: Typical usage of a Time Series Prediction Neural Network (TSPNN).

As previously discussed, the neural network actually utilized by the TSPNN is a standard feed-forward multi-layer perception. Theoretically, the neural network has the ability to contain any amount of inputs, hidden layers, and outputs. Additionally, the neurons within the network may utilize any type of nonlinear sigmoid function and also have the option of including bias. The ability to choose the particular neural network configuration with the TSPNN further enables the successful customization and application of TSPNN modeling to many different types of dynamical systems.

However, a tradeoff exists with the highly configurable characteristic of TSPNNs. Although the TSPNN can be applied to a wide range of dynamical systems, there is no scientific method to determining the actual configuration of the TSPNN. Most configurations are determined empirically, although there have been attempts to use optimization techniques to determine optimal network configurations [132] [133] [134]. Since these techniques must occur offline, the online driver models presented in this paper will use TSPNNs configured through empirical means.
5.3.2.2: ANVID Driver Model Definition
As discussed previously in this section, the MPC process utilized by the ANVID software uses a driver model to determine suboptimal reference steering and speed signals for the driver to track. An optimization process within the Model Predictive Controller uses the driver model to predict how well the driver will match the commanded actions from the current motion profile when utilizing candidate reference signals. Therefore, a driver model must be in place that can predict what the steering and speed responses will be to a particular set of reference steering and speed signals. Before such a model can be created, the full Blind Driver Challenge® system using the ANVID software must be observed as a dynamical system. Figure 5-6 displays the dual-loop dynamical system the represents the combination of the driver and Blind Driver Challenge® technologies.

![Figure 5-6: Full Blind Driver Challenge® system block diagram.](image)

Figure 5-6 indicates that the driver utilizes the non-visual stimuli communicated over the DriveGrip and SpeedStrip interfaces to control the instantaneous steering wheel angle and speed of the vehicle. Different factors, such as the inertial forces perceived by the driver, can certainly act as additional inputs; however these are assumed negligible compared to the non-visual stimuli inputs and are ignored in this implementation. At a rate of 2-8Hz, the outer loop containing the Motion Planner and Model Predictive Controller generates a new horizon of future steering and speed reference data. The driver then closes the loop internally by attempting to minimize the difference between the reference and actual steering and speed data at 40Hz. It should be noted that, because the inner loop runs at a higher sample rate than the outer loop, the dual loop system can be implemented in a stable manner.
Theoretically, the true inputs to the driver are the instantaneous steering and speed error \((\theta_E, S_E)\) communicated as non-visual stimuli through the DriveGrip and SpeedStrip interfaces. However, the MPCr can optimize the reference steering and speed values \((\theta_R, S_R)\) instead with the same result as long as the driver model predicts the actual steering and speed responses. Recalling from (3.1) and (3.9), the instantaneous error for steering and speed is calculated directly from the actual and reference values. If the model predicts the actual values, then either of the associated error or reference values can be inferred with knowledge of the other used as an input. In order to conceptualize the full system in a more simplistic manner, the ANVID MPCr uses a driver model that encapsulates the inner system loop from Figure 5-6 and thus predicts the driver’s actions \((\theta_A, S_A)\) using the reference data \((\theta_R, S_R)\) as an input. The block diagram for this particular simplified system is depicted in Figure 5-7.

![Figure 5-7: Simplified Blind Driver Challenge® system block diagram.](image)

The simplified system diagram in Figure 5-7 shows that the steering and speed states of the closed-loop driver as a dynamical system are in fact coupled. This coupled relationship was additionally seen in the simultaneous steering and speed tracking results presented in Section 3.3. Unfortunately, initial efforts to create a model that could successfully predict the behavior of the coupled steering and speed states proved quite difficult. The coupled model required significant computational time and memory to store and train which ultimately caused the production of sub-standard prediction results. Therefore, the simplified system was decoupled to improve the prediction performance and ensure that the model could train online in real time. By decoupling the steering and speed states, this implementation assumes that the interaction of the two states is negligible compared to the non-visual stimuli in influencing future outputs by the driver. While this assumption will be shown as valid in later discussions within this section, it is still suggested that future research be conducted to implement an MPC process that keeps the coupled relationship intact. That being said, the decoupled system can be seen in the block diagram representation provided in Figure 5-8.
Figure 5-8: Decoupled simplified Blind Driver Challenge® system block diagram.

At this point, the full Blind Driver Challenge® system has been reconfigured so that a functional driver model may be implemented to accurately predict the driver’s steering and speed actions ($\theta_A$, $S_A$) in real-time. More specifically, two separate models are implemented as the steering and speed states have been decoupled. One model predicts the driver’s steering actions in response to a particular reference steering signal. A second model predicts the driver’s speed actions in response to a particular reference speed signal. The ANVID software makes use of two separate TSPNNs to implement the described steering and speed driver models. Recalling from the previous subsection, TSPNNs use past output data and future input data to predict future output data. The TSPNN driver models therefore utilize past steering and speed data as well as future reference steering and speed data to predict the future steering and speed actions of the driver. The MPCr is then able to use these models to find suboptimal reference values for both steering and speed to induce responses from the driver that most closely match the commanded actions designed by the current motion profile.

To begin defining the TSPNN models for steering and speed, the iteration timing of the actual software implementation must be first considered. As previously discussed, the inner loop shown in Figure 5-6 involving the driver’s steering and speed actions iterates at a frequency of 40Hz. This frequency therefore extends to the measurement of instantaneous steering and speed values ($\theta_A$, $S_A$) at every 25ms. The outer loop in Figure 5-6 is responsible for calculating reference values from motion profile data iterates at a frequency between 2Hz and 8Hz. This variable frequency is directly associated with the rate at which motion profiles are regenerated by the motion planner (Section 2.3.2.2). With this variable frequency in place, each motion profile is only valid for a maximum time of 500ms.
The 500ms validity limit placed on the motion profiles actually provides a major advantage to the ANVID software. Recalling from Section 2.3.2.2.3, each motion profile typically contains 2-3s of future commanded steering and speed actions that describe the trajectory planned by the TORC AutonoNav™ system. If a new motion profile was sent only after the completion of the previous motion profile, the ANVID MPCr would be forced to calculate 2-3s of suboptimal reference steering and speed signals. This, in turn, would require driver models that could predict steering and speed actions 2-3s into the future. Since the inner driver loop operates at 40Hz, this timeframe would translate to the prediction of up to 120 samples from each of the steering and speed models. Predicting such a large horizon of data requires a significantly more complex neural network and dramatically increases the associated network training time. The complication is doubled by the fact that two separate models are used in parallel. However, the 500ms validity limit enforced by the motion profile regeneration rate requires that only 20 data points be predicted by each of the steering and speed models. For this reason, relatively simple TSPNNs can be used to implement the steering and speed models to enable prediction and online training in real time.

Although only a horizon of 500ms of predicted steering and speed actions is required, the TSPNN models utilized in this implementation actually operate with prediction horizons that are 750ms long. The reason for the increased prediction horizon is to enable the MPCr to account for small extraneous delays that may occur in the generation or communication of motion profiles by the TORC AutonoNav™ system. Theoretically, a new motion profile should be received at least every 500ms. However, unforeseen variations in the actual implementation can cause this period to exceed past 500ms. By utilizing a 750ms prediction horizon, the MPCr ensures that the driver is always being supplied with suboptimal reference signals even if there are momentary lapses in the regeneration of the motion profiles. Therefore, the outputs of the steering and speed TSPNN models contain 30 data points representing the predicted actions of the driver over a 750ms long horizon.

As discussed in the previous subsection, the inputs to the TSPNN steering and speed models must contain past actual values and future reference values for each respective driver action. TSPNN models utilize a specific “memory depth” that describes how much past data is utilized as an input to the prediction model. The memory depth is typically determined empirically based on prediction performance and was chosen as 750ms for both the steering and speed models in this particular implementation. Thus, each model requires the 30 past data values recorded at 40Hz from the actual driver steering and speed responses. It is important to note that the memory of this data is recorded from actual driver responses and is in no way related to the predicted responses of the driver models. The memory of this data is what enables TSPNN models to maintain the relationship between the past actual and future predicted steering and speed states within the dynamical driver system.
The reference values that are supplied as inputs to the TSPNN models are actually defined by the parent MPC process. In this implementation, the MPCr calculates equal-sized horizons of suboptimal future reference data and predicted response data due to the dependency of the response on the reference. As discussed in the previous paragraphs, the model only utilizes a 750ms-long horizon of prediction data due to the validity limit of the motion profiles. Because the reference and predicted response horizons are equal size, the MPCr must only calculate a likewise 750ms-long horizon of suboptimal future reference data. However, the frequency of the reference data is not enforced by any component within the Blind Driver Challenge® system and is thus fully customizable by the designer. In the ANVID implementation, a reference signal data frequency of 8Hz was chosen. This particular rate was chosen to match the minimum horizon size of validity-limited motion profile, simplify the MPCr optimization process, and generate smoother suboptimal reference signals. Thus, the MPCr must only calculate 6 data points for each of the suboptimal steering and speed reference signals at each iteration. The relatively low frequency of the reference signals remains valid due to the fact that the driver cannot track high-frequency reference signals with satisfactory results (Sections 3.1.4 and 3.2.4).

The comprehensive inputs and outputs of the TSPNN models for driver steering and speed prediction can finally be summarized and presented in unison. A breakdown of the inputs and outputs for both models is shown in Figure 5-9 below. As previously discussed, the 30 past value data points are recorded directly from actual driver steering and speed actions and not from prediction data. The inputs for the 6 future reference values and 30 predicted response values can also be seen in this figure. In total, the TSPNNs for each of the steering and speed models input 36 discrete values and output 30 discrete values. This particular input and output configuration is additionally displayed in relation the neural networks of the separate TSPNN models in Figure 5-10.

![Figure 5-9: Input/Output definition for the driver steering and speed TSPNN prediction models.](image-url)
Figure 5-10: Input and output diagrams for both steering and speed TSPNN driver models. The inner loop (40Hz) iteration count is signified by (i) while the reference (8Hz) iteration count is signified by (j).

Finally, the theoretical definition of the TSPNN driver steering and speed models can be concluded with a discussion on the particular neural network structure utilized within each model. Recalling from the previous subsection, a scientific method for configuring a neural network structure and its internal neurons currently does not exist. Empirical means are therefore used to determine the amount of layers, neurons per layer, and neuron configuration. In this implementation, a single network configuration was empirically determined that was able to produce satisfactory prediction results from both the steering and speed TSPNN driver models (as will be proven in Section 5.5.3). Although the internal dynamics of the driver’s steering and speed actions are most certainly different, the black box nature of neural networks allows a certain degree of flexibility in terms of network configuration. It is this black box nature of the neural network that provides the advantage of rapid and accurate modeling through empirical determination. The designer simply has to determine a network configuration that produces satisfactory results rather than attempt to derive the complex dynamics of the systems being identified.

The particular multi-layer perception (MLP) network configuration determined for the steering and speed TSPNN driver models is pictured in Figure 5-11. The size of the networks’ input and output layers are predetermined by the TSPNN input/output definition in Figure 5-9. It can be seen that the input layer contains 36 inputs: 30 inputs for the past 750ms (@40Hz) of actual driver data (or memory) and 6 inputs for the future 750ms (@8Hz) of future reference signal data. The output layer contains 30 neurons: one neuron to calculate each of the 30 predicted driver responses over the next 750ms (@40Hz). Two hidden layers with 36 neurons each are also included to increase the ability of the network to model the complex dynamical behavior of the
driver steering and speed actions. The total number of neurons within each network is thus 102 neurons, all of which utilize no input bias. The hidden layers contain as many neurons as there are inputs in an effort to conserve and use all of the data provided by the inputs in the final calculations of the predicted driver responses at the output layer. It does not seem valid to compress the data from the 36 different inputs into a smaller set of data through a hidden layer with less than 36 neurons. The smaller hidden layer would effectively combine the 36 inputs into a smaller number of inputs for subsequent layers and could possibly lose important information as the method of combination is not provided to the subsequent layer. While this method yielded a satisfactory network configuration for this implementation, further research could be conducted to examine the effects on model performance of the hidden layer size in relation to the input layer.

Figure 5-11: Multi-Layer Perceptron (MLP) network configuration for the steering and speed TSPNN driver models. The particular network is shown for the driver speed model; the driver steering model can be found by replacing S with θ.

The neurons within the two hidden layers utilize hyperbolic tangent sigmoid activation functions at the neuron output. The output of the hidden layer neurons is thus defined by:

\[ y_N = \tanh \left( \sum_{i=1}^{I} w_i u_{N,i} \right) \]  

(5.1)
where $y_N$ is the neuron output, $u_N$ is a particular neuron input, $w$ is a weight associated with a particular neuron input, $i$ is the index of a particular neuron input and associated weight, and $I$ is the total amount of neuron inputs. The hyperbolic tangent sigmoid was chosen as an activation function because its nonlinearity directly enables the neural network to model nonlinear systems. Without a nonlinear activation function, neural networks would be completely linear and unable to model nonlinear systems such as a human driver. Aside from nonlinearity, a continuous activation function is also required so that the derivative of the activation function can be calculated. Activation function derivatives are needed for model training and also for gradient calculations later in the MPCr optimization process. The hyperbolic tangent sigmoid was chosen as an activation function rather than the popular logistic function because it operates over a range of ±1. The logistic function only operates between 0 and 1 and is typically used for classification-type neural networks that threshold the strictly positive neural output. The hyperbolic sigmoid allows neural outputs that are both positive and negative, thus making it more suitable for function approximation and system modeling. A plot of the hyperbolic sigmoid activation function is provided in Figure 5-12 to demonstrate the function’s nonlinearity, continuity, and output range.

![Figure 5-12: The hyperbolic tangent sigmoid activation function.](image)

Figure 5-12 demonstrates that the hyperbolic sigmoid output converges to ±1 for larger positive and negative values. According to (5.1), the output of a neuron using this particular activation function will therefore saturate once the sum of weight and input products exceeds approximately ±3. The saturation of neural outputs ultimately hinders the network’s ability to model a system because the network outputs are effectively clamped at a certain size. Therefore, it is very important to ensure that the sum of weight and input products is approximately within the ±3 range so that output saturation is avoided. This can be achieved by choosing an appropriate scaling factor at the input and output layers of the neural network. The scaling factor downscales the full input and output ranges of the modeled system so that the neural network can fully operate between the saturation ranges.
The scaling factors for the TSPNN driver models in this implementation were chosen empirically to maximize the prediction performance of the neural networks. In this process, the first step was determining the full operating ranges of the driver’s steering and speed actions. According to Section 2.3.2.1.4, the 2010 Ford Escape Hybrid SUV has a maximum steering wheel angle and thus steering operating range of ±537°. While the Escape does have an electronically limited maximum speed, a speed operating range of ±30m/s (67.1mph) was chosen as the current scope of the Blind Driver Challenge® does not require the driver to travel faster than 40mph. The reason that ±30m/s was chosen rather than 0-30m/s is to maintain neural network prediction performance at values closer to 0m/s. Prediction performance tends to decrease as the operating range limits and thus the saturation ranges are approached. Since the driver speed actions often operate within the proximity of 0m/s, it is important that the accuracy at this range is maintained. Therefore, ±30m/s is chosen to position the typical operating range approximately the same distance from the positive and negative saturation ranges. After determining the operating ranges for the steering and speed TSPNN driver models, a scaling factor had to be chosen that maximized the respective model prediction performances. It was empirically determined that satisfactory prediction performance occurs when both the steering and speed operating ranges are downscaled to ±0.5 at the input layers of the respective neural networks.

The output layer of the steering and speed TSPNN driver models does not utilize activation function and thus its output does not directly suffer from saturation. However, the output is still indirectly affected by saturation because the inputs to the output layer are the outputs of the previous hidden layer, which uses a hyperbolic activation function. Since the hidden layer outputs are limited to a range of ±1, the weights of the output layer would be primarily responsible for creating outputs that operate far outside of the ±1 range. If, for example, the unscaled steering angle operating range was enforced at the output layer, the output layer weights would have to be significantly large to generate values close to the limit of the ±537° range as the inputs are always inclusively within ±1. This would make the network output highly sensitive to the output layer weights compared to the hidden layer weights and hinders the prediction abilities of the network. Therefore, a scaling factor must also be applied at the output layer of the neural network to ensure that the weights from all layers are contributing relatively equally to the network output. Similarly to the input layer, it was empirically determined that downsampling the operating ranges to ±0.5 at the output layer yielded satisfactory prediction results.
5.3.2.3: **ANVID Driver Model Training**

Artificial neural networks are able to serve as prediction model of a dynamical system by learning how to recreate the relationship between the dynamical system’s inputs and outputs. As a black-box identification method, ANN prediction models do not learn the actual dynamics of the system in question. Instead, the ANN simply learns how to recreate the same outputs that the actual dynamical system would exhibit for a particular input. In order to learn how to recreate the correct outputs, the ANN must be properly trained by adjusting the weights associated with each synapse within the ANN. Many different types training methods exist; however the TSPNN driver models utilized by the ANVID software are particularly trained online with supervised learning.

The TSPNN driver models in this implementation use online training to continuously adapt with the human driver in time. As previously discussed, the human driver is not only highly nonlinear and stochastic, but is also time variant. Because the driver is time-variant, his or her dynamics change over time. This means that a driver’s particular response to a particular steering or speed reference signal can be expected to change with time. Such a scenario presents itself as the driver continues to use the Non-Visual Interface System for extended periods of time and improves his or her tracking skills. The driver is able to gain experience using the system and adapt to it. The purpose of using online training for the driver models is to enable the system to conversely adapt to the driver in parallel. If the driver and the Blind Driver Challenge® system adapt to each other at the same time, it can be expected that significant improvements in the combined system performance will occur.

As previously mentioned, online training allows ANN prediction models to continuously adapt to drivers utilizing the Blind Driver Challenge® system. The prediction models’ rate of adaption governs what types of changes in driver dynamics the model can adapt to. If the adaption rate is slow, it will take a considerable amount of time for the ANN prediction model to converge to the current dynamics of the driver. In this case, the model would only be able to adapt to dynamics that change very slowly over time. An example of this would be the improvement of the driver’s tracking skills using the Non-Visual Interface system as more experience is gained. However, a slow adaption rate would not be able to adapt to relatively more sudden and temporary changes in driver dynamics, such as the onset of sleepiness.

Therefore, it is desirable to choose a training process with a fast adaption rate so that the prediction model may adapt more closely to the driver’s constantly changing dynamics. The adaption rate of a training process is defined by two different process characteristics. The first characteristic describes how fast the training process can force the weights to converge to values that minimize the model error. The second characteristic describes how model plasticity is governed within the training process. Thus, a training process must be chosen with a fast convergence rate and adjustable
model plasticity so that a fast adaption rate can be achieved without sacrificing the accuracy of the prediction model. For this reason, the ANVID software utilizes the Multiple Extended Kalman Algorithm (MEKA) training process to permit the system's rapid and accurate adaption to the driver.

5.3.2.3.1: The Multiple Extended Kalman Algorithm (MEKA) Trainer

The MEKA trainer is a supervised feed-forward neural network training algorithm developed by Shah and Palmieri in [135] that is able to achieve network convergence significantly faster than other standard training techniques (such as backpropagation). Essentially, the MEKA trainer is a local approach that updates the network weights with an Extended Kalman Filter (EKF) imposed on each neuron. The process treats each neuron as a nonlinear dynamical system whose state and measurement equations are

\[ w_{i+1} = w_i \]  \hspace{1cm} (5.2)
\[ d_{n,i} = f(w_i^T u_i) + e_{b,i} \]  \hspace{1cm} (5.3)

where \( w \) is the neuron weight vector and considered the neuron state, \( d_n \) is the desired output that the neuron should exhibit and is considered as the measurement, \( f(\cdot) \) is the neuron's nonlinear sigmoidal activation function, and \( e_b \) is the local neuron output error determined from standard backpropagation [136] and is considered as the measurement noise. An EKF can be used on the nonlinear neuron system described in (5.2) and (5.3) to estimate the state vector (and thus the neuron weights) using

\[ \hat{w}_{i+1} = \hat{w}_i + e_{b,i} K_i \]  \hspace{1cm} (5.4)
\[ \hat{P}_{i+1} = \lambda^{-1} \hat{P}_i - K_i z_i^T \]  \hspace{1cm} (5.5)
\[ K_i = z_i [1 + z_i^T u_i]^{-1} \]  \hspace{1cm} (5.6)
\[ z_i = \lambda^{-1} \hat{P}_i q_i \]  \hspace{1cm} (5.7)
\[ q_i = f'(w_i^T u_i) u_i \]  \hspace{1cm} (5.8)

where \( \hat{w} \) is the estimate of the neuron state vector, \( K \) is the Kalman gain, \( q \) is the Jacobian of the measurement with respect to the state variables, \( u \) is the neuron input vector, \( \hat{P} \) is the estimate of the inverse covariance matrix of \( q \), and \( \lambda^{-1} \) is a forgetting factor. Thus, each neuron uses a local EKF to update the weight vector and minimize the backpropagated error over the course of training iterations. This nonlinear localized approach enables the network weights to converge within significantly fewer training iterations when compared to other methods such as standard backpropagation. Additionally, the authors found that the non-convex local performance surfaces at each neuron reduce the chance that the network converges to a local minimum.
Aside from the rapid network convergence rates, the MEKA trainer offers several other main advantages for the ANVID software implementation. The first advantage is that the trainer uses supervised learning to train the network. Supervised learning is typically used for training ANNs to learn an input-output relationship as long as training data already containing sets of correctly matched input-output relationships is available. In this process, the trainer subjects the ANN to a particular input and compares the ANN output to the input’s correctly matched output specified by the supervision. The difference between the ANN output and the correct output, or the ANN error, is found and backpropagated through the network so each neuron’s local EKF can adjust the weights accordingly. This allows the ANN prediction model to be trained using actual input and output data from the dynamical system in question.

A second advantage of the MEKA trainer is that its formulation in (5.4)-(5.8) contains a configurable forgetting factor, $\lambda$. This forgetting factor ultimately allows the plasticity of the model to be controlled. This is a desirable trait because it allows the designer to choose how quickly the network converges and thus how fast the adaptation rate of the training process is. A value of $\lambda$ equal to 1 allows the network to converge at the fastest rate possible and quickly adapt to changes in the modeled system’s dynamics. Values of $\lambda$ less than 1 and that approach 0 decrease the networks convergence rate and the plasticity of the ANN prediction model. If $\lambda$ is set to zero, the plasticity of the model will become completely rigid and thus no changes will be made to the ANN by the trainer.

One disadvantage of the MEKA trainer is that the local approach forces each neuron to maintain a separate $\hat{P}$ matrix, as defined in (5.5). For networks containing a large amount of neurons, the maintenance of the $\hat{P}$ matrices can consume significant amounts of memory in an actual computer implementation. The overall computational cost also increases due to the linear operations that are carried out on the high amount of large $\hat{P}$ matrices. Even though the MEKA trainer takes less training iterations to converge, the computational cost associated with each iteration may actually lead to longer training times than other training methods. Thus, the tradeoff of utilizing the MEKA trainer is that it can train networks with rapid convergence; however the size of the network is limited by the desired length of training time and the specifications of the implementation computer.

5.3.2.3.2: Considerations for Online Supervised Learning

It was described earlier that the TSPNN driver models utilized by the ANVID require a training method that enables the prediction models to continuously adapt to the driver’s dynamics over time. These capabilities require a training method that has a fast convergence rate, can be implemented online, and that requires low training time for real-time adaptation. The MEKA training algorithm discussed in the previous subsection solves the first requirement and yields fast convergence rates with its local EKF approach. However, the authors of [135] do not discuss any way in which the
MEKA can be used to solve the final two requirements. Therefore, the ANVID software utilizes a custom implementation of the MEKA training method so that it can be operated online and in real-time.

Let the discussion begin with the definition of the online MEKA implementation. As discussed earlier in Section 5.3.2.3, online training enables the ANN prediction model to adapt with the time-variant driver dynamics. Online training is able to perform this function because the model is trained simultaneously and indefinitely with the operation of the dynamical system in question. Outputs from the dynamical system are passed in real-time to the training algorithm so that prediction model can be trained sample-by-sample. This is true the definition of online learning [131]. Training methods that utilize online learning are highly advantageous in that they continuously observe the dynamical system and update its ANN prediction model in parallel. This type of method directly provides the ANN prediction model’s capability to adapt in time with the time-variant dynamical system.

Section 5.3.2.3.1 defines that the MEKA trainer utilizes EKFs to locally update the weight vectors for each neuron in the network. Since the EKF is recursive in nature, the EKFs used in the MEKA trainer are able to update the weight vectors using information from only the current and previous training iterations. Furthermore, the only new information the EKFs need for each weight vector update is a single sample of the backpropagated error (5.4). The other information from the current and previous training iterations is only used for the persistence of the weight vectors and inverse covariance matrices between training iterations. Since the only “external” requirement of the localized EKFs is the current sample of the backpropagated error, the MEKA method can be operated on a sample-by-sample basis, otherwise known as online [131].

An important factor to consider when applying the MEKA as a supervised online training method in this implementation is that the TSPNN driver models each contain multiple outputs that construct a signal. In order to implement sample-by-sample, or online, supervised training, the MEKA must maintain a buffer of the previous inputs and outputs of the dynamical system being modeled. The buffers are equal to the size of the input and output layers of the neural network and enable the persistence of past input/output data for use in supervised network training. Each time the dynamical system is subjected to a new input and responds with an output, the new input and output are added to the buffer while the oldest is discarded. This enables the trainer to maintain a memory of the past inputs and outputs of the dynamical system that can be used as supervisory training data. Each time the trainer updates the weight vectors, it uses the input and output buffers as a single sample.
An example of the sample-by-sample buffering process is shown in Figure 5-13. Each time a new discrete input and output are recorded from the dynamical system, the pair is added to the input and output buffers. The MEKA trainer then uses these buffers as a single sample to determine the backpropagated error to each neuron. The local EKFs at each neuron then perform a single weight vector update using (5.4)-(5.8). Simultaneous to the weight update, the dynamical system is still operating and responding to new inputs with new outputs. These new inputs and outputs are continuously added to the buffers while the training still occurs with the previous sample. Once the training iteration using the first sample is complete, the MEKA trainer copies the current state of the buffers and utilizes the information as a new sample for the next training iteration. Utilizing the buffers as samples allows the training to be performed online and does not require that the training iterations operate at the same rate as the measurements of the dynamical system’s inputs and outputs. Thus, data may be measured from the dynamical system at 40Hz while the training only occurs at 20Hz.

**Figure 5-13: Online training for multi-output ANNs using buffers.**

The ANVID MEKA trainer implements this method of online learning to train the TSPNN driver models. Before the details of this implementation can be defined, however, the method of supervisory training must be examined. Recalling from Section 5.3.2.2, the TSPNN driver models output a finite horizon of predicted driver actions. Thus, the outputs of these ANNs are future data that is not yet known. However, the MEKA trainer is a supervised learning process and requires knowledge of the correct ANN output to properly train the network. The only way to have knowledge of future driver actions for training the TSPNN driver models is to create a virtual timespan offset by the length of future time data required.

An example of this virtual shift in time is shown in Figure 5-14. The first timeframe depicts the normal timeframe of a TSPNN prediction model. Recalling from Section 5.3.2.1, a TSPNN uses knowledge of past outputs to help predict future outputs. Thus,
the standard TSPNN uses a $\tau_P$ long horizon of previous outputs to predict a $\tau_F$ long horizon of future outputs. In order to use supervised learning, however, the training algorithm must virtually shift the normal TSPNN timeframe backwards with an offset of $\tau_F$. With this virtual shift in place, the inputs to the TSPNN prediction model are taken from $-(\tau_F+\tau_P)$ to -$\tau_F$ of the past dynamical system output data. When subjected to this particular timeframe of inputs, the TSPNN outputs a prediction of data in a timeframe that has already occurred. Thus, the TSPNN is shifted backwards in time and essentially predicts data that has already happened. Since the prediction timeframe is shifted into the past, the actual outputs of the dynamical system have already been measured and can be used as supervisory training data. This time-shifted supervisory training process can be conducted as long as a $(\tau_F + \tau_P)$ long horizon of past dynamical system output data is stored in memory.

![Diagram of time-shifted supervised learning for TSPNN prediction models.]

**Figure 5-14: Time-shifted supervised learning for TSPNN prediction models.**

### 5.3.2.3.3: ANVID MEKA Definition

At this point, the training method and considerations for online supervised training have been defined and discussed for the ANVID driver modeling process. This information will now be used to present the implementation of the MEKA training algorithm within the ANVID software. Recalling from Section 5.3.2.2, the steering and speed TSPNN driver models utilize past driver actions and the future reference signal to predict future driver actions. The past driver actions contain 30 data points over the past 750ms, while the future reference signal contains 6 data points over the next 750ms. The TSPNN models use these inputs to predict the driver’s future actions in the form of 30 data points over the next 750ms. In order to perform online supervised training, the methods discussed in the previous subsections must actually be implemented.
The first step of implementing the MEKA trainer for online supervised learning is to define the buffers that will store the TSPNN’s supervised training data samples. Section 5.3.2.3.2 discussed that these buffers will be used to permit online training in parallel with the dynamical system’s operation. Thus, the ANVID MEKA implementation will use buffers so that the TSPNN driver models may be trained in real-time while the driver is navigating the vehicle. Under normal operation, these buffers would only store the past 30 steering wheel angles and speed exhibited by the driver. However, training the driver models with supervised learning requires the application of a virtual time shift to TSPNN driver models. Section 5.3.2.3.2 indicated that the backwards time-shift must be equal to the length of the model’s prediction time horizon. For both the steering and speed TSPNN driver models, the prediction horizon and thus the virtual backwards time-shift is 750ms. Thus, the buffers must be expanded to hold the 750ms of time-shifted data. Since the driver actions are recorded at 40Hz, the driver action buffers must be expanded by 30 slots. This brings the total size of each driver action buffer to 60 slots.

Before the virtual time-shift was applied, no buffer was required to store the future reference signal data points because they do not exist in the past. However, the time-shift also shifts the reference signal data points into a past time-frame. Thus, a new buffer must be created to store this information. Since the reference data points are generated at 8Hz, the buffer must contain 6 slots to hold the past 750ms of reference data points.

The final configuration of the supervisory sample buffers for the online MEKA training implementation is shown in Figure 5-15. This configuration applies to both the steering and speed TSPNN driver models as they share the same neural structure. This configuration is actually very similar to the normal TSPNN driver model operation configuration shown previously in Figure 5-9. The only difference is that this figure depicts the configuration of the sample data that is used to train the TSPNN and does not contain actual data from the TSPNN itself. Each type of data is stored in one of the discussed buffers and is called upon during each MEKA training iteration.

![Figure 5-15: Sample buffers utilized for ANVID driver model training.](image-url)
A particular advantage of this configuration is that only one buffer is required to store both the past driver actions and the desired outputs for the predicted driver actions. Since the buffer is FIFO enforced, each new measurement of the driver’s action shifts the oldest measurement out of the buffer. Since the separation of the past driver actions and the desired prediction outputs are only separated by a time convention, both are found in the 60-slot driver action FIFO buffer. This one buffer can be visualized in Figure 5-15 as the connection of the blue and red rectangles. The green rectangles represent the 6-slot reference signal buffer.

Each training iteration of the MEKA method grabs an instantaneous copy of the data shown in Figure 5-15 as a single supervisory training sample. The first half of the driver actions buffer, shown in blue, contains the past driver actions and is passed as an input to the TSPNN driver model. The data from the reference signal buffer, shown in green, contains the “future” reference signal points that the driver “will” track and is also passed as an input to the TSPNN driver model. Using these inputs, the TSPNN can then calculate and output the predicted driver response for the “next” 750ms (30 data points). However, the prediction is actually predicting driver actions that have already occurred. These actual actions that have already occurred are stored in the second half of the driver actions buffer, shown in red. The MEKA trainer considers this data as the desired output of the TSPNN driver model and utilizes it to correct the TSPNN’s internal weights with the localized EKFs (5.4)-(5.8).

One final consideration worth noting in the theoretical definition of the ANVID MEKA implementation is that the localized EKF processes of each TSPNN layer can be completed in parallel. During the training method, the total error of the TSPNN output is backpropagated through each layer of the network. Once the backpropagated errors have been calculated, the EKFs of each neuron in the current layer implement (5.4)-(5.8) to update the weight vectors for that particular training iteration. Because the EKFs are completely local to the neuron, the EKF calculations for each neuron may be performed simultaneously in parallel. If the MEKA trainer is implemented on a computational platform containing actual multi-core processing technology, the training time for each layer may be reduced by a factor of the processing core count. For TSPNNs with several layers and many neurons, such as the ANVID driver models, the overall training time may be significantly reduced and thus increases the capable adaptation rate of the MEKA training method.
5.3.3: Reference Optimization

The definition of MPC given in Section 5.3.1 states that MPC seeks suboptimal control inputs to a dynamical system that are determined by predicting the system’s output. In the particular case of the ANVID, the dynamical system is the driver and the control inputs are synonymous with the reference steering and speed signals that the driver attempts to track with the Non-Visual Interface System. To determine suboptimal steering and speed signals, the ANVID MPCr uses a Quasi-Newton Optimization (QNO) process supplemented with the driver models defined in Section 5.3.2.2. This subsection provides a theoretical definition of the QNO implemented by the ANVID and describes its advantages.

One important consideration that must be taken is that the driver model defined in Section 5.3.2.2 is actually split into two identically structured models; one for steering and one for speed. As a result of the separate modeling, the ANVID MPCr was actually divided into a steering MPCr and speed MPCr in the interest of separately tunable controllers. Each MPCr utilizes a separate QNO process to determine its associated reference signal. Similarly to the steering and speed driver models in Section 5.3.2.2, the QNOs for each MPCr are identically structured and operated. Thus, the theoretical definition of the QNO in this section will be in reference to a more generic term of “action” rather than “steering” or “speed” as these two actions share an identical QNO formulation.

5.3.3.1: Summary of the Quasi-Newton Optimization (QNO) Process

A full definition of the Quasi-Newton Optimization process can be found in [137]; however its application to the ANVID and its advantages will be provided here. Generally speaking, the QNO process finds a suboptimal value by sufficiently minimizing an objective function dependent on the value being optimized. The objective function serves as a cost function and can be tailored by the designer for each particular QNO implementation. In the ANVID implementation, the QNO seeks to find steering and speed reference signals that will “trick” the driver into recreating the commanded steering and speed actions of the current motion profile. Thus, the objective function of the ANVID QNO is defined as

\[ O(r) = \frac{1}{30} \sum_{i=1}^{30} (c[i] - \hat{a}[i])^2 \]

where \( O(\cdot) \) is the objective function, \( r \) is the reference action signal, \( c \) is the commanded actions from the current motion profile, \( \hat{a} \) is the predicted actions of the driver found through the associated driver model (Section 5.3.2.2), \( M(\cdot) \) is a function
representing the associated TSPNN driver action model and returns a vector, \( \alpha \) is the past driver actions, and \( i \) is an array/vector indexing factor. (5.9) is simply the mean squared error (MSE) between the commanded actions of the motion profile and the predicted actions of the driver. Recalling from Section 5.3.2.2, the motion profile and its internal commanded actions have a validity limit of 500ms. It was also discussed that the reference signals generated by the MPCr would contain 6 data points over a range of 750ms. Thus, \( r \) is a 6x1 vector containing a candidate set of 6 data points that create a 750ms reference signal. \( \hat{a} \) is a 30x1 vector containing the output of the associated driver action model, which is defined in Section 5.3.2.2 as 30 data points that predict the driver’s actions over a 750ms future horizon. Since the predicted driver actions are compared point-by-point with the commanded actions from the motion profile, the QNO process must interpolate 30 data points at 40Hz from the motion profile’s encoded commands (Section 2.3.2.2.3).

The objective function defined in (5.9) is essentially a cost function that decreases in cost as the driver’s predicted actions more closely mimic the commanded actions from the TORC AutonoNav™ motion profiles. The QNO seeks to minimize this cost by finding reference action signals that are predicted to elicit driver response actions that are the closest to the commanded actions. Thus, the QNO calculates reference action signals that “trick” the driver into accurately recreating the commanded actions as exemplified previously in Figure 5-4. The QNO is able to perform this function by minimizing the objective function (5.9) using the Quasi-Newton Method (QNM).

The QNM is a gradient descent-based function-minimizing algorithm that iteratively finds stationary points using first and second derivatives of the function surface. The use of second derivatives allows the descent towards a stationary point to take a more direct path along the function surface and converge quickly in comparison to algorithms using only first derivatives. A particular advantage of the QNM in comparison to Newton’s Method is that the inverse of the Hessian is approximated and does not need to be explicitly calculated. The approximation of the inverse Hessian saves a considerable amount of processing time and enables faster convergence of the QNM.

The QNO is able to use the QNM to iteratively find which value of \( r \) minimizes the objective function defined by (5.9). Thus, the formulation of the QNO is defined by

\[
\Delta r_k = -s_k \hat{H}_k^{-1} \nabla O(r_k)
\]

\[
r_{k+1} = r_k + \Delta r_k
\]

where \( k \) is the iteration index, \( \Delta r \) is the increment, \( s \) is the step size determined in Section 5.3.3.3, \( \hat{H} \) is the estimated Hessian matrix of the objective function, and \( \nabla O(\cdot) \) is the gradient of the objective function evaluated at \( r \). The estimation of the Hessian matrix is iteratively calculated using the Davidon-Fletcher-Powell (DFP) formula [138]:
where $G$ is the forward gradient difference between the iterations. In a single optimization task, the QNO rapidly iterates through (5.11)-(5.14) to calculate new candidates for the reference action signal ($r_k$) until (5.9) falls below a specified threshold. This threshold represents the acceptable MSE between driver’s predicted actions and the commanded actions of the motion profile. Once this threshold has been reached, the optimizer has successfully found a suboptimal reference action signal.

It is important to recall from Figure 5-8 that the steering and speed MPCrs are called each time a new motion profile is received from the TORC AutonoNav™ system. When each MPCr is called, it must determine a suboptimal set of reference actions as soon as possible so that the driver may subsequently start tracking the suboptimal reference signals. Any delay imposed by the MPCrs is pure time delay injected into the Blind Driver Challenge® system. Since each MPCr is essentially a QNO, the QNO implemented by the ANVID must return a suboptimal reference action signal as fast as possible. However, the QNO is an iterative process that can take relatively large amounts of time to converge if not implemented efficiently. Furthermore, the ANVID must operate a separate QNO for steering and speed. Therefore, the ANVID must implement the QNOs as efficiently as possible to minimize the pure delay between the receipt of a new motion profile and the calculation of an associated set of suboptimal reference steering and speed signals. The theoretical considerations taken to increase the efficiency and robustness of the ANVID QNO implementation will be discussed in the following subsections.

5.3.3.2: Objective Function Gradient Calculation

The gradient of the objective function is used throughout several calculations (5.11)-(5.13) within a single QNO iteration. Observation of the objective function itself in (5.9) shows that calculating its gradient is highly complex due to objective function’s utilization of the TSPNN driver action model. It is possible to avoid this complex calculation by estimating the gradient using a forward difference of the objective function evaluated at small perturbations from the current $r$ value:

$$
\nabla O(r_k) = \begin{bmatrix}
\frac{\partial O(r_k)}{\partial r[1]_k} \\
\frac{\partial O(r_k)}{\partial r[2]_k} \\
\vdots \\
\frac{\partial O(r_k)}{\partial r[6]_k}
\end{bmatrix}
$$

$$
\nabla O(r_k) \approx \begin{bmatrix}
\frac{O(r_k + \epsilon_1) - O(r_k)}{\epsilon_1} \\
\frac{O(r_k + \epsilon_2) - O(r_k)}{\epsilon_2} \\
\vdots \\
\frac{O(r_k + \epsilon_6) - O(r_k)}{\epsilon_6}
\end{bmatrix}
$$
where $\epsilon_n$ is a column vector of zeros except for a small perturbation, $\epsilon$, located at the $n^{th}$ index. This method enables the gradient of the objective function to be estimated with simple evaluations of the performance function itself. However, the method requires seven evaluations of the objective function each time the gradient needs to be calculated. Recalling from (5.9), each evaluation of the objective function requires an evaluation of the neural network. Thus, one gradient estimation must evaluate the neural network driver action model seven times. Furthermore, each iteration of the QNO method requires two different gradient estimations (5.13) and brings the total neural network evaluations to fourteen times per QNO iteration. Even furthermore, the QNO may require hundreds of iterations before a suboptimal reference signal is found. Utilizing the gradient estimation method in (5.16) could therefore require thousands of neural network evaluations to determine a single suboptimal reference signal. The TSPNN driver models defined in Section 5.3.2.2 are relatively complex and, if evaluated for a significant amount of times in a row, would require an equally significant amount of processing time. The complexity additionally doubles because two separate QNOs are called simultaneously with each MPCr iteration. It was indicated in the previous subsection that the QNOs must be able to determine suboptimal reference steering and speed signals as quickly as possible to maximize the performance of the system. Therefore, estimating the gradient with (5.16) is not a desirable method to utilize within the ANVID QNO.

To maximize the efficiency of the ANVID QNO implementation, the gradient of the objective function must be explicitly calculated. Although this calculation is complex, it requires incredibly less amounts of processing time than the estimation method in (5.16). The formulation of the objective function gradient begins with

$$\nabla O(r_k) = \frac{\partial O(r_k)}{\partial r_k}$$

(5.17)

Using the chain rule, the partial derivative of the objective function with respect to the vector $r$ becomes

$$\frac{\partial O(r_k)}{\partial r_k} = \frac{\partial O(r_k)}{\partial M(r,a)} \frac{\partial M(r,a)}{\partial r_k}$$

(5.18)

The first partial derivative of the chain in (5.18) is taken with respect to a vector. Thus the partial derivative is given by

$$\frac{\partial O(r_k)}{\partial M(r,a)} = \left[ \frac{\partial O(r_k)}{\partial M(r,a)[1]} \frac{\partial O(r_k)}{\partial M(r,a)[2]} \ldots \frac{\partial O(r_k)}{\partial M(r,a)[30]} \right]$$

(5.19)
Thus, the partial derivative of the objective function with respect to each output of the neural network model must be found. To make this process simpler, the objective function can be redefined by combining (5.9) with (5.10) to get

$$o(r) = \frac{1}{30} \sum_{i=1}^{30} (c[i] - M(r, a)[i])^2$$  \hspace{1cm} (5.20)

The partial derivative of the objective function with respect to the \(i\)th element of the model output vector is calculated from (5.20) and is defined by

$$\frac{\partial o(r_k)}{\partial M(r, a)[i]} = \frac{1}{15} (M(r, a)[i] - c[i])$$  \hspace{1cm} (5.21)

(5.19) can now be rewritten using (5.21) as

$$\frac{\partial o(r_k)}{\partial M(r, a)} = \frac{1}{15} [(M(r, a)[1] - c[1]) \ldots (M(r, a)[30] - c[30])]$$  \hspace{1cm} (5.22)

The second partial derivative of the chain in (5.18) must now be solved. This particular partial derivative is taken of a vector with respect to another vector and will thus yield the Jacobian matrix

$$\frac{\partial M(r, a)}{\partial r_k} = \begin{bmatrix}
\frac{\partial M(r, a)[1]}{\partial r_1[k]} & \frac{\partial M(r, a)[1]}{\partial r_2[k]} & \cdots & \frac{\partial M(r, a)[1]}{\partial r_6[k]} \\
\frac{\partial M(r, a)[2]}{\partial r_1[k]} & \frac{\partial M(r, a)[2]}{\partial r_2[k]} & \cdots & \frac{\partial M(r, a)[2]}{\partial r_6[k]} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial M(r, a)[30]}{\partial r_1[k]} & \frac{\partial M(r, a)[30]}{\partial r_2[k]} & \cdots & \frac{\partial M(r, a)[30]}{\partial r_6[k]}
\end{bmatrix}$$  \hspace{1cm} (5.23)

Each element of the Jacobian matrix in (5.23) is the partial derivative of one output of the neural network driver model to one input. The calculation of these partial derivatives is highly complex due to the interconnected structure of the neurons within the neural network driver model. The multivariable chain rule can solve these partial derivatives; however, the chains require an additional level of nesting with each layer in the neural network. Performing these calculations by hand is quite tedious and error prone. Additionally, if the designer needed to change the neural network structure even by one neuron, the entire calculation would have to be redone. Fortunately, an iterative algorithm can be used to greatly simplify the process by calculating the partial derivatives step-by-step. Consider the simple three layer neural network shown in Figure 5-16.
The output of each neuron is labeled as the capital of the respective neuron label.

The Jacobian of this particular network is given by

$$J_{network} = \begin{bmatrix}
\frac{\partial G}{\partial x} & \frac{\partial G}{\partial y} & \frac{\partial G}{\partial z} \\
\frac{\partial H}{\partial x} & \frac{\partial H}{\partial y} & \frac{\partial H}{\partial z} \\
\frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} & \frac{\partial I}{\partial z}
\end{bmatrix} \tag{5.24}$$

The first element of the Jacobian matrix can be found using the multivariable chain rule through the layers and neurons of the neural network.

$$\frac{\partial G}{\partial x} = \frac{\partial G}{\partial D} \frac{\partial D}{\partial x} + \frac{\partial G}{\partial E} \frac{\partial E}{\partial x} + \frac{\partial G}{\partial F} \frac{\partial F}{\partial x} \tag{5.25}$$

The second partial derivative in each of these terms must be solved using the multivariable chain rule once more. Performing the multivariable chain rule on just the first element of (5.25) gives

$$\frac{\partial D}{\partial x} = \frac{\partial D}{\partial A} \frac{\partial A}{\partial x} + \frac{\partial D}{\partial B} \frac{\partial B}{\partial x} + \frac{\partial D}{\partial C} \frac{\partial C}{\partial x} \tag{5.26}$$

At this point, the multivariable chain rule is no longer required as the second partial derivative in each term can be solved directly. This type of calculation must be performed for each term in (5.25). Furthermore, the calculation in (5.25) must be calculated for each partial derivative in (5.24). It can be seen that the nesting of the multivariable chain rule for each partial derivative in (5.24) makes the calculation very complex and error prone. Each layer requires a deeper nested level of multivariable chain rule calculations that must be performed for each neuron in that level. The TSPNN driver models defined in Section 5.3.2.2 contain 3 layers with up to 36 neurons in each layer. This makes calculating the Jacobian in (5.24) significantly complex if performed manually. Fortunately, the calculation can be performed iteratively to solve the partial derivatives layer by layer.
Let $\tilde{J}_L$ represent the Jacobian of a layer $L$ with respect to the layer’s inputs. This term will be defined as the local Jacobian of the layer. Let $J_L$ represent the Jacobian of a layer $L$ with respect to the network’s inputs. This term will be defined as the global Jacobian of the layer. The local Jacobian of the first layer in Figure 5.16 is given by

$$\tilde{J}_{L1} = \begin{bmatrix} \frac{\partial A}{\partial x} & \frac{\partial A}{\partial y} & \frac{\partial A}{\partial z} \\ \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} & \frac{\partial B}{\partial z} \\ \frac{\partial C}{\partial x} & \frac{\partial C}{\partial y} & \frac{\partial C}{\partial z} \end{bmatrix} = J_{L1} \tag{5.27}$$

It is important to note that the first layer is a special case in that the local Jacobian is the same as the global Jacobian. This occurs because the first layer inputs are also the network inputs. The local Jacobian of the second layer is given by

$$\tilde{J}_{L2} = \begin{bmatrix} \frac{\partial D}{\partial A} & \frac{\partial D}{\partial B} & \frac{\partial D}{\partial C} \\ \frac{\partial E}{\partial A} & \frac{\partial E}{\partial B} & \frac{\partial E}{\partial C} \\ \frac{\partial F}{\partial A} & \frac{\partial F}{\partial B} & \frac{\partial F}{\partial C} \end{bmatrix} \tag{5.28}$$

The multivariable chain rule across the first layer can now be found by multiplying the second layer local Jacobian by the first layer global Jacobian. The matrix resulting from this multiplication will be the global Jacobian of the second layer. As an example, the first element of the global Jacobian matrix resulting from the multiplication of (5.28) by (5.27) is found with

$$\frac{\partial D}{\partial A} \frac{\partial A}{\partial x} + \frac{\partial D}{\partial B} \frac{\partial B}{\partial x} + \frac{\partial D}{\partial C} \frac{\partial C}{\partial x} = \frac{\partial D}{\partial x} \tag{5.29}$$

Thus, the global Jacobian of the second layer can be found with

$$\tilde{J}_{L2}J_{L1} = \begin{bmatrix} \frac{\partial D}{\partial x} & \frac{\partial D}{\partial y} & \frac{\partial D}{\partial z} \\ \frac{\partial E}{\partial x} & \frac{\partial E}{\partial y} & \frac{\partial E}{\partial z} \\ \frac{\partial F}{\partial x} & \frac{\partial F}{\partial y} & \frac{\partial F}{\partial z} \end{bmatrix} = J_{L2} \tag{5.30}$$
This process will continue with the third layer. The local Jacobian of the third layer is given by

\[
\tilde{J}_{L3} = \begin{bmatrix}
\frac{\partial G}{\partial D} & \frac{\partial G}{\partial E} & \frac{\partial G}{\partial F} \\
\frac{\partial H}{\partial D} & \frac{\partial H}{\partial E} & \frac{\partial H}{\partial F} \\
\frac{\partial I}{\partial D} & \frac{\partial I}{\partial E} & \frac{\partial I}{\partial F}
\end{bmatrix}
\] (5.31)

The global Jacobian of the third layer can be found in the same manner as (5.30) by multiplying the local Jacobian of the third layer by the global Jacobian of the second layer. An example calculation for the first element of the resulting Jacobian matrix is given by

\[
\frac{\partial G}{\partial D} \frac{\partial D}{\partial x} + \frac{\partial G}{\partial E} \frac{\partial E}{\partial x} + \frac{\partial G}{\partial F} \frac{\partial F}{\partial x} = \frac{\partial G}{\partial x}
\] (5.32)

Thus, the global Jacobian of the second layer can be found with

\[
\tilde{J}_{L3}J_{L2} = \begin{bmatrix}
\frac{\partial G}{\partial x} & \frac{\partial G}{\partial y} & \frac{\partial G}{\partial z} \\
\frac{\partial H}{\partial x} & \frac{\partial H}{\partial y} & \frac{\partial H}{\partial z} \\
\frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} & \frac{\partial I}{\partial z}
\end{bmatrix} = J_{\text{network}}
\] (5.33)

Notice how the global Jacobian of each layer is found by calculating the local Jacobian of the layer and multiplying it by the global Jacobian of the previous layer. With this iterative process, the Jacobian matrix of the entire neural network may be found with simple linear operations. The process itself iterates through each layer of the neural network model, starting at the first layer and ending at the output layer. The iterative process for calculating the global Jacobian of the neural network is thus defined by

\[
J_L = \tilde{J}_LJ_{L-1}
\] (5.34)

where \(L\) is the index of the current layer during the iteration process. The first iteration conducted at the first layer has no previous layer, thus the global Jacobian of the first layer is simply equal to the local Jacobian. The iteration process makes calculating the global Jacobian of the neural network a simple step-by-step procedure and only requires the memory to store two matrices at a time.
The iterative process shown in (5.34) requires that the local Jacobian be calculated for each particular layer. The local Jacobian simply contains the partial derivatives of each layer output with respect to each layer input. Consider the second layer of the neural network shown in Figure 5-16. The local Jacobian for this layer has been supplied in (5.28). As an example, the first element of this local Jacobian will be solved. First, the output of the “d” neuron must be defined as a function of the inputs “A”, “B”, and “C”:

\[ D(A, B, C) = f_d(\Sigma_d) \]  
\[ \Sigma_d = w_d A + w_B B + w_C C \]

where \( f_d(\cdot) \) is the activation function of the “d” neuron, \( \Sigma_d \) is the sum of the neuron’s weight and input products, and \( w \) is a weight associated with a particular input. The partial derivative of the neuron output, “D”, with respect to the input “A” is

\[ \frac{\partial D}{\partial A} = f'_d(\Sigma_d)w_A \]  

The remaining elements in the local Jacobian of the second layer are all found in the same manner as (5.37), giving

\[ \bar{J}_{l2} = \begin{bmatrix} f_d'(\Sigma_d)w_A & f_d'(\Sigma_d)w_B & f_d'(\Sigma_d)w_C \\ f_e'(\Sigma_e)w_A & f_e'(\Sigma_e)w_B & f_e'(\Sigma_e)w_C \\ f_f'(\Sigma_f)w_A & f_f'(\Sigma_f)w_B & f_f'(\Sigma_f)w_C \end{bmatrix} \]  

The local Jacobian is simple to calculate and only requires that the neural network has been run once to find the sums of the weight and input products. In summary, the determination of the neural network’s global Jacobian only requires performing one local Jacobian calculation and matrix multiplication per layer in the neural network.

Returning to the gradient calculation of the objective function, the iterative method given in (5.34) can be used to find the Jacobian matrix given in (5.23). However, the global Jacobian of the TSPNN driver model \( M(r, a) \) will include partial derivatives that are not only in respect to the input \( r \), but also in respect to the input \( a \). The global Jacobian of \( M(r, a) \) is defined by

\[
\frac{\partial M(r, a)}{\partial r, a} = \begin{bmatrix}
\frac{\partial M(r, a)[1]}{\partial r[1]_k} & \ldots & \frac{\partial M(r, a)[1]}{\partial a[1]} & \ldots & \frac{\partial M(r, a)[1]}{\partial a[30]} \\
\frac{\partial M(r, a)[1]}{\partial r[2]_k} & \ldots & \frac{\partial M(r, a)[1]}{\partial a[1]} & \ldots & \frac{\partial M(r, a)[1]}{\partial a[30]} \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial M(r, a)[30]}{\partial r[1]_k} & \ldots & \frac{\partial M(r, a)[30]}{\partial a[1]} & \ldots & \frac{\partial M(r, a)[30]}{\partial a[30]}
\end{bmatrix}
\]
The global Jacobian of the driver model $M(r, a)$ can be split into two separate sub-matrices to determine the global Jacobian with respect to only the $r$ input with

$$
\frac{\partial M(r, a)}{\partial r_k, a} = \left[ \frac{\partial M(r, a)}{\partial r_k} \quad \frac{\partial M(r, a)}{a} \right]
$$

(5.40)

Thus, the iterative process defined in (5.34) can be used to find (5.23) as long as the appropriate sub-matrix is taken from (5.39). It can be argued that the sub-matrix step could be taken in the actual global Jacobian calculation to make the calculation more efficient. The iterative process defined in (5.34) calculates the partial derivatives with respect to all network inputs when, in this case, the partial derivatives are only required to be taken with respect to the $r$ inputs. However, testing confirmed that the implementation of (5.34) consumed negligible processing time, making any increases in efficiency also negligible and thus unnecessary.

The definitions of the partial derivatives in (5.18) are now complete and thus the theoretical method for calculating the gradient in (5.17) is fully defined. This process requires significantly less processing time than the process shown in (5.16) as it only requires one evaluation of the neural network rather than thousands. Furthermore, the single evaluation can be shared amongst all of the gradient calculations in a single QNO iteration ($k$) because the reference signal inputs ($r$) are constant inside of a single $k$ iteration. Thus, the ANVID QNO utilizes the gradient calculation method rather than the gradient estimation method due to the significantly increased processing efficiency.

### 5.3.3.3: Step Size Calculation

The QNO method defined in (5.11)-(5.14) iteratively approaches the objective function (5.9) minimum using a line search strategy. This strategy simply finds a descent direction on the objective function surface and chooses a step size $s$ to decide how far $r$ should descend down the surface. In this implementation, the QNO utilizes an inexact, backtracking line search with variable step size to reduce the amount of iterations required for the objective function to reach a minimum threshold. The descent direction is calculated in (5.11) of the QNO, while calculation for the step size is defined in this subsection.

The step size is chosen in a way that it is maximized while satisfying a series of rules that ensure the descent causes the objective function and gradient to sufficiently decrease. The maximum step is chosen to descend as quickly as possible down the objective function surface. The step size maximization itself is conducted through a secondary iterative process nested within each QNO iteration step. The maximization process must require low processing time as a single QNO evaluation may require hundreds of QNO iteration steps.
The iterative process for calculating the maximum step size simply starts with a large step size and continuously decreases it until it satisfies the rules previously mentioned. This process is typically known as a backtracking line search. The step decrement process is defined by

\[ s_{j+1} = \tau s_j \]  
\[ 0 < \tau < 1 \]  

where \( s \) is the step size, \( \tau \) is the decrement factor, and \( j \) is the iteration index for the step size maximization process. Note that \( j \neq k \) as this iteration is nested within a single \( k \) iteration of the QNO process. \( \tau \) is chosen between zero and one to continuously decrease the step size by a certain amount at each iteration. Recall that the step size is initialized as a large value at the beginning of each iteration process.

The iteration process continually decreases \( s \) until at least two of three conditions are met. Each of these conditions will be discussed in the following subsections.

### 5.3.3.3.1: The Armijo Rule

The first condition, called the Armijo Rule [139] must always be met. The Armijo Rule requires that

\[ O(r_k + s_j D_k) \leq O(r_k) + c_1 s_j D_k^T \nabla O(r_k) \]  
\[ 0 < c_1 \ll 0.5 \]  
\[ D_k = -H_k^{-1} \nabla O(r_k) \]  

where \( D \) is the descent direction and \( c_1 \) is a tuning parameter typically close to zero. The Armijo Rule is responsible for ensuring that the step size \( s \) forces the objective function to decrease by a sufficient amount. The sufficiency of this amount is determined by the second term of the right hand side of (5.42) and is thus tunable with \( c_1 \).

### 5.3.3.3.2: The Wolfe Condition

The second condition, called the Wolfe Condition [140], must also always be met. The Wolfe Condition requires that

\[ D_k^T \nabla O(r_k + s_j D_k) \geq c_2 D_k^T \nabla O(r_k) \]  
\[ c_1 \ll c_2 < 1 \]  

where \( c_2 \) is a tuning parameter typically close to one. The Wolfe Condition is a curvature condition that ensures the step size \( s \) forces the gradient of the objective
function to sufficiently decrease. The sufficiency of this amount is determined by the right hand side of (5.44) and is thus tunable with \( c_2 \).

**5.3.3.3: The Strong Wolfe Condition**

The third and final condition, called the Strong Wolfe Condition \([140]\), is not required to be met. This condition is a more conservative version of the Wolfe Condition that forces the step size to lie close to a critical point of

\[
O_s(s) = O(r_k + sD_k)
\]

(5.45)

where \( O_s(s) \) is the objective function as a function of the step size. Thus, \( s \) should be chosen so that it minimizes (5.45) as much as possible. In order to do so, the Strong Wolfe Condition requires that

\[
|D_k^T \nabla O(r_k + s_j D_k)| \leq c_2 |D_k^T \nabla O(r_k)|
\]

(5.46)

\[
c_1 \ll c_2 < 1
\]

An additional advantage of the Strong Wolfe Condition is it ensures that the estimation of the inverse Hessian matrix in (5.14) is always positive definite as long as the initial inverse Hessian is also positive definite.

However, implementation of the Strong Wolfe Condition demonstrated multiple cases where the conditions could not be met and a satisfactory step size could not be found. It is suggested that further research be conducted to discover the cause of this frequent failure. This issue was indirectly solved in the ANVID QNO implementation by attempting to satisfy the Strong Wolfe Condition first and switching to the weaker Wolfe Condition if necessary. The Wolfe Condition is able to always find a suitable step size and only has the disadvantage of mildly decreasing the convergence rate of the QNO method. The Strong Wolfe Condition is used whenever possible to help maximize the efficiency of the QNO. However, if the Strong Wolfe Conditions cannot be satisfied after a specified amount of (5.41) iterations, only the Wolfe Conditions must be satisfied.

It was stated in the previous subsection that the Wolfe Condition must always be satisfied. This is possible because a step size that satisfies the Strong Wolfe Condition also satisfies the weaker Wolfe Condition. If satisfaction of the Strong Wolfe Condition is abandoned due to lack of a solution after a certain amount of iterations, then only the Wolfe Condition must be satisfied. Thus, \( s \) must be chosen to always satisfy the Wolfe Condition.
5.3.3.4: Choice of Initial Guess

The QNO method defined in (5.11)-(5.14) is an iterative process; thus an initial guess for $r_{k=0}$ and $\hat{H}^{-1}_{k=0}$ is required for the process to begin. The initial guess for $\hat{H}^{-1}_{k=0}$ is simply an identity matrix. However, the choice of an initial guess for $r_{k=0}$ is more complicated as it influences the continuity of subsequent QNO solutions in the MPCr.

Recalling from Section 5.3.3, the QNO solves for a new suboptimal set of reference action signals each time a new motion profile is received from the TORC AutonoNav™ system. Since the motion profiles are received at 2-8Hz and each suboptimal reference action signal is 750ms long, the QNO replaces the obsolete suboptimal reference action signals with a new suboptimal set before the old set is fully tracked by the driver. This causes parts of each suboptimal reference signal calculated by the QNO to form a chain in time. This chain is otherwise known as the full reference signal. An example of the chain formations is shown in Figure 5-17.

![Figure 5-17: Discontinuities in chain formations of the full reference signal. Square markers represent the time at which the QNO generated a new suboptimal reference action signal to replace the previous obsolete reference action signal.](image)

The chain formations in Figure 5-17 display how discontinuity occurs in the full reference signal if continuity between QNO solutions is not maintained. These discontinuities propagate into the non-visual stimuli communicated to the driver and cause confusion similar to that of the motion profile sawtooth waves discussed in Section 4.1.1. Thus, the QNO must ensure that new suboptimal reference signals merge with the replaced obsolete reference signals with a certain degree of continuity.

Initial attempts at ensuring this continuity modified the objective function in (5.9) to additionally increase the cost if the candidate reference signal would raise the derivative of the full reference signal. However, the addition of this derivative term in the objective
function further complicates the optimization process and increases the required processing time. To increase the efficiency of the QNO process, a different method was chosen to ensure the continuity of the full reference signals. Instead, the continuity of each full reference signal could be ensured with a particular choice of the initial guess for each \( r \) calculated.

Thus, the initial guess for the steering and speed QNO processes is a weighted average between the commanded action signal of the motion profile and the suboptimal \( r \) found from the previous QNO evaluation. The weight ratio of the average between the past suboptimal \( r \) to the commanded action signal is 10:1. Thus, the initial guess is very close to the previous QNO solution in comparison to the commanded action signal. This choice of an initial guess ensures that the QNO process finds an suboptimal reference action signal (\( r \)) that is as close to the previous reference action signal as possible. The QNO process adjusts \( r \) from the initial guess of the previous \( r \) until the objective function (5.9) falls below the specified threshold. This permits the full reference value to be smooth and continuous without adding any more complexity to the QNO process.
5.4: Algorithm Definitions

The ANVID software component is constructed of nested algorithms that cooperate in both series and parallel to ultimately generate suboptimal non-visual stimuli based on TORC AutonoNav™ planned trajectory data. The purpose of this subsection is to provide a detailed description of these algorithms and discuss how the functionalities of each contribute to the overall goal of the ANVID software. Each description will supply pseudocode that defines the algorithm in a simplistic and easy to follow manner. A legend of color-codes used in the pseudo-code descriptions has been provided in Algorithm 5-1.

Algorithm 5-1: Pseudocode Legend

% Comments
BuiltInFunction() % Identifies a function or subroutine that is not necessary to explicitly define.
CustomFunction() % Identifies a custom function or subroutine that will be explicitly defined.
PrivateVariables % Identifies variables private to the parent function or subroutine.
PublicVariables % Identifies variables public to all functions and subroutines.

The ANVID algorithm contains several levels of nested sub-algorithms. In order to present each sub-algorithm in an intuitive manner, the sub-algorithms will be described in the order at which they are first presented. The descriptions will begin with the ANVID() algorithm and continue through each sub-algorithm until all have been defined and discussed. Table 5-1 displays the hierarchy of the nested ANVID structure and identifies the section in which each sub-algorithm is defined and discussed.

Table 5-1: ANVID algorithm hierarchy. Each sub-algorithm is listed along with the corresponding section it is described within.
5.4.1: ANVID()
The pseudocode describing the ANVID() algorithm is given below in Algorithm 5-2. The algorithm begins by immediately starting two parallel processes. These processes work in parallel to simultaneously model the driver (Section 5.3.2) and operate the ANVID Model Predictive Controller (Section 5.3.1). The first process, DriverModeling(), trains a model of the driver in real-time based on his or her reactions to steering and speed reference signals. A complete description of this sub-algorithm can be found in Section 5.4.2. The second process, ModelPredictiveController(), uses MPC to find a suboptimal set of reference steering and speed signals as well as generate the associated non-visual stimuli. A complete description of this sub-algorithm can be found in Section 5.4.7. These parallel processes run indefinitely until the user requests a software stop.

Algorithm 5-2: ANVID() Pseudocode.

The simultaneous driver modeling and Model Predictive Control is made possible through the use of two separate loops in parallel. The advantage of a parallel structure is that it completely decouples the execution of each sub-algorithm while still enabling data communication between them. This greatly simplifies implementation and troubleshooting because the execution of each sub-algorithm is completely separate. The only interaction between the driver modeling and MPC is the one-way passing of the driver model. An additional advantage posed by the parallel structure is that it can operate in true parallel on multicore processor computers, such as the Non-Visual Interface Computer described in Section 3.5.

5.4.2: DriverModeling() The purpose of the DriverModeling() algorithm is to supply the MPCr with a current model of the driver. As discussed in Section 5.3.1, the ANVID makes use of an online-trained driver model to adapt to changes in the driver’s dynamics over time. Thus, the DriverModeling() algorithm runs in parallel with the MPCr and continuously updates the model of the driver in real-time. The driver model is stored in a shared memory location that can be accessed by the MPCr during the reference optimization process. The pseudocode that defines this algorithm is presented in Algorithm 5-3.
Algorithm 5-3: DriverModeling() Pseudocode.

Subroutine DriverModeling()
    [DriverSteeringModel, DriverSpeedModel] = LoadDriverModels(DriverName)
    Start Parallel Processes
        ManageBuffers()
        TrainModel(DriverSteeringModel)
        TrainModel(DriverSpeedModel)
    End Parallel Processes
    SaveDriverModels()
End Subroutine

The algorithm begins by loading saved versions of the current driver’s model from previous ANVID operation sessions using the LoadDriverModels() function. This function simply loads a file containing the steering and speed TSPNN models saved for the driver specified by the DriverName global variable. The DriverName global variable is set by the user to define which driver will be operating the vehicle. The loading and saving feature enables a model to be stored for each driver that can be continuously updated and improved with each system use. The file containing the model simply stores the network structure, including input and output definitions, and the trained weights of each neuron within the network. The loaded models are stored in the DriverSteeringModel and DriverSpeedModel global variables. These global variables store the actual driver models for steering and speed defined in Section 5.3.2.2 and provide the MPCr optimization process with the current driver models.

Once the models are loaded into memory, three parallel processes begin. The first process, called ManageBuffers(), stores the actual and reference signal data for steering and speed into training buffers as described in Section 5.3.2.3.3. These buffers store the input and output data used for evaluating and training the steering and speed TSPNN driver models with online, supervised learning. The actual and reference signal data is provided from the TORC ByWire XGV™ and the ModelPredictiveControl() process through JAUS messages (Section 2.4.4). The JAUS messages are handled by the MessageHandler, a VI that will be described later in Section 5.5.2.6. The TrainModel() algorithm continuously trains the driver steering and speed models as fast as possible until the user requests a software stop. A complete definition of the TrainModel() algorithm is provided in Section 5.4.3.

Once a software stop has been requested by the user, the three parallel processes stop execution. The algorithm then uses the SaveDriverModels() subroutine to save the current state of the steering and speed TSPNN driver models to a file on the Non-Visual Interface Computer (Section 3.5). The saved models can then be loaded the next time the driver operates the vehicle to maintain the trained model indefinitely.
5.4.3: TrainModel()

The purpose of the TrainModel() algorithm is to train the steering and speed TSPNN driver models in real-time with online supervised learning. This algorithm simply implements the MEKA trainer that was discussed in Section 5.3.2.3. The DriverModeling() algorithm calls two instances of this algorithm in parallel; one to train the steering model and one to train the speed model. Because the models are decoupled (Section 5.3.2), they can be trained in parallel to decrease the amount of processing time required for each training iteration. The pseudocode that defines this algorithm is presented in Algorithm 5-4.

**Algorithm 5-4: TrainModel() Pseudocode.**

```plaintext
Subroutine TrainModel(DriverModel)
    While StopRequested = False
        If CheckBufferFill(DriverModel) = True Then
            [Inputs, CorrectOutputs] = ReadTrainingBuffer()
            PredictedActions = EvaluateModel(DriverModel, Inputs)
            MEKA(DriverModel, PredictedActions, CorrectOutputs)
            ShieldModel(DriverModel, Inputs, PredictedActions, DivergenceThreshold)
        End If
    End While
End Subroutine
```

The algorithm begins by starting a while loop that continues to iterate until the user requests a software stop. This enables the continual training of the particular driver model in parallel with the MPCr process. Each iteration of the while loop is equivalent to a single online-training iteration. The iteration begins by first checking if the training data buffers (Section 5.3.2.3.3) are filled using the CheckBufferFill() function. The driver model cannot be trained if the training buffers are not full and missing parts of the actual and reference signals. This can only occur during the first iterations of the algorithm when the system is started. It will take the longest length of time that the buffer stores to initially fill the buffer with the required data.

If the training buffers are at capacity, the algorithm will actually perform a training iteration on the TSPNN driver model. The training iteration begins by reading the most current inputs and outputs from the training buffers with the ReadTrainingBuffer() function and storing the data in the Inputs and CorrectedOutputs local variables. The Inputs variable contains the time-shifted past actual data and “future” reference signal data that will be passed into the TSPNN driver model. The CorrectedOutputs variable contains the time-shifted “future” responses of the driver that will be used as supervisory output data to train the TSPNN driver model. The data read from the buffers is downscaled appropriately to maximize the prediction performance of the TSPNN. A definition of the time-shifted data for each model and description of the associated theory is provided in Section 5.3.2.3.
Once the necessary input data has been read, the algorithm then evaluates the TSPNN driver model with the EvaluateModel() algorithm and stores the resulting output in the PredictedActions local variable. A complete definition of the EvaluateModel() algorithm is provided in Section 5.4.4. Recalling from Section 5.3.2.3, the resulting output of the TSPNN model evaluation in this case is the model’s prediction of driver actions that have already occurred. The actions that have already occurred were previously stored in the CorrectOutputs variable and will be used to “correct” the TSPNN model.

The algorithm then uses the MEKA() algorithm to perform the actual weight updates of the TSPNN. This algorithm simply implements the MEKA trainer discussed in Section 5.3.2.3. A complete definition of the MEKA() algorithm itself can be found in Section 5.4.5. After the TSPNN driver model has been trained, it is passed through the ShieldModel() algorithm to protect the model from diverging. A complete definition of this algorithm is provided in Section 5.4.6. The completion of the ShieldModel() algorithm marks the completion of a single training iteration. This process is repeated continuously at maximum speed through the while loop until a software stop is requested by the user.

5.4.4: EvaluateModel()

The EvaluateModel() algorithm is responsible for evaluating a specified TSPNN driver model using the appropriate model inputs. The algorithm subjects the TSPNN’s internal neural network with the given inputs and evaluates the outputs of each neuron and layer of the network itself. This algorithm can be called during the training process to discover the prediction error or can be called by the MPC optimization process to actually predict the actions of the driver. The pseudocode definition of this algorithm is presented in Algorithm 5-5.

**Algorithm 5-5: EvaluateModel() Pseudocode.**

```plaintext
Function EvaluateModel(DriverModel, Inputs) Returns ModelOutput
    LayerInputs = Inputs
    For Each Layer in DriverModel.Layers
        For Each Neuron in Layer % Parallel For Loop
            Outputs[Neuron] = EvaluateNeuron(Neuron, LayerInputs)
        Next
        LayerInputs = Outputs
    Next
    % Outputs are now the outputs of the last layer.
    Return Outputs
End Function
```

The algorithm begins by initializing the LayerInputs local variable to the Inputs of the TSPNN driver model. This sets the network inputs as the inputs to the first layer of the neural network. A for loop is then used to iterate through each layer of the neural network. Within each layer, a parallel for loop is used to calculate the outputs of each
neuron within the layer using the `EvaluateNeuron()` function. This function applies the layer inputs to the specified neuron and calculates the associated output using (5.1). The parallel for loop is used to perform the `EvaluateNeuron()` function in parallel for all neurons within the current layer. The parallel calculation decreases the required processing time if a multi-core processor is utilized, such as the quad-core processor of the Non-Visual Interface computer (Section 3.5). The parallel calculation is able to be conducted because the neurons within a single layer of a Multi-Layer Perceptron network are completely independent of each other. The outputs of each neuron within the layer are then stored as the inputs to the next layer through the `LayerInputs` local variable. The outputs of the last layer within the neural network are the outputs of the neural network itself and are returned as the `ModelOutput`.

5.4.5: MEKA()

The MEKA() algorithm is responsible for performing a single MEKA training iteration on a specified TSPNN driver model. This algorithm implements the MEKA trainer presented in Section 5.3.2.3 and performs a single, recursive update to the neural network weights at each training iteration. The training is performed online with supervisory data retrieved from the training buffers discussed in Section 5.3.2.3.2. The pseudocode definition of the MEKA() algorithm is provided in Algorithm 5-6.

![Algorithm 5-6: MEKA() Pseudocode.](image)

The Extended Kalman Filters (EKFs) operated by the MEKA at each neuron rely on backpropagated error to adjust the neuron weight vector accordingly. Thus, the layers must be iterated through in reverse order in coordination with backpropagation. The first step of the algorithm is to therefore reverse the order of the TSPNN driver model layers. Next, the “backpropagated” error of the last layer of the network is simply set to the difference between the correct neural network output and the actual neural network output (in this case `PredictedActions`). The algorithm then begins iterating through the layers of the TSPNN driver model in reverse order. At each layer iteration, a parallel for loop is used to conduct the EKFs for each neuron within the layer in parallel. The EKF is conducted using the `EKF()` function, which simply implements (5.4)-(5.8). It is important to note that the neuron weight vector and inverse covariance matrix is stored
Within the Neuron local variable. The EKF() algorithm uses this information and the neuron’s backpropagated error stored in NeuronErrors to perform a single recursive update to the neuron's weight vector. Similarly to EvaluateModel(), the EKF of each neuron within the layer is conducted in parallel with the parallel for loop. Once again, this decreases the required processing time for each layer on multi-core CPUs such as that utilized by the Non-Visual Interface Computer (Section 3.5). The calculation can be performed in parallel since the EKFs are local to the neurons and have no interdependence.

Once the weight update has been completed, the backpropagated error through the current layer is calculated using the Backpropagate() function. This function uses a stored copy of the layer’s weights before the training update to backpropagate the error of each neuron to the previous layer using standard backpropagation [136]. This backpropagation cannot be conducted inside of the parallel for loop as the calculation depends on the interdependency of the layer’s neurons. The backpropagated error through the current layer is finally stored in the NeuronErrors local variable for use by the previous layer in the reversed layer iterations.

5.4.6: ShieldModel()

The purpose of the ShieldModel() algorithm is to prevent the model from diverging due to training errors. Such errors occur if the driver exhibits uncharacteristic responses in steering and speed to the reference signals generated by the ANVID. An example of this scenario includes cases in which the driver deliberately ignores the non-visual stimuli. The algorithm functions by detecting model divergence after a single MEKA training iteration and throws away the malicious training data if divergence has occurred. The pseudocode definition of this algorithm is provided in Algorithm 5-7.

**Algorithm 5-7: ShieldModel() Pseudocode.**

```
Subroutine ShieldModel(DriverModel, Inputs, PredictedActions, DivergenceThreshold)
  UntrainedOutputs = PredictedActions
  TrainedOutputs = EvaluateModel(DriverModel, Inputs)
  MeanSquaredError = MSE(UntrainedOutputs, TrainedOutputs)
  If MeanSquaredError > DivergenceThreshold Then
    Restore(DriverModel)
  End If
End Subroutine
```

The algorithm begins by setting the UntrainedOutputs local variable equal to the PredictedActions parameter passed in by the calling algorithm. The untrained outputs are the output of the TSPNN driver model was just prior to the MEKA training iteration. Conversely, the trained outputs are the output of the TSPNN driver model just after the MEKA training iteration. The trained outputs are found by evaluating the TSPNN driver model with the updated weights using the EvaluateModel() function previously defined.
in Section 5.4.4. It is important to note that the evaluation of the trained output is performed using the same inputs that resulted in the untrained model output. This enables any difference in the trained and untrained outputs to be directly attributed to the weight updates from the MEKA training iteration.

Once the trained and untrained outputs to the same input are known, the two outputs are compared via Mean-Squared Error (MSE). The MSE() function performs the MSE calculation and stores the mean-squared error of the trained output to the untrained output in the MeanSquaredError local variable. An if statement is then used to determine if the MSE is above the designer-specified divergence threshold. If the training iteration causes the driver model to diverge, the MSE between the untrained and trained TSPNN outputs will result in a relatively large number. If the MSE is greater than the divergence threshold, the training data is thrown away by the Restore() function. This function simply restores the weights of the TSPNN driver model to the values previous to the weight update through the MEKA training iteration. It is through this method that the model remains stable and accurate even in situations when the driver is ignoring the non-visual stimuli generated by the ANVID.

5.4.7: ModelPredictiveController()

The ModelPredictiveController() algorithm is responsible for using Model Predictive Control to generate suboptimal reference steering and speed signals based on the TORC AutonoNav™ motion profiles. The reference signals are found through a Quasi-Newton Optimization process that utilizes the TSPNN driver prediction models to adapt to each particular driver. Aside from calculating the suboptimal reference signals, the MPCr additionally generates the associated non-visual stimuli over the Non-Visual Interface System (Section 3). The pseudocode definition for the ModelPredictiveController() algorithm is presented in Algorithm 5-8 below.

Algorithm 5-8: ModelPredictiveController() Pseudocode.

<table>
<thead>
<tr>
<th>Subroutine ModelPredictiveController()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Parallel Processes</td>
</tr>
<tr>
<td>ManageInputBuffers()</td>
</tr>
<tr>
<td>OptimizeReferenceSignals()</td>
</tr>
<tr>
<td>CalculateNonVisualStimuli()</td>
</tr>
<tr>
<td>GenerateNonVisualStimuli()</td>
</tr>
<tr>
<td>End Parallel Processes</td>
</tr>
<tr>
<td>End Subroutine</td>
</tr>
</tbody>
</table>

The ModelPredictiveController() algorithm begins by immediately starting four parallel processes. The first process, called ManageInputBuffers(), is simply responsible for managing input buffers similar to the TSPNN driver model buffers discussed in Section 5.3.2.3.2. These buffers are required because they store the past actual steering and speed actions of the driver that are needed to predict the future actions of the driver in
the optimization process. As the driver steering and speed actions are recorded by the TORC ByWire XGV™, the ManageInputBuffers() function records the actions into the appropriate input buffers. The actions are physically communicated from the ByWire XGV™ through JAUS messages handled by the MessageHandler VI (Section 5.5.2.3).

The second parallel process is the OptimizeReferenceSignals() algorithm. This algorithm is responsible for performing the Quasi-Newton Optimization processes that calculate the suboptimal reference steering and speed signals for each new motion profile received from the TORC AutonoNav™ system. A complete definition of this algorithm is provided in Section 5.4.8. The third parallel process is the CalculateNonVisualStimuli() algorithm. This algorithm calculates appropriate non-visual stimuli based on the current suboptimal reference steering and speed values and the driver’s associated responses. The complete definition of this algorithm is provided in Section 5.4.10. The fourth and final parallel process is the GenerateNonVisualStimuli() algorithm. This algorithm is responsible for physically generating the calculated non-visual stimuli over the Non-Visual Interface System (Section 3). This algorithm is the same algorithm used in the PNVID and has been previously defined in Section 4.2.10.

5.4.8: OptimizeReferenceSignals()

The OptimizeReferenceSignals() algorithm is responsible for calculating the suboptimal reference steering and speed signals each time a new motion profile is received from the TORC AutonoNav™ Motion Planner. This process continuously calculates updated suboptimal reference signals from the motion profiles until the user requests a software stop. The suboptimal reference signals are then stored in global variables for use by other parallel processes. As discussed in Section 5.3.2.2, the suboptimal reference signals contain a 750ms long horizon of future reference data with an 8Hz sample frequency. The pseudocode definition of this algorithm is presented in Algorithm 5-9.

<table>
<thead>
<tr>
<th>Subroutine OptimizeReferenceSignals()</th>
</tr>
</thead>
<tbody>
<tr>
<td>While StopRequested = False</td>
</tr>
<tr>
<td>While NewMotionProfileFlag = False and StopRequested = False</td>
</tr>
<tr>
<td>End While</td>
</tr>
<tr>
<td>If NewMotionProfileFlag = True Then</td>
</tr>
<tr>
<td>NewMotionProfileFlag = False</td>
</tr>
<tr>
<td>Start Parallel Processes</td>
</tr>
<tr>
<td>SuboptimalSteeringReferenceSignal = QNO(&quot;Steering&quot;)</td>
</tr>
<tr>
<td>SuboptimalSpeedReferenceSignal = QNO(&quot;Speed&quot;)</td>
</tr>
<tr>
<td>End Parallel Processes</td>
</tr>
<tr>
<td>NewReferenceSignalFlag = True</td>
</tr>
<tr>
<td>End If</td>
</tr>
<tr>
<td>End Subroutine</td>
</tr>
</tbody>
</table>
The algorithm begins by starting a while loop that continuously determines suboptimal reference steering and speed signals until the user requests a software stop. Each iteration of this while loop contains a single optimization iteration that is conducted each time a new motion profile is received. The algorithm uses a nested while loop to wait indefinitely until the NewMotionProfileFlag global variable is set to true or the user requests a software stop. The NewMotionProfileFlag is externally set to true by the MessageHandler VI (Section 5.5.2.3) whenever a new motion profile is received from the Motion Planner JAUS component via a JAUS message. Once the nested while loop exits, an if statement is used to determine if a new motion profile is available.

If a new motion profile is available, the algorithm sets the NewMotionProfileFlag to False and begins two separate optimization processes in parallel. One optimization process calculates the suboptimal steering reference signal while the other process calculates the suboptimal speed reference signal. Although these processes are separate and parallel, they utilize the same algorithm for performing the Quasi-Newton Optimization. This algorithm, called QNO(), outputs the suboptimal reference signal for the action type specified by its only calling parameter. A complete definition of this algorithm can be found in Section 5.4.9. Once suboptimal reference signals are found for steering and speed, they are stored in the SuboptimalSteeringReferenceSignal and SuboptimalSpeedReferenceSignal global variables. These global variables ultimately provide the parallel CalculateNonVisualStimuli() algorithm defined in Section 5.4.10 with the most up-to-date suboptimal reference signals. Once the global variables are set, the optimization iteration is complete and the NewReferenceSignalFlag is set to True.

5.4.9: QNO()

The purpose of the QNO() algorithm is to perform a Quasi-Newton Optimization process to calculate suboptimal reference action signals based on the commanded actions from the motion profile and the current driver model. This algorithm simply implements the Reference Optimization theory discussed in Section 5.3.3 of this work. The pseudocode definition of the QNO() algorithm is presented in Algorithm 5-10.

The algorithm begins by interpolating the commanded actions of the motion profile for the particular type of reference action being optimized. Recalling from Section 2.3.2.2.3, the motion profiles only include a series of desired values and rates for each steering and speed motion. The InterpolateMotionProfile() function is used to interpolate 750ms of commanded actions with a 40Hz sample frequency so that the predicted driver actions can be compared to the commanded actions later in the optimization process. Next, the initial guess is chosen with the weighted average between the previous iteration’s optimized reference signal and the commanded actions as discussed in Section 5.3.3.4. The initial guess is stored in the ReferenceSignal local variable which holds the current candidate reference signal solution throughout the QNO process.
Algorithm 5-10: QNO() Pseudocode.

```
Function QNO(ActionType) Returns SuboptimalActionReferenceSignal
    CommandedActions = InterpolateMotionProfile(CurrentMotionProfile.ActionType)
    % Set Initial Guess
    ReferenceSignal = WAvg(ReferenceActions, CommandedActions, 10, 1)
    While ObjectiveFunction(ReferenceSignal, DriverModel, CommandedActions) > MaxCost and _
        _OptimizationTime() < TimeLimit
        ReferenceSignal = UpdateReferenceSignal()
    End While
    Return ReferenceSignal
End Function
```

The algorithm then begins the actual optimization iterations that generate new reference signals and observe the associated cost of each through the objective function. The iterations are conducted through a while loop that continuously modifies the candidate reference signal until the cost from the objective function falls below a maximum cost threshold or the optimization time has surpassed the allowable time limit. The MaxCost and TimeLimit local variables can be configured by the designer to adjust the accuracy and timing characteristics of the QNO process. The ObjectiveFunction() algorithm simply implements (5.9) and the OptimizationTime() function returns the current time consumed by the optimization process.

During each optimization iteration, the cost associated with the candidate reference signal stored in ReferenceSignal is assessed using the ObjectiveFunction() algorithm. Recalling from Section 5.3.3, the objective function essentially calculates the predicted driver actions in response to the candidate reference signal and compares it to the commanded actions specified by the motion profile. It is important to note that the DriverModel parameter of the ObjectiveFunction algorithm is the global variable containing either the driver steering or driver speed TSPNN model updated by the DriverModeling() algorithm in Section 5.4.2. These models are updated in real-time through online supervised training and in parallel to the optimization process. Thus, the objective function utilizes the most up-to-date version of the driver steering and speed models to calculate the predicted driver actions.

The predicted driver actions for the associated candidate reference signal are compared to the commanded actions specified by the current motion profile through Mean-Squared-Error (MSE). If the mean-squared-error (MSE) is too large, a new candidate reference signal is calculated using quadratic gradient descent. This calculation is performed using the UpdateReferenceSignal() algorithm, which simply implements (5.11)-(5.14) from the Reference Optimization theory in Section 5.3.3. This function is called repetitively until the MSE falls below a specified maximum cost value or the QNO process has exhausted its allotted timeframe. Once one of these conditions is met, the most recent candidate reference signal is returned as the suboptimal reference signal.
5.4.10: CalculateNonVisualStimuli()

The purpose of the CalculateNonVisualStimuli() equation is to form the full steering and reference signals and calculate the non-visual stimuli resulting from the driver’s steering and speed responses. Recalling from Section 5.3.3.4, the suboptimal reference signals determined by the OptimizeReferenceSignals() algorithm form chains that overlap in time as new motion profiles are received. This algorithm merges the steering and speed chains into full steering and speed reference signals by iterating through the most current suboptimal steering and speed reference values in time. The non-visual stimuli are then found by calculating the difference between the instantaneous driver responses and the current reference values. The pseudocode definition of this algorithm can be found in Algorithm 5-11.

Algorithm 5-11: CalculateNonVisualStimuli() Pseudocode.

```
Subroutine CalculateNonVisualStimuli()
    % Parallel While Loop #1
    While StopRequested = False
        Wait for 125ms interval
        % Determine instantaneous reference values.
        CurrentSteeringReferenceValue = SuboptimalSteeringReferenceSignal [SampleIndex]
        CurrentSpeedReferenceValue = SuboptimalSpeedReferenceSignal [SampleIndex]
        If NewReferenceSignalFlag = False Then
            SampleIndex = SampleIndex + 1
        Else
            SampleIndex = 0
            NewReferenceSignalFlag = False
        End If
    End While

    % Parallel While Loop #2
    While StopRequested = False
        Wait for 25ms interval
        % Determine instantaneous errors.
        InstantaneousSteeringError = CurrentSteeringReferenceValue - CurrentSteeringAngle
        InstantaneousSpeedError = CurrentSpeedReferenceValue - CurrentSpeed
        % Determine non-visual stimuli.
        DriveGripStimulus = ConvertToDriveGripStimulus(InstantaneousSteeringError)
        SpeedStripStimulus = ConvertToSpeedStripStimulus(InstantaneousSpeedError)
    End While
End Subroutine
```

This algorithm functions by running two parallel while loops: one to index the current reference steering and speed values at 8Hz and one to calculate the non-visual stimuli at 40Hz. Both while loops run continuously to calculate the non-visual stimuli until a software stop is requested by the user. The first parallel loop begins by waiting for the next 125ms interval to ensure that the reference signals are iterated through at 8Hz. Next, the instantaneous steering and speed reference values are indexed out from the
current suboptimal reference steering and speed signals calculated and stored into global variables by the OptimizeReferenceSignals() algorithm in Section 5.4.8. The sample index keeps track of how far the suboptimal reference signals have been iterated through in time. Once the instantaneous reference values are stored in the appropriate local variables, the NewReferenceSignalFlag is checked to determine how to update the sample index. If the flag is set to true by the OptimizeReferenceSignals() algorithm, a new set of suboptimal reference steering and speed signals have been calculated from a new motion profile. Thus, the sample index is reset to zero as the suboptimal reference signal global variables contain a new set of signals. If the NewReferenceSignalFlag is false, the sample is incremented by one to continue iterating through the current suboptimal reference signals in time.

The second parallel while loop begins by waiting for the next 25ms interval to ensure that the non-visual stimuli are calculated at 40Hz. Next, the instantaneous steering and speed errors are found by taking the difference between the current values and the reference values. The current steering and speed values are read from global variables that are updated continuously by JAUS service connections with the TORC ByWire XGV™ handled by the Message Handler (Section 5.5.2.3). The reference values are the instantaneous reference steering and speed values determined by the first parallel loop in this algorithm. Once the instantaneous errors are found, the DriveGrip and SpeedStrip non-visual stimuli are calculated using the ConvertToDriveGripStimulus() and ConvertToSpeedStripStimulus() algorithms. These are the same algorithms utilized by the PNVID and convert the instantaneous steering and speed errors into DriveGrip and SpeedStrip stimuli. The definitions of these algorithms have already been provided in Section 4.2.7 and Section 4.2.9, respectively. The resulting DriveGrip and SpeedStrip stimuli are then stored in global variables for use by the parallel GenerateNonVisualStimuli() method in the ModelPredictiveController() algorithm. This process repeats continually at 40Hz until the user requests a software stop.
5.5: Software Implementation

The ANVID algorithm defined in Section 5.4 was implemented in two specialized pieces of software, called the MPC Component and the Driver Modeling Component. These components were specifically designed to realize the ANVID algorithm and provide the necessary JAUS interoperability for implementation on the JAUS-based Blind Driver Challenge™ system described in Section 2.4.4.2. This subsection documents the development tools used to write the ANVID software components, presents the algorithm that defines the software itself, and concludes with a discussion of how the software is executed within the comprehensive Blind Driver Challenge™ system.

5.5.1: Development Tools

The development of the ANVID software application relied heavily on the use of two important programming tools. The first tool, called LabVIEW™, is a graphical programming language designed by National Instruments that was used to write the actual software of the ANVID implementation. The second tool, called the TORC JAUS Toolkit™ (JTK), is an add-on module to the LabVIEW Development Environment and dramatically decreases the amount of development required to create a JAUS interoperable software application. These two development tools were utilized by the PNVID software implementation and were previously described with detail in Sections 4.3.1.1 and 4.3.1.2, respectively.

5.5.2: Algorithm Definitions

The ANVID software is implemented as two separate JAUS components. The first component, called the MPC Component, implements the model-predictive controller of the ANVID and is ultimately responsible for generating the non-visual stimuli. The second component, called the Driver Modeling Component, is only responsible for performing the real-time, online supervised training of the driver model in parallel with the MPC Component operation. Both of these components are constructed from several LabVIEW Virtual Instruments (VIs) that cooperate in both series and parallel to implement the necessary functions. The purpose of this subsection is to provide a detailed description of these VIs and discuss how the functionalities of each contribute to the overall goal of the ANVID software. Each description will supply pseudocode that defines the VI algorithm in a simplistic and easy to follow manner. A legend of color-codes used in the pseudo-code descriptions has been provided in Algorithm 5-12.

Algorithm 5-12: Pseudocode Legend

<table>
<thead>
<tr>
<th>% Comments</th>
<th>Custom.vi</th>
<th>% Identifies custom software VIs that will be explicitly defined.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TORC_JTK.vi</td>
<td>% Identifies TORC JAUS Toolkit VIs that will be summarized.</td>
</tr>
<tr>
<td></td>
<td>PublicVariables</td>
<td>% Identifies variables public to all software VIs.</td>
</tr>
</tbody>
</table>
The MPC and Driver Modeling Components each contain several levels of nested sub-VIs. In order to present each and every VI in an intuitive manner, the VIs will be described in the order at which they are first presented. The descriptions will begin with the MPCComponent VI and continue through each VI until all have been defined and discussed. It is important to note that several of the VIs utilized by the ANVID software implementation are shared with the PNVID software implementation. The shared VIs will be referenced to the appropriate sections in the PNVID Software Implementation discussion (Section 4.3) where necessary. Conversely, the VIs particular to the ANVID implementation will be defined in this subsection. Table 5-2 displays the hierarchy of the MPC and Driver Modeling Component VIs and identifies the section in which each VI is defined and discussed.

Table 5-2: ANVID software custom VI hierarchy and section index.

<table>
<thead>
<tr>
<th>Custom VI Hierarchy</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: MPCComponent.vi</td>
<td>5.5.2.1</td>
</tr>
<tr>
<td>1.1: ExecuteMPC.vi</td>
<td>5.5.2.2</td>
</tr>
<tr>
<td>1.1.1: MPCMessageHandler.vi</td>
<td>5.5.2.3</td>
</tr>
<tr>
<td>2: DriverModelingComponent.vi</td>
<td>5.5.2.4</td>
</tr>
<tr>
<td>2.1: ExecuteDriverModeling.vi</td>
<td>5.5.2.5</td>
</tr>
<tr>
<td>2.1.1: DMMMessageHandler.vi</td>
<td>5.5.2.6</td>
</tr>
</tbody>
</table>

5.5.2.1: MPCComponent.vi

The purpose of the MPCComponent VI is to realize the ModelPredictiveController() (MPC) algorithm presented in Section 5.4.7 and provide full compatibility with the Joint Architecture for Unmanned Systems (JAUS). As described in Section 2.4.4, the Blind Driver Challenge™ platform implements JAUS to create an organized system topology and standardize the communication between TORC Robotics and Virginia Tech software. Since the MPC algorithm relies on data from TORC Robotics software, the MPCComponent VI must therefore possess the ability to communicate with the TORC software through JAUS defined methods and messages. Pseudocode has been provided in Algorithm 5-13 to fully define these functionalities.

Generally speaking, the MPCComponent VI utilizes the TORC JTK to create a JAUS component that executes the MPC algorithm. According to the Reference Architecture Specification in [110], the component must have the ability to be started, stopped, and reset at the discretion of other system components. The while loop that this VI begins with enables the JAUS component to be reset infinitely within the execution of the VI itself. The pseudocode within the while loop represents a single life cycle of the JAUS component and exits when a stop or reset has been requested by the driver. If a reset is requested, the while loop repeats and resets the component. If a stop is requested
instead of a reset, the while loop exits and the component is destroyed in memory through the DestroyComponent VI of the JTK.

**Algorithm 5-13: MPComponent.vi Pseudocode**

```plaintext
Main MPComponent.vi
    While Reset = True
        InitializeComponent.vi
        StartComponent.vi
        Start Parallel Processes
            ExecuteMPC.vi
            MonitorStopReset.vi
        End Parallel Processes
        CloseServiceConnections.vi
        StopComponent.vi
    End While
End Main
```

A single life cycle of the JAUS component begins with the InitializeComponent custom VI defined previously in Section 4.3.2.2. This VI creates or recreates the JAUS component in memory and sets the necessary configuration parameters. The StartComponent JTK VI is then called to start the background process that dynamically connects to the TORC Node Manager and handles the core subgroup functionalities of the JAUS component defined in [107]. Once the component has been started, the ExecuteMPC and MonitorStopReset custom VIs are started in parallel. The ExecuteMPC VI defined in Section 5.5.2.2 performs additional JAUS-related initialization procedures and subsequently executes the actual MPC algorithm. The MonitorStopReset VI defined previously in Section 4.3.2.7 continuously monitors if a JAUS stop or reset has occurred and sets the Stop and Reset public variables accordingly. Once a stop or reset occurs, both parallel processes exit and the CloseServiceConnection JTK VI is called to close the active service connections of the MPC Component. The component is then stopped and disconnected from the TORC Node Manager with the StopComponent JTK VI. This marks the end of a single life cycle of the MPC JAUS component.

5.5.2.2: **ExecuteMPC.vi**

The purpose of the ExecuteMPC VI is to ensure that the necessary TORC software components are online, secure the necessary message service connections, and finally run the MPC algorithm defined in Section 5.4.7. The full definition of this VI is presented with pseudocode in Algorithm 5-14 below. The VI begins with calling the WaitForComponents VI defined previously in Section 4.3.2.4. This custom VI forces the thread to wait until all the required TORC software components are online. Next, the VI calls the EstablishServiceConnections VI previously defined in Section 4.3.2.5 to create the necessary message service connections with the TORC software
components. Once these first two steps are complete, the SetReadyState JTK VI is called to set the component to the ready state. This identifies to other software components in the system that the MPC component has completed initialization and is now under normal operation.

**Algorithm 5-14: ExecuteMPC.vi Pseudocode**

<table>
<thead>
<tr>
<th>Subroutine ExecuteMPC.vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitforComponents.vi</td>
</tr>
<tr>
<td>EstablishServiceConnections.vi</td>
</tr>
<tr>
<td>% Initialization complete, begin normal operation.</td>
</tr>
<tr>
<td>SetReadyState.vi</td>
</tr>
<tr>
<td>Start Parallel Processes</td>
</tr>
<tr>
<td>MPCMessageHandler.vi</td>
</tr>
<tr>
<td>ManageInputBuffers.vi</td>
</tr>
<tr>
<td>OptimizeReferenceSignals.vi</td>
</tr>
<tr>
<td>CalculateNonVisualStimuli.vi</td>
</tr>
<tr>
<td>GenerateNonVisualStimuli.vi</td>
</tr>
<tr>
<td>End Parallel Processes</td>
</tr>
</tbody>
</table>

The normal operation of the MPC component consists of five parallel processes. The first process is the MPCMessageHandler custom VI and is defined in Section 5.5.2.3. This VI manages all incoming messages, including service connections, and stores the associated information in the necessary public variables for use by the MPC algorithm. The MPC algorithm itself is realized by the remaining four processes running in parallel with the MPCMessageHandler VI. The second parallel process is the ManageInputBuffers VI, which realizes the ManageInputBuffers() sub-algorithm of the MPC algorithm and was previously defined in Section 5.4.7. The third parallel process is the OptimizeReferenceSignals VI, which realizes the OptimizeReferenceSignals() sub-algorithm previously defined in Section 5.4.8. The fourth parallel process is the CalculateNonVisualStimuli VI, which realizes the CalculateNonVisualStimuli() sub-algorithm previously defined in Section 5.4.10. The fifth and final parallel process is the GenerateNonVisualStimuli VI, which realizes the GenerateNonVisualStimuli() sub-algorithm of the MPC algorithm previously defined in Section 4.2.10. All five of these parallel processes execute the MPC algorithm continuously until a JAUS Stop or Reset event occurs through the MonitorStopReset VI defined previously in Section 4.3.2.7.

**5.5.2.3: MPCMessageHandler.vi**

The purpose of the MPCMessageHandler VI is to receive and parse incoming JAUS messages and store the resulting values in the corresponding public variables. The VI is notified of incoming messages through the JTK component VIs and only handles messages that are not part of the core message group. Recalling from Section 4.3.1.2, core subgroup messages are internally handled by the JTK component VIs to simplify JAUS implementation for the developer. Thus, the MPC Component is configured to
only receive five types of JAUS messages. Before these messages are discussed in detail, the full pseudocode definition of the MPCMessageHandler VI is presented in Algorithm 5-15.

**Algorithm 5-15: MPCMessageHandler.vi Pseudocode**

```plaintext
Subroutine MPCMessageHandler.vi
    While Stop = False and Reset = False
        Select Case NewIncomingMessage
            Case “Set Motion Profile (xE238)”
                CurrentMotionProfile = ReadIncomingMessage()
                NewMotionProfileFlag = True
            Case “Report Velocity State (x4404)”
                CurrentSpeed = ReadIncomingMessage()
            Case “Report Wrench Efforts (x4405)”
                CurrentSteeringAngle = ReadIncomingMessage()
            Case “Query Reference Steering (xE500)”
                ReportReferenceSteering()
            Case “Query Reference Speed (xE502)”
                ReportReferenceSpeed()
            Case “No Message”
                % Do nothing.
        End Select
    End While
End Subroutine
```

The first message, called Set Motion Profile, is sent by the Motion Planner at a rate of 2-8Hz and contains the most up-to-date motion profile. A complete definition of this message can be found in Section 2.3.2.2.3. The second message, called Report Velocity State, is received through a 40Hz service connection with the Velocity State Sensor component of the ByWire XGV™ and reports the current longitudinal velocity of the vehicle. The third message, called Report Wrench Efforts, is received through a 40Hz service connection with the Primitive Driver component of the ByWire XGV™ and reports the current steering wheel angle of the vehicle. Both the Report Velocity State and Report Wrench Efforts messages are fully defined in Section 2.3.2.1.4. The MPCMessageHandler VI receives each of these three messages and stores the data in the public variables utilized by the MPC algorithm definitions in Section 5.4.7.

The fourth message and fifth messages, called Query Reference Steering and Query Reference Speed, are received from a service connection with the DriverModeling VI discussed in Section 5.5.2.4. Once the service connection is confirmed, the JTK background thread generates these query messages at the specified service connection rate. In this case, the rate is always 8Hz. The MPCMessageHandler VI receives each of these two messages and replies to the querying component (the DriverModel VI) with the associated Report message. The Report Reference Steering and Report Reference Speed messages are defined in Table 5-3 and Table 5-4, respectively.
The MPCMessageHandler VI runs continuously through a while loop and only ceases execution when one of the Stop or Reset public variables has been set to true by the MonitorStopReset VI previously discussed in Section 4.3.2.7. Inside of the loop, a case selector block is utilized to determine which message, if any, has been received by the MPC component. The first case occurs when the Set Motion Profile message is received. In this case, the VI reads the new motion profile from the Set Motion Profile message and stores it in the CurrentMotionProfile public variable for use by the MPC algorithm. The VI then sets the NewMotionProfileFlag public variable to true, also for use by the MPC algorithm. The second and third cases occur when the Report Velocity State and Report Wrench Efforts messages are received, respectively. The VI then reads the current speed or current steering wheel angle from these messages and stores them in the CurrentSpeed and CurrentSteeringAngle public variables for use by the MPC algorithm. Because the MPC Component receives these three messages periodically, the CurrentMotionProfile, CurrentSpeed, and CurrentSteeringAngle public variables are always kept up-to-date with the most current data. The third and fourth cases occur when the Query Reference Steering and Query Reference Speed messages are generated by the JTK background process. In each of these cases, the VI builds the corresponding Report message and sends it to the querying VI. It is in this way that the DriverModel VI (Section 5.5.2.4) is able to read the reference values that the driver has been tracking so that the steering and speed models may be continuously updated online.
5.5.2.4: DriverModelingComponent.vi
The purpose of the DriverModelingComponent VI is to realize the DriverModeling() algorithm presented in Section 5.4.2 and provide full compatibility with the Joint Architecture for Unmanned Systems (JAUS). As described in Section 2.4.4, the Blind Driver Challenge™ platform implements JAUS to create an organized system topology and standardize the communication between TORC Robotics and Virginia Tech software. Since the DriverModeling() algorithm relies on data from TORC Robotics software, the DriverModelingComponent VI must therefore possess the ability to communicate with the TORC software through JAUS defined methods and messages. Pseudocode has been provided in Algorithm 5-16 to fully define these functionalities.

Generally speaking, the DriverModelingComponent VI utilizes the TORC JTK to create a JAUS component that executes the DriverModeling() algorithm. According to the Reference Architecture Specification in [110], the component must have the ability to be started, stopped, and reset at the discretion of other system components. The while loop that this VI begins with enables the JAUS component to be reset infinitely within the execution of the VI itself. The pseudocode within the while loop represents a single life cycle of the JAUS component and exits when a stop or reset has been requested by the driver. If a reset is requested, the while loop repeats and resets the component. If a stop is requested instead of a reset, the while loop exits and the component is destroyed in memory through the DestroyComponent VI of the JTK.

Algorithm 5-16: DriverModelingComponent.vi Pseudocode

```
Main DriverModelingComponent.vi
    While Reset = True
        InitializeComponent.vi
        StartComponent.vi
        Start Parallel Processes
            ExecuteDriverModeling.vi
            MonitorStopReset.vi
        End Parallel Processes
        CloseServiceConnections.vi
        StopComponent.vi
    End While
    DestroyComponent.vi
End Main
```

A single life cycle of the JAUS component begins with the InitializeComponent custom VI defined previously in Section 4.3.2.2. This VI creates or recreates the JAUS component in memory and sets the necessary configuration parameters. The StartComponent JTK VI is then called to start the background process that dynamically connects to the TORC Node Manager and handles the core subgroup functionalities of the JAUS component defined in [107]. Once the component has been started, the ExecuteDriverModeling and MonitorStopReset custom VIs are started in parallel. The
ExecuteDriverModeling VI defined in Section 5.5.2.2 performs additional JAUS-related initialization procedures and subsequently executes the actual DriverModeling() algorithm. The MonitorStopReset VI defined previously in Section 4.3.2.7 continuously monitors if a JAUS stop or reset has occurred and sets the Stop and Reset public variables accordingly. Once a stop or reset occurs, both parallel processes exit and the CloseServiceConnection JTK VI is called to close the active service connections of the Driver Modeling Component. The component is then stopped and disconnected from the TORC Node Manager with the StopComponent JTK VI. This marks the end of a single life cycle of the Driver Modeling JAUS component.

5.5.2.5: ExecuteDriverModeling.vi
The purpose of the ExecuteDriverModeling VI is to ensure that the necessary TORC software components are online, secure the necessary message service connections, and finally run the DriverModeling() algorithm defined in Section 5.4.2. The full definition of this VI is presented with pseudocode in Algorithm 5-17 below. The VI begins with calling the WaitForComponents VI defined previously in Section 4.3.2.4. This custom VI forces the thread to wait until all the required TORC software components are online. Next, the VI calls the EstablishServiceConnections VI previously defined in Section 4.3.2.5 to create the necessary message service connections with the TORC software components. Once these first two steps are complete, the SetReadyState JTK VI is called to set the component to the ready state. This identifies to other software components in the system that the Driver Modeling component has completed initialization and is now under normal operation.

Algorithm 5-17: ExecuteDriverModeling.vi Pseudocode

```
Subroutine ExecuteDriverModeling.vi
   WaitForComponents.vi
   EstablishServiceConnections.vi
   % Initialization complete, begin normal operation.
   SetReadyState.vi
   LoadDriverModels.vi
   Start Parallel Processes
      DMMessageHandler.vi
      ManageBuffers.vi
      TrainDriverSteeringModel.vi
      TrainDriverSpeedModel.vi
   End Parallel Processes
End Subroutine
```

The normal operation of the Driver Modeling component consists of four parallel processes. The first process is the DMMessageHandler custom VI and is defined in Section 5.5.2.6. This VI manages all incoming messages, including service connections, and stores the associated information in the necessary public variables for use by the DriverModeling() algorithm. The DriverModeling() algorithm itself is realized
by the remaining three processes running in parallel with the DMMessageHandler VI. The second parallel process is the ManageBuffers VI, which realizes the ManageBuffers() sub-algorithm of the DriverModeling() algorithm and was previously defined in Section 5.4.2. The third and fourth parallel processes are the TrainDriverSteeringModel and TrainDriverSpeedModel VIs. These VIs realize the two TrainModel() sub-algorithms conducted for the driver steering and speed models and defined in Section 5.4.3. All four of these parallel processes execute the MPC algorithm continuously until a JAUS Stop or Reset event occurs through the MonitorStopReset VI defined previously in Section 4.3.2.7.

5.5.2.6: DMMessageHandler.vi

The purpose of the DMMessageHandler VI is to receive and parse incoming JAUS messages and store the resulting values in the corresponding public variables. The VI is notified of incoming messages through the JTK component VIs and only handles messages that are not part of the core message group. Recalling from Section 4.3.1.2, core subgroup messages are internally handled by the JTK component VIs to simplify JAUS implementation for the developer. Thus, the Driver Modeling Component is configured to only receive four types of JAUS messages. Before these messages are discussed in detail, the full pseudocode definition of the DMMessageHandler VI is presented in Algorithm 5-18.

Algorithm 5-18: DMMessageHandler.vi Pseudocode

<table>
<thead>
<tr>
<th>Subroutine DMMessageHandler.vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>While Stop = False and Reset = False</td>
</tr>
<tr>
<td>Select Case NewIncomingMessage</td>
</tr>
<tr>
<td>Case “Report Velocity State (x4404)”</td>
</tr>
<tr>
<td>CurrentSpeed = ReadIncomingMessage()</td>
</tr>
<tr>
<td>Case “Report Wrench Efforts (x4405)”</td>
</tr>
<tr>
<td>CurrentSteeringAngle = ReadIncomingMessage()</td>
</tr>
<tr>
<td>Case “Report Reference Steering (xE501)”</td>
</tr>
<tr>
<td>ReferenceSteeringBuffer.Add(ReadIncomingMessage())</td>
</tr>
<tr>
<td>Case “Report Reference Speed (xE503)”</td>
</tr>
<tr>
<td>ReferenceSpeedBuffer.Add(ReadIncomingMessage())</td>
</tr>
<tr>
<td>Case “No Message”</td>
</tr>
<tr>
<td>% Do nothing.</td>
</tr>
<tr>
<td>End Select</td>
</tr>
<tr>
<td>End While</td>
</tr>
<tr>
<td>End Subroutine</td>
</tr>
</tbody>
</table>

The first message, called Report Velocity State, is received through a 40Hz service connection with the Velocity State Sensor component of the ByWire XGV™ and reports the current longitudinal velocity of the vehicle. The second message, called Report Wrench Efforts, is received through a 40Hz service connection with the Primitive Driver component of the ByWire XGV™ and reports the current steering wheel angle of the
vehicle. Both the Report Velocity State and Report Wrench Efforts messages are fully defined in Section 2.3.2.1.4. The DMMessageHandler VI receives each of these two messages and stores the data in the public variables utilized by the DriverModeling() algorithm definitions in Section 5.4.2.

The third message, called Report Reference Steering, is received through an 8Hz service connection with the MPC component and reports the current instantaneous reference steering value. The fourth message, called Report Reference Speed, is also received through an 8Hz service connection with the MPC component and reports the current instantaneous speed value. When these messages are received, the DMMessageHandler VI adds the reported reference value to the associated reference input buffer as described in Section 5.3.2.3. The Report Reference Steering and Report Reference Speed messages received by the DMMessageHandler were previously defined in Table 5-3 and Table 5-4, respectively.

The DMMessageHandler VI runs continuously through a while loop and only ceases execution when one of the Stop or Reset public variables has been set to true by the MonitorStopReset VI previously discussed in Section 4.3.2.7. Inside of the loop, a case selector block is utilized to determine which message, if any, has been received by the Driver Modeling component. The first and second cases occur when the Report Velocity State and Report Wrench Efforts messages are received, respectively. Upon receiving these messages, the VI subsequently reads the current speed or current steering wheel angle from the message and stores it in the CurrentSpeed or CurrentSteeringAngle public variables for use by the DriverModeling() algorithm. Because the Driver Modeling Component receives these two messages periodically, the CurrentSpeed and CurrentSteeringAngle public variables are always kept up-to-date with the most current data. The third and fourth cases occur when the Report Reference Steering and Report Reference Speed messages are received from the MPC component. In each of these cases, the received reference value is added to the training buffers described in Section 5.3.2.3. It is in this way that the Driver Model component is able to read the reference values that the driver has been tracking so that the steering and speed models may be continuously updated online.

5.5.3: Software Execution
As described in Sections 2.4.4.2 and 3.5, the Non-Visual Interface Computer (NVICPU) hosts the ANVID software and acts as a single computing node within the JAUS network of the Blind Driver Challenge™ platform. Because the ANVID software executes on the NVICPU, the MPC and Driver Modeling components are classified as components within the NVICPU node. The ANVID components are able to communicate with other software components within the platform’s JAUS network through the network connection and Node Manager of the NVICPU.
Each component within the JAUS network contains special properties that identify the component within the system and describe the services that the component offers. These properties are defined for the ANVID MPC and Driver Modeling components in Table 5-5 and Table 5-6, respectively.

**Table 5-5: MPC Component Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Identification Type</th>
<th>Component ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC</td>
<td>30001</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported Incoming Messages</th>
<th>Service ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Motion Profile</td>
<td>15</td>
<td>xE238</td>
</tr>
<tr>
<td>Report Velocity State</td>
<td>15</td>
<td>x4404</td>
</tr>
<tr>
<td>Report Wrench Efforts</td>
<td>15</td>
<td>x4405</td>
</tr>
<tr>
<td>Query Reference Steering</td>
<td>15</td>
<td>xE500</td>
</tr>
<tr>
<td>Query Reference Speed</td>
<td>15</td>
<td>xE502</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported Outgoing Messages</th>
<th>Service ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Reference Steering</td>
<td>15</td>
<td>xE501</td>
</tr>
<tr>
<td>Report Reference Speed</td>
<td>15</td>
<td>xE503</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Services</th>
<th>Service ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Planner</td>
<td>22</td>
</tr>
<tr>
<td>Velocity State Sensor</td>
<td>42</td>
</tr>
<tr>
<td>Primitive Driver</td>
<td>33</td>
</tr>
<tr>
<td>Driver Modeling</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Service Connections</th>
<th>Service ID</th>
<th>Message</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Velocity State</td>
<td>42</td>
<td>x4404</td>
<td>40Hz</td>
</tr>
<tr>
<td>Report Wrench Efforts</td>
<td>33</td>
<td>x4405</td>
<td>40Hz</td>
</tr>
</tbody>
</table>

The configuration in Table 5-5 indicates that the MPC component identifies itself with a component ID of 15. This particular ID was set to describe the MPC service in coordination with TORC Robotics so that it may be identified by other software components within the platform JAUS network. The remainder of the MPC JAUS address is dynamically determined by the NVICPU Node Manager; however the instance address is typically 1 as only one instance of the MPC component is executed at any given time.

The supported incoming messages of the MPC Component include the Set Motion Profile, Report Velocity State, Report Wrench Efforts, Query Reference Steering and Query Reference Speed messages. Recalling from Section 5.5.2.3, the first three messages provide the current motion profile, actual steering wheel angle, and actual speed data that is utilized by the MPC algorithm. The last two messages enable the Driver Modeling component to query the reference steering and speed values for driver modeling uses. The supported incoming message specification configures the JTK
component VIs to receive these messages internally and expose them to the MPCMessageHandler VI of the MPC component. Additionally, the Motion Planner component uses the supported incoming message information to determine which components within the JAUS network require the Set Motion Profile message. This procedure must be taken as the Set Motion Profile message is a command class message and cannot be requested as a service connection by the MPC component. The supported outgoing messages of the MPC component include the Report Reference Steering and Report Reference Speed messages. Recalling from Section 5.5.2.3, these messages are sent in reply to Query Reference Steering and Query Reference Speed messages to report the instantaneous reference steering and reference speed values.

The supported incoming messages of the Driver Modeling Component include the Report Velocity State, Report Wrench Efforts, Report Reference Steering and Report Reference Speed messages. Recalling from Section 5.5.2.6, the first two messages provide the actual steering wheel angle and actual speed data that is utilized by the DriverModeling() algorithm. The last two messages provide the instantaneous reference steering and speed values to the Driver Modeling component from the MPC component for driver modeling requirements. The supported incoming message specification configures the JTK component VIs to receive these messages internally and expose them to the DMMessageHandler VI of the Driver Modeling component.
Table 5-6: Driver Modeling Component Configuration

<table>
<thead>
<tr>
<th>Supported Incoming Messages</th>
<th>Service ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Velocity State</td>
<td>100</td>
<td>x4404</td>
</tr>
<tr>
<td>Report Wrench Efforts</td>
<td>100</td>
<td>x4405</td>
</tr>
<tr>
<td>Report Reference Steering</td>
<td>100</td>
<td>xE501</td>
</tr>
<tr>
<td>Report Reference Speed</td>
<td>100</td>
<td>xE503</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported Outgoing Messages</th>
<th>Service ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Reference Steering</td>
<td>100</td>
<td>xE500</td>
</tr>
<tr>
<td>Query Reference Speed</td>
<td>100</td>
<td>xE502</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Services</th>
<th>Service ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity State Sensor</td>
<td>42</td>
</tr>
<tr>
<td>Primitive Driver</td>
<td>33</td>
</tr>
<tr>
<td>MPC</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Service Connections</th>
<th>Service ID</th>
<th>Message</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Velocity State</td>
<td>42</td>
<td>x4404</td>
<td>40Hz</td>
</tr>
<tr>
<td>Report Wrench Efforts</td>
<td>33</td>
<td>x4405</td>
<td>40Hz</td>
</tr>
<tr>
<td>Report Reference Steering</td>
<td>15</td>
<td>xE501</td>
<td>8Hz</td>
</tr>
<tr>
<td>Report Reference Speed</td>
<td>15</td>
<td>xE503</td>
<td>8Hz</td>
</tr>
</tbody>
</table>

The required services and service connections are utilized by the WaitForComponent and EstablishServiceConnections VIs of the Driver Modeling component to ensure that the necessary message streams are prepared before normal operation begins. The configuration lists the Velocity State Sensor, Primitive Driver, and MPC as required services because they provide the Set Motion Profile, Report Velocity State, Report Wrench Efforts, Report Reference Steering, and Report Reference Speed messages that are vital to the operation of the Driver Modeling component. The configuration also defines required service connections to the Report Velocity State and Report Wrench Efforts messages with a rate of 40Hz and service connections to the Report Reference Steering and Report Reference Speed messages with a rate of 8Hz. These service connections ensure that constant streams of current actual and reference steering and speed data are provided to the Driver Modeling component.
5.6: Performance Analysis

This subsection presents a series of experiments that investigate the performance of the ANVID and the comprehensive Blind Driver Challenge® system. The discussions begin with an analysis of the Driver Modeling component of the ANVID software. This analysis examines both the software timing characteristics as well as the accuracy of the driver model itself. A second analysis is then conducted on the Model Predictive Controller component of the ANVID software. Similarly to the driver modeling analysis, the software timing of the Model Predictive Controller is first examined. The analysis then continues with an investigation of the reference signal continuity. Finally, the navigational abilities of the driver are analyzed to determine the overall performance of the comprehensive Blind Driver Challenge® system supplemented with the ANVID software.

5.6.1: Driver Modeling

This subsection presents and discusses the results from various experiments conducted to investigate the performance of the Driver Modeling component of the ANVID software. Two separate experiments were conducted. The first experiment analyzes the software timing characteristics of the Driver Modeling component to examine the efficiency and adaptation rate of the models and the training process. The second experiment examines the accuracy of the driver models in predicting the actions of the human driver as a dynamical system.

5.6.1.1: Software Timing

It was discussed in Section 5.3.2.3 that software timing is a critical factor in the performance of the ANVID software. The rate at which the driver model is trained online directly influences the adaptivity rate of the model. Furthermore, the rate at which the driver model can be evaluated directly influences how much processing time is required by the optimization process within the Model Predictive Controller. Thus, the processing time required by the various components of the Driver Modeling process will be examined.

The timing experiment was conducted by operating the Driver Modeling component for 10 minutes and measuring the average processing time taken by each sub-component. It is important to note that the processing time measured in this experiment is dependent upon the computational platform that executes the Driver Modeling software. In this experiment, the Non-Visual Interface Computer described in Section 3.5 was utilized to execute the Driver Modeling software. The results from this experiment are presented in Table 5-7.
Table 5-7: Average processing times of Driver Modeling algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avg. Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrainModel()</td>
<td>15.943ms</td>
</tr>
<tr>
<td>EvaluateModel()</td>
<td>0.237ms</td>
</tr>
<tr>
<td>MEKA()</td>
<td>6.332ms</td>
</tr>
<tr>
<td>ShieldModel()</td>
<td>1.211ms</td>
</tr>
</tbody>
</table>

Recalling from Section 5.4.3, the main algorithm responsible for training the driver models is the TrainModel() algorithm. The Driver Modeling component runs two instances of the TrainModel() algorithm in parallel to simultaneously train the steering and speed TSPNN driver models. Table 5-7 indicates that the TrainModel() algorithm takes approximately 15.943ms of processing time per single training iteration. This means that the driver steering and speed models are continuously updated online at a rate of approximately 62.7Hz. Such a high iteration rate enables the steering and speed TSPNN driver models to adapt very quickly to changes in the driver’s dynamics.

Table 5-7 also describes the processing times associated with the sub-algorithms of the main TrainModel() algorithm. The EvaluateModel() algorithm defined in Section 5.4.4 only requires 0.237ms to evaluate the steering and speed TSPNN models and output predictions of future driver actions. This low processing time is particularly advantageous in regard to the optimization processing time later discussed in Section 5.6.2.1. The MEKA() algorithm defined in Section 5.4.5 is able to evaluate the local EKFs and provide a single update to the weights of each TSPNN model in 6.332ms. The ShieldModel() algorithm defined in Section 5.4.6 only requires 1.211ms to determine if the model is diverging and recover from a poor weight update.

The three sub-algorithms only account for 7.78ms of the 15.943ms of processing time demonstrated by the TrainModel() algorithm. Thus, the remaining 8.163ms is attributed to training buffer management and global variable updates. Each time the EvaluateModel() algorithm is called, the entire input buffer must copied into memory and passed into the TSPNN model. Each time the MEKA() algorithm is called, the entire output buffer must be copied in a similar manner. Furthermore, the ShieldModel() algorithm makes a second, internal call of the EvaluateModel() algorithm and requires the input buffer to be copied for a second time. While the extraction of data from the training buffer does require a certain amount of processing time, a significant amount of processing time is required to update the global variables. The global variables are essentially network-published storage locations that can be accessed by other pieces of software such as the ModelPredictiveController() algorithm. Each time the model is updated and published to the global variable network storage location, 3.731ms of processor time is consumed.
5.6.1.2: Model Accuracy
The accuracy of the steering and speed TSPNN driver models directly influences the performance of the ANVID Model Predictive Controller. The MPCr requires accurate prediction models because the reference signals are optimized solely on the predicted actions given by the model. In order to investigate the accuracy of the steering and speed models, an experiment was conducted to monitor and compare the actual actions of a driver and the associated predicted actions given by the TSPNN models. A separate analysis will be given on the driver steering model and the driver speed model.

5.6.1.2.1: Steering TSPNN Driver Model
Recalling from Section 5.3.2.2, the driver steering model predicts a future horizon of steering actions that is 750ms long. In order to examine the accuracy of the 750ms long horizons, an experiment was conducted to make a prediction with the current steering model every 750ms while a candidate driver was navigating a complex road course. It is important to remember that, even though the predictions in this experiment are only made every 750ms, the online real-time MEKA training algorithm updates the model at 62.7Hz. Thus, the model was continuously adapting to the driver over the length of the experiment. The results of this experiment are given in Figure 5-18.

![Figure 5-18: Accuracy of the steering TSPNN driver model with respect to chain predictions at 750ms intervals.](image)

Figure 5-18 demonstrates that the steering TSPNN model is able to predict the driver's steering actions with a certain degree of accuracy over the length of each 750ms prediction horizon. The mean absolute error of the predicted actions with respect to the actual actions from Figure 5-18 is only 2.06°. Considering the highly nonlinear and stochastic nature of the driver, this level of modeling accuracy is quite satisfactory. The largest errors typically occur at the sharper stationary points in the driver actions, which suggest that the model is not complex enough to recreate sharp curvatures in a short
period of time. Increasing the complexity of the model by adding additional neurons and hidden layers may mitigate this issue but will also increase the amount of processing time required for model evaluation and training.

Although the steering TSPNN driver model predicts 750ms of future actions, only a fraction of that data is actually utilized by the Model Predictive Controller. Recalling from Section 5.3.2.2, the motion profiles are sent by the TORC AutonoNav™ system every 125-500ms. Each time a new motion profile is received, the MPCr calculates an suboptimal reference signal based on the commanded actions from the motion profile and the predicted driver actions. At the end of each MPCr iteration, a reference signal is provided that is predicted to elicit a driver response closest to the commanded actions of the motion profile. Each suboptimal reference signal has an associated predicted response that the optimality was decided upon. The predicted response contains 750ms of data; however portions of this chain are overlapped similarly to the discussion in Section 5.3.3.4. These overlapped chains are the only useful set of prediction data as each 750ms chain maintains the same validity limit as its associated motion profile. Figure 5-19 depicts a comparison between the predicted and actual driver actions using the overlapped chains.

![Figure 5-19: Accuracy of the steering TSPNN driver model with respect to overlapped chain predictions from the Model Predictive Control process.](image)

Figure 5-19 shows that the overall accuracy of the driver model is actually improved in the MPCr implementation as later data in the prediction horizon are typically discarded due to overlapping. This seems valid as predicted outputs of a dynamical system tend to diverge from the actual output at larger prediction horizons. Since the MPCr overlaps latter predictions in the horizon, only the more accurate prediction data from earlier in the horizon is utilized. Thus, the MPCr is able to use the steering TSPNN driver model to make highly accurate predictions of the driver’s response.
5.6.1.2.2: Speed TPSNN Driver Model

Similarly to the driver steering model, the driver speed model predicts a future horizon of steering actions that is 750ms long. In order to examine the accuracy of the 750ms long horizons, an experiment was conducted to make a prediction with the current speed model every 750ms while a candidate driver was navigating a complex road course using the Blind Driver Challenge® simulator (Section 2.5). It is important to remember that, even though the predictions in this experiment are only made every 750ms, the online real-time MEKA training algorithm updates the model at 62.7Hz. Thus, the model was continuously adapting to the driver over the length of the experiment. The results of this experiment are given in Figure 5-20.

![Figure 5-20: Accuracy of the speed TSPNN driver model with respect to chain predictions at 750ms intervals.](image)

Figure 5-20 demonstrates that the speed TSPNN model is able to predict the driver’s speed actions with an acceptable degree of accuracy over the length of each 750ms prediction horizon. The mean absolute error of the predicted actions with respect to the actual actions from Figure 5-20 is only 0.05m/s. Considering the highly nonlinear and stochastic nature of the driver, this level of modeling accuracy is quite satisfactory. Similarly to the driver steering model, the largest errors typically occur at the sharper stationary points in the driver actions. Increasing the complexity of the model by adding additional neurons and hidden layers may mitigate this issue but will also increase the amount of processing time required for model evaluation and training.

Although the speed TSPNN driver model predicts 750ms of future actions, only a fraction of that data is actually utilized by the Model Predictive Controller. Recalling from Section 5.3.2.2, the motion profiles are sent by the TORC AutonoNav™ system every 125-500ms. Each time a new motion profile is received, the MPCr calculates a suboptimal reference signal based on the commanded actions from the motion profile
and the predicted driver actions. At the end of each MPCr iteration, a reference signal is provided that is predicted to elicit a driver response closest to the commanded actions of the motion profile. Each suboptimal reference signal has an associated predicted response that the optimality was decided upon. The predicted response contains 750ms of data; however portions of this chain are overlapped similarly to the discussion in Section 5.3.3.4. These overlapped chains are the only useful set of prediction data as each 750ms chain maintains the same validity limit as its associated motion profile. Figure 5-21 depicts a comparison between the predicted and actual driver actions using the overlapped chains.

![Graph showing speed comparison](image)

**Figure 5-21:** Accuracy of the speed TSPNN driver model with respect to overlapped chain predictions from the Model Predictive Control process.

Figure 5-21 shows that the overall accuracy of the driver model is actually improved in the MPCr implementation as later data in the prediction horizon are typically discarded due to overlapping. This seems valid as predicted outputs of a dynamical system tend to diverge from the actual output at larger prediction horizons. Since the MPCr overlaps latter predictions in the horizon, only the more accurate prediction data from earlier in the horizon is utilized. Thus, the MPCr is able to use the speed TSPNN driver model to make highly accurate predictions of the driver's response.

### 5.6.2: Model Predictive Controller

This subsection presents and discusses the results from various experiments conducted to investigate the performance of the Model Predictive Controller component of the ANVID software. Two separate experiments were conducted. The first experiment analyzes the software timing characteristics of the Model Predictive Controller component to examine the efficiency of the optimization process. The second experiment examines the continuity of the reference signal that is necessary according to the functional requirements in Section 5.1.1.
5.6.2.1: Software Timing

It was discussed in Section 5.3.3 that software timing is a critical factor in the performance of the ANVID software. The optimization processes utilized by the Model Predictive Controller must be able to find suboptimal reference signals in a minimal amount of time to avoid increasing the pure time delay of the system. Thus, the processing time required by the various components of the Model Predictive Controller process will be examined.

A timing experiment was conducted by operating the Model Predictive Controller component for 5 minutes and measuring the average processing time taken by each sub-component. It is important to note that the processing time measured in this experiment is dependent upon the computational platform that executes the Model Predictive Controller software. In this experiment, the Non-Visual Interface Computer described in Section 3.5 was utilized to execute the MPCr software. The results from this experiment are presented in Table 5-8.

Table 5-8: Average processing times of Model Predictive Control algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avg. Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptimizeReferenceSignals()</td>
<td>38.379ms</td>
</tr>
<tr>
<td>QNO()</td>
<td>14.343ms</td>
</tr>
<tr>
<td>ObjectiveFunction()</td>
<td>0.854ms</td>
</tr>
</tbody>
</table>

Recalling from Section 5.4.8, the OptimizeReferenceSignals() algorithm is responsible for determining the suboptimal reference steering and speed signals each time a new motion profile is received from the TORC AutonoNav™ system. The processing time taken to calculate the suboptimal reference signals is injected into the Blind Driver Challenge® system as a pure time delay. Thus, the algorithm is required to consume minimal amounts of processing time. Table 5-8 indicates that this particular algorithm in fact requires only 38.379ms of processing time on average to calculate both suboptimal reference signals once a new motion profile is received. The definition of the OptimizeReferenceSignals() algorithm in Section 5.4.8 indicates that it calls the QNO() sub-algorithm twice within each 38.379ms long iteration. Table 5-8 shows that the QNO() algorithm only requires 14.343ms on average to determine a suboptimal solution for each of the steering and speed reference signals. Although the QNO() algorithms called by OptimizeReferenceSignals() are in parallel, it seems that the quad-core processor of the NVICPU does not fully execute the two processes in parallel. A single iteration of the OptimizeReferenceSignals() algorithm takes just short of twice the processing time of a single QNO() algorithm. This seems valid as there are a significant number of processes running in parallel throughout the ANVID() algorithm. The total number of processes must be distributed amongst the four physical cores of the NVICPU and thus some of the processes are partially or fully run in series.
In the QNO() algorithm, the ObjectiveFunction() algorithm is actually used to serve two separate purposes. The first purpose is to predict the driver’s actions for a candidate reference signal solution and determine the cost associated with that particular candidate. The second purpose is to internally calculate the gradient of the objective function evaluated at that particular candidate. In Section 5.3.3.2, it was explained that the gradient calculation must consume minimal amounts of processing time as it can be executed a significant number of times within a single optimization process. The same requirement is applied to the evaluation of the objective function and thus the determination of the candidate reference signal cost. The timing requirement for the objective function evaluation is actually more constrained as the evaluation is called not only in the optimization iteration, but also in the step size calculation sub-iterations.

Table 5-8 indicates that each call of the ObjectiveFunction() algorithm requires only 0.854ms of processing time to evaluate both the objective function (5.9) and the objective function gradient (5.18). It was shown previously in Table 5-7 that the evaluation of the neural network model within the cost calculation only requires 0.237ms of processing time on average. Thus, 0.617ms of the ObjectiveFunction() processing time are attributed to calculating the mean-squared error (5.9) and evaluating the objective function gradient (5.18). These low processing times permit the QNO() algorithms to calculate a suboptimal reference signal in only 14.343ms even though the ObjectiveFunction() algorithm is called a significant number of times per optimization.

The iteration count statistics for a single QNO() algorithm execution are recorded in Table 5-9 to provide a better understanding of why the low processing times are required for the ObjectiveFunction() algorithm. According to (5.41)-(5.46), the objective function cost and the associated gradient must be calculated each time a new step size is generated and evaluated against the Armijo, Wolfe, and Strong Wolfe conditions. Since the ObjectiveFunction() algorithm evaluates both the objective function and its gradient with respect to a candidate reference signal, it is called each time a new step signal is generated. The results in Table 5-9 show that, each optimization requires 2 iterations of (5.11)-(5.14) and 4 sub-iterations of (5.41)-(5.46) on average. Thus, the ObjectiveFunction() algorithm is typically executed 8 times. However, this number can significantly increase as maximum counts of 9 iterations and 50 sub-iterations were also found. Thus, the low 0.854ms processing time of the ObjectiveFunction() algorithm is required to minimize the overall processing time of the OptimizeReferenceSignals() algorithm even though it may be required to execute a significant number of times.

Table 5-9: QNO() iteration count statistics.

<table>
<thead>
<tr>
<th>Iteration Type</th>
<th>Min. Iterations</th>
<th>Avg. Iterations</th>
<th>Max Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Iteration</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Step Size Iteration</td>
<td>1</td>
<td>4</td>
<td>50</td>
</tr>
</tbody>
</table>
5.6.2.2: Reference Signal Continuity

According to the functional requirements outlined in Section 5.1, one of the more significant functionalities that must be provided by the ANVID software is that the reference signals must be in a form that the driver can understand and reasonably interact with through the Non-Visual Interface System. Thus, the reference signals must be smooth and exhibit continuity to avoid confusing the driver with rapidly varying non-visual stimuli. An experiment was conducted to measure the typical suboptimal reference steering wheel angle and speed signals generated by the Model Predictive Controller of the ANVID software and examine them for continuity. The results of this experiment are provided in Figure 5-22 and Figure 5-23 below.

Figure 5-22: Typical suboptimal reference steering wheel angle signal generated by the Model Predictive Controller component of the ANVID software.

Figure 5-23: Typical suboptimal reference speed signal generated by the Model Predictive Controller component of the ANVID software.
The typical reference signals depicted in these figures show that the Model Predictive Controller is in fact capable of generating smooth, continuous reference signals. The driver responses associated with each reference signal generated for this particular experiment are included on each of these plots. The response data demonstrates that the driver is able to understand and interact with the generated reference signals and can successfully track them using the FLFT and TE operation configurations defined in Table 3-2 and Table 3-7, respectively. An in-depth analysis of how well the driver is actually able to navigate a road course by tracking these optimized reference signals will be provided in the next subsection.

5.6.3: Comprehensive System Performance
At this point, it has been determined that the ANVID is able to supply the driver with smooth reference steering and speed signals that are generated in a timely manner. However, the reference generation performance does not completely describe the abilities of the comprehensive Blind Driver Challenge® system. This subsection aims to characterize the performance of the BDC system as a whole by investigating the driver’s ability to navigate a complex road course under complete blindness. The results from this investigation reflect not only on the abilities of the ANVID, but also on the perception and motion planning abilities of the Blind Driver Challenge® research platform.

5.6.3.1: Experimental Setup
In this experiment, a blindfolded driver was asked to navigate a simulated complex road course utilizing the complete Blind Driver Challenge® system with ANVID software. The purpose of the experiment was to record the driver’s trajectories along the road course and analyze his lane keeping, speed tracking, and obstacle avoidance abilities using the non-visual stimuli provided by the BDC system. While the driver was navigating the road course, the trajectory of the vehicle was recorded through the logging of global position points from a virtual Inertial Navigation System similar to that described in Section 2.2.1. The reference steering and speed signals as well as the driver’s associated responses were also recorded to match with the vehicle trajectories for a more detailed analysis.

The road course utilized in this experiment was a portion of the Daytona International Speedway (DIS) similar to that of the Rolex 24 demonstration course. The simulated course, pictured in Figure 5-24, is a 1.4mi long paved course composed of various straightaways and curvatures. This road course was chosen for the experiment because its diverse composition of roadways enabled a wide array of driving scenarios to be tested. While the actual width of the course is approximately 30-50ft, the RNDF (Section 2.6.1) created for the simulated course superimposes virtual 18ft lanes over the center of the track.
Figure 5-24: Road course utilized for comprehensive Blind Driver Challenge® system testing.

Figure 5-24 additionally describes the speed limits of each course section and identifies the locations of start and stop points. The course begins with a long straightaway that sweeps slightly to the left with a speed limit of 20mph. A medium curvature turn follows and the speed is decreased to 15mph as the driver navigates the turn and a chicane. The speed limit increases to 20mph after the chicane once the driver enters the second straightaway. The first hairpin turn follows with a decreased speed of 10mph throughout the length of the turn. After exiting the turn, the driver enters a long, left-handed dogleg with a 20mph speed limit. The second hairpin turn then follows also with a decreased speed of 10mph throughout the length of the turn. As the driver exits the turn, he enters a 15mph transition zone into the final 20mph straightaway. Next, the driver encounters a sharp left-hand turn with a low speed limit of 5mph. After the turn, the course ends with a stop point located at the finish line at which the driver must accurately bring the vehicle to a complete stop.

The navigation experiment was conducted with a blindfolded, sighted driver that navigated the described course through the Blind Driver Challenge simulation platform discussed in Section 2.5. The sighted driver is a fully trained and experienced non-visual interface user under complete blindness with a blindfold. The results presented in this subsection are taken from this particular driver as he best demonstrates the capabilities of the comprehensive Blind Driver Challenge® system augmented with the
ANVID software. The data collected and presented in these results cooperates with the Virginia Tech Institutional Review Board (IRB) Human Subject Research Protocol.

5.6.3.2: Results

The results obtained from the comprehensive system performance experiment will be systematically presented and discussed in this subsection. First, a high level analysis of the overall performance will be conducted for the entire length of the track. Following this discussion will be a series of further analyses that break down the track into separate sections to perform a more thorough investigation of the driver’s navigation performance. Each analysis will examine not only the vehicle trajectory in relation to the course, but will also examine lateral lane and speed deviations with correspondences in the steering and speed tracking responses. The lateral lane deviations are calculated in a similar manner to Section 4.4.3.2 using (4.5)-(4.7). The lateral lane deviation plots additionally show the maximum allowable lateral deviation for the particular course section that the driver is navigating in. The maximum allowable values, shown with a dashed line, are determined by the width of the driving lane and the width of the vehicle itself. These values show the maximum lateral deviation that may occur before the wheels of the vehicle travel outside of the driving lane.

Figure 5-25 depicts the trajectory navigated by the driver over the entire length of the DIS simulation course. A quick observation of this figure indicates that the driver is generally able to navigate the full course without any significant lane deviations. The lateral lane deviations are examined in much more detail through Figure 5-26. This figure proves that the driver never exceeds the maximum allowable lateral deviation and stays completely within the driving lane over the entire length of the course. The driver comes as close as 2.6ft from laterally exiting the lane in five different course locations, each of which will be described in more detail in later subsections. The mean absolute lateral deviation (MALD) over the entire course is only 1.33ft, proving that the driver is able to utilize the comprehensive Blind Driver Challenge® system to navigate a road with satisfactory accuracy.

The minimal lateral deviations can be partially attributed to the driver’s ability to accurately track the reference steering wheel angle signal, as shown in Figure 5-27. Over the length of the course, the motions planned by the TORC AutonoNav™ system required many complex steering maneuvers to stay within the driving lane. Keeping in mind the time scale, Figure 5-27 shows that the ANVID successfully transformed the complex steering maneuvers into smooth reference steering signals that the driver could interact with. Figure 5-28 indicates that the driver was able to track the reference signal with typically low error throughout the entire experiment using the FFLT DOC defined in Table 3-2 and chosen in Section 3.1.4.3. The maximum error exhibited by the driver was approximately -59°, while the mean absolute error was only 6.2°. The combination of the planning abilities of the TORC AutonoNav™ system and the low
MAE of the driver’s steering wheel angle response is what enabled the driver to navigate the entire course with only 1.33 ft of average lateral lane deviation.

Figure 5-25: Trajectory (blue) navigated by a blind driver on the DIS simulation course characterized in Figure 5-24. The vehicle trajectory (blue) starts at 0,0 and ends at -533, 167. Lane edges (solid black) and lane centerline (dotted black) are also shown.

Figure 5-26: Lateral lane and speed deviations over the length of the entire DIS simulation course.
Figure 5-27: Reference steering wheel angle generated by the ANVID and the associated driver response over the entire DIS simulation course.

Figure 5-28: Instantaneous steering wheel angle error exhibited by the driver over the entire DIS simulation course.

The speed deviation, also shown in Figure 5-26, indicates that the driver is generally able to track the speed limits of each course section with maximum deviations of 20mph. It is important to recognize that the large deviations occur during transitions between course sections with different speed limits, as will be discussed in later subsections. Even with the transition deviations, the mean absolute speed deviation (MASD) is 3.28mph. This low speed deviation can be attributed to the driver’s ability to track the reference speed signals generated by the ANVID in coordination with the desired speeds specified by the TORC AutonoNav™ system. Figure 5-29 portrays the reference speed signal generated by the ANVID and how well the driver is able to track it using the TE SOC defined in Table 3-7. The tracking ability is further proven by the
plot of speed error over time in Figure 5-30. This plot indicates that the driver exhibited a maximum speed error of only 4.59 mph and a mean absolute error of only 1.51 mph. The MAE of the speed response differs from the MASD because the MASD relates the deviations to the section speed limit while the MAE relates the error to the reference speed, which is not always equal to the speed limit.

Figure 5-29: Reference speed generated by the ANVID and the associated driver response over the entire DIS simulation course.

Figure 5-30: Instantaneous speed error exhibited by the driver over the entire DIS simulation course.
These results prove that the blind driver is able to navigate the complex DIS simulation road course with acceptable levels of lateral lane and speed deviation. However, the results presented thus far generalize over the entire course and do not capture particular periods of interest that would help provide a more in-depth analysis of the driver's performance. Therefore, the course will be broken down into sections that can be investigated with more detail. The results of each course section are presented and discussed in the following sub-sections.

5.6.3.2.1: Starting Straightaway
The starting straightaway section, shown in Figure 5-31, includes the starting line of the course as well as a long straightaway that bears slightly to the left. As indicated by Figure 5-24, the speed limit for this particular course section is set at 20mph. The trajectory navigated by the blindfolded driver can be seen in the section map shown in Figure 5-31. A quick observation of the blue trajectory line proves that the driver is generally able to navigate the starting straightaway with a smooth trajectory that follows the dotted lane centerline.

![Figure 5-31: Trajectory navigated by the driver on the starting straightaway section of the DIS simulation course. Timing is indicated by the green crosses.](image)

A more detailed analysis of the driver's navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 5-32. Observation of the speed deviation shows that the driver is able to start the vehicle from standstill and reach the 20mph speed limit within approximately 16.5 seconds with negligible overshoot. Although the speed deviation exhibits large error while the driver begins to accelerate from a complete stop, the driver actually tracks the reference speed signal with low error. This is clearly exemplified in the speed tracking response and error shown in Figure 5-33 and Figure 5-34, respectively. Upon startup, the instantaneous speed error slowly increased to approximately 2.5mph over the first 5 seconds of
recorded data. The driver was able to recognize the error through the vibrations generated by the SpeedStrip interface on his middle thighs and make the appropriate corrections with tracking performance similar to that presented in Section 3.2.4. Once past the initial accelerations from a complete stop, the driver exhibited only 1.86mph of average absolute speed error.

Figure 5-32: Lateral lane and speed deviations over the starting straightaway section of the DIS simulation course.

Figure 5-33: Reference speed generated by the ANVID and the associated driver response over the starting straightaway section of the DIS simulation course.
Throughout the entire course section, the driver is able to keep the lateral lane deviation within the maximum limits and thus keep the tires on both sides of the vehicle fully within the lane boundaries. The maximum lane deviation occurs at approximately 60s and can be seen in the trajectory plot on Figure 5-31. Although the deviation at this point is as much as 2.25ft, the vehicle still remains within the driving lane with 4ft of clearance between the left wheels and the left lane boundary. The deviation occurs as a driver enters the first turn of the simulation course and is the result of how turns are navigated by the TORC AutonoNav™ motion planning system. It was shown in a significant amount of cases in Section 4.4.3 that lateral deviations cutting the corners of turns are not accompanied with steering wheel angle deviations. Since the driver was controlling the steering wheel angle correctly in all of these situations, it was concluded that the AutonoNav™ system plans trajectories that cut the corners of turns. This is further proven in this course section as the steering tracking and error plots shown in Figure 5-35 and Figure 5-36 indicate no significant periods of large error leading up to the 2.25ft lateral deviation. Aside from this large deviation at the first turn, the driver is able to keep the mean absolute lateral deviation (MALD) at an impressive 0.65ft over the entire starting straightaway section.

Using the data from the steering wheel angle error plot in Figure 5-36, the mean absolute steering wheel angle error is found to be 3.8°. The MAE of the steering wheel angle is relatively large compared to the typical MAEs seen in Section 3.1.4.3 due to the interaction between the driver and the TORC AutonoNav™ motion planner of the dual closed-loop system. This combination causes the reference signal to continuously oscillate at low frequencies when compared to the non-periodic arbitrary signals used in the experiments of Section 3.1.4.3. These oscillations require continuous corrective action from the driver and thus increase the MAE over the length of the timeframe.
Figure 5-35: Reference steering wheel angle generated by the ANVID and the associated driver response over the starting straightaway course section.

Figure 5-36: Instantaneous steering wheel angle error exhibited by the driver over the starting straightaway section of the DIS simulation course.

5.6.3.2.2: Medium Curvature Turn
The medium-curvature turn section, shown in Figure 5-37, is a single turn with a maximum curvature of 0.026rad and equivalent turning radius of 124ft. This course section additionally includes a small chicane shortly after the single turn. The course layout in Figure 5-24 indicates that this course section sets a 15mph speed limit due to the increased curvature of the lanes. The trajectory navigated by the blindfolded driver can additionally be seen in the section map shown in Figure 5-37. A quick observation of the blue trajectory line proves that the driver is generally able to navigate the medium-curvature turn with a smooth trajectory that follows the dotted lane centerline closely.
Figure 5-37: Trajectory navigated by the driver on the medium-curvature turn section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly.

A more detailed analysis of the driver’s navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 5-38. Observation of the speed deviation shows that the driver is able to remain within 3mph of the section speed limit over the entire section. Figure 5-39 shows that the interaction between the driver’s response and the reference signal generated by the ANVID and AutonoNav™ system cause the driver to typically exhibit positive error and travel at speeds slightly lower than the speed limit for the first 30s of the section. The speed error given in Figure 5-40 proves that the driver tracked the reference speed with satisfactory accuracy; thus the
length of 3mph deviation is solely attributed to how the reference reacted against the responses of the driver. This is further proven by the fact that the mean absolute speed deviation (MASD) was 3mph while the mean absolute speed error was only 1.72mph.

Figure 5-38: Lateral lane and speed deviations over the medium-curvature turn section of the DIS simulation course.

Figure 5-39: Reference speed generated by the ANVID and the associated driver response on the medium-curvature turn section of the DIS simulation course.
Figure 5-40: Instantaneous speed error exhibited by the driver over the medium-curvature turn section of the DIS simulation course.

The lateral deviation also shown in Figure 5-38 indicates that the driver was able maintain at least 2.34ft of clearance between the lane boundaries and the vehicle tires for the entire duration of the course section. The maximum deviation from the lane centerline never surpassed ±3.9ft and exhibited a mean absolute lateral deviation of 1.81ft. Closer examination of the lateral deviation and the vehicle trajectory shows that the driver tends to drive inside of the lane centerline over the length of the turn, leading to an increased MALD. The steering wheel angle tracking and error plots given in Figure 5-41 and Figure 5-42 demonstrate that the driver does not exhibit significant error over the length of the turn and thus the tendency to drive inside of the curve is once again a result of the TORC AutonoNav™ planned trajectories.

The steering wheel angle tracking and error plots additionally indicate that the driver actually exhibits a significant amount of control effort throughout the medium-curvature turn section. The reference steering signal generated by the AutonoNav™ and the ANVID hardly remains constant over the length of the course section. It seems that the interaction between the motion planner and the driver’s steering response causes the reference signal to exhibit large steering wheel angles to match the curvature of each turn that are superimposed with smaller, higher frequency adjustments that are meant to make slight curvature corrections while in the turn. It is these small adjustments that enable the driver to accurately stay within the lane boundaries even while navigating turns with considerable curvature. Although there was a significant amount of control effort required from the driver, he was still able to exhibit a mean absolute steering wheel angle error of 8.9°.
Figure 5-41: Reference steering wheel angle generated by the ANVID and the associated driver response on the medium-curvature turn course section.

Figure 5-42: Instantaneous steering wheel angle error exhibited by the driver over the medium-curvature turn section of the DIS simulation course.

5.6.3.2.3: Hairpin Turn #1
The hairpin turn #1 section, shown in Figure 5-43, is composed of a short straightaway and a single turn with a maximum curvature of 0.04rad (74ft turning radius). The course layout in Figure 5-24 indicates that this course section enforces a 20mph speed limit on the straightaway and a 10mph limit on the turn due to the increased curvature of the lanes. The trajectory navigated by the blindfolded driver can additionally be seen in the section map shown in Figure 5-43. A quick observation of the blue trajectory line indicates that the driver is able to navigate the hairpin turn and is generally able to track the lane centerline. However, slight deviations that cut the inside of the right-hand turn can be observed and will be discussed in the following paragraphs.
Figure 5-43: Trajectory navigated by the driver on the hairpin turn #1 section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly.

The lateral lane and speed deviation exhibited by the driver over this course section is displayed in Figure 5-44. The speed deviation indicates that the driver was able to slowly accelerate and reach the 20mph speed limit in the straightaway and quickly decelerate to the 10mph speed limit in preparation for the hairpin turn. The deceleration heading into the hairpin turn required only 3.12s and exhibited less than 2mph of overshoot. After reaching each of the specified speed limits, the driver demonstrated a tendency to travel at a slightly lower rate of speed and was forced to make two corrections to accelerate the vehicle and return to an acceptable speed. This is clearly demonstrated in the speed tracking and instantaneous error plots provided in Figure 5-45 and Figure 5-46, respectively. Although two corrections were required, the driver was able to quickly make the necessary accelerations and never exhibited speed error larger than ±4.5mph. The mean absolute speed error for the hairpin turn #1 section was found to be only 1.3mph.
Figure 5-44: Lateral lane and speed deviations over the hairpin turn #1 section of the DIS simulation course.

Figure 5-45: Reference speed generated by the ANVID and the associated driver response on the hairpin turn #1 section of the DIS simulation course.

Figure 5-46: Instantaneous speed error exhibited by the driver over the hairpin turn #1 section of the DIS simulation course.
The lateral lane deviations also shown in Figure 5-44 demonstrate that the driver was able to maintain deviations from the centerline within ±4ft over the entire hairpin turn #1 section. The largest deviations occur at the 150s, 163s, and the 176s mark and can be observed in the trajectory plotted in Figure 5-43. Examination of the steering tracking and instantaneous error plots in Figure 5-47 and Figure 5-48 suggest that all three of these deviations are not caused by a lapse in tracking performance from the driver. The timeframes before and at each of these deviation occurrences exhibit no significant length or amount of steering wheel angle error. The trajectory plot in Figure 5-43 demonstrates that the deviations at 163s and 176s cut inside of the turn. Once again, these types of deviations are attributed to the manner in which the TORC AutonoNav™ system navigates through turns. However, the deviation at 150s occurs just outside of the turn. Examination of Figure 5-47 shows that the reference signal requires a counter-clockwise turn of the steering wheel between 139s and 144s just before the right-hand turn wand while the vehicle is center in the lane. The small left-hand turn required by the reference signal may be the result of a small lapse in motion planning from the AutonoNav™ system just before the hairpin turn.

Over the entire course section, the driver exhibits a mean absolute lateral deviation of 1.68ft and a mean absolute steering error of 7.06°. Despite the large deviations previously discussed, the driver demonstrates the ability to track the reference steering wheel angle with satisfactory accuracy and is generally able to navigate the center of the driving lane. The only significant deviations occur as a result of the motion planning tendencies and still provide at least ±2.25ft of clearance between the lane edges and the vehicle tires.

![Figure 5-47: Reference steering wheel angle generated by the ANVID and the associated driver response on the hairpin turn #1 course section.](image)
Figure 5-48: Instantaneous steering wheel angle error exhibited by the driver over the hairpin turn #1 section of the DIS simulation course.

5.6.3.2.4: Left-Handed Dogleg
The left-handed dogleg section, shown in Figure 5-49, contains a long straightaway followed by a left-handed dogleg ending in a smaller straightaway. As indicated by Figure 5-24, the speed limit for this particular course section is set at 20mph over the length of the entire dogleg. The trajectory navigated by the blindfolded driver can additionally be seen in the section map shown in Figure 5-49. A quick observation of the blue trajectory line proves that the driver is generally able to navigate the starting straightaway with a smooth trajectory that follows the dotted lane centerline.

Figure 5-49: Trajectory navigated by the driver on the left-handed dogleg section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly.
A more detailed analysis of the driver’s navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 5-50. Observation of the speed deviation shows that the driver is able to accelerate and reach the 20mph speed limit within approximately 15s with no overshoot. Recalling from Section 3.2.1, the speed control is secondary and it is not necessary for the driver to immediately accelerate the vehicle when signaled by SpeedStrip non-visual stimuli. The speed deviation data as well as the speed tracking data provided in Figure 5-51 demonstrates that the driver was able to hold a constant speed at 3mph below the 20mph speed limit. Figure 5-52 indicates that the driver tended to driver at a slower velocity than specified by the reference speed signal and was forced to make several small corrective accelerations along the dogleg. The driver exhibited a mean absolute speed deviation of 3.58mph and a mean absolute tracking error of 1.57mph. Examination of the reference signal in Figure 5-51 shows that the reference was continuously generated at approximately 19mph rather than 20mph, which accounts for a partial amount of the MASD and MAE difference.

Figure 5-50: Lateral lane and speed deviations over the left-handed dogleg section of the DIS simulation course.

Figure 5-51: Reference speed generated by the ANVID and the associated driver response on the left-handed dogleg section of the DIS simulation course.
Figure 5-52: Instantaneous speed error exhibited by the driver over the left-handed dogleg section of the DIS simulation course.

The lateral deviation also shown in Figure 5-50 indicates that the driver was able maintain at least 4.25ft of clearance between the lane boundaries and the vehicle tires for the entire duration of the course section. The maximum deviation from the lane centerline never surpassed ±2.58ft and exhibited a mean absolute lateral deviation of 0.84ft. Closer examination of the lateral deviation and the vehicle trajectory shows that the driver tends to drive inside of the lane centerline over the relatively short left-handed turn, leading to a somewhat increased MALD. The steering wheel angle tracking and error plots given in Figure 5-53 and Figure 5-54 demonstrate that the driver does not exhibit significant error over the length of the turn and thus the tendency to drive inside of the curve is once again a result of the TORC AutonoNav™ planned trajectories.

The steering wheel angle tracking and error plots additionally indicate that the driver was not required to put forward a significant amount of control effort throughout the left-handed dogleg. During the two straightaways of this course section, the ANVID and AutonoNav™ system only required 15-20° corrections in the steering wheel angle to keep the vehicle straight and centered within the driving lane. During the slight left-handed turn, the driver exhibited the most controller effort to create the necessary vehicle curvature while still remaining close to the lane centerline. The typical low-frequency curvature tracking superimposed with the higher frequency centering corrections can be clearly seen throughout the timeframe of this turn. Over the length of the entire left-handed dogleg, the driver demonstrated steering wheel angle errors within ±17° with a mean absolute error of only 4.52°.
Figure 5-53: Reference steering wheel angle generated by the ANVID and the associated driver response on the left-handed dogleg course section.

Figure 5-54: Instantaneous steering wheel angle error exhibited by the driver over the left-handed dogleg section of the DIS simulation course.

5.6.3.2.5: Hairpin Turn #2
The hairpin turn #2 section, shown in Figure 5-55, is composed of a single turn with a maximum curvature of 0.04rad (74ft turning radius) followed by a short straightaway. The course layout in Figure 5-24 indicates that this course section enforces a 10mph speed limit on the turn followed by a 15mph transition area to the 20mph straightaway. The trajectory navigated by the blindfolded driver can additionally be seen in the section map shown in Figure 5-55. A quick observation of the blue trajectory line indicates that the driver is able to navigate the hairpin turn and is generally able to track the lane centerline. However, slight deviations that cut the inside of the right-hand turn can be observed and will be discussed in the following paragraphs.
Figure 5-55: Trajectory navigated by the driver on the hairpin turn #2 section of the DIS simulation course. Timing is indicated by the green crosses and labeled accordingly.

The lateral lane and speed deviation exhibited by the driver over this course section is displayed in Figure 5-56. The speed deviation indicates that the driver was able to quickly decelerate to the 10mph speed limit posted for the hairpin turn and hold the speed constant throughout the remainder of the turn. The deviation also suggests that the driver ignored most of the 15mph speed limit in the transition area and only started to increase speed just short of the 20mph straightaway. This is clearly demonstrated in the speed tracking and instantaneous error plots provided in Figure 5-57 and Figure 5-58, respectively. These plots additionally demonstrate that the driver overshot the initial deceleration into the turn and the acceleration into the straightaway by as much as 4.5mph. The overshoot during deceleration is common as the driver is trained to decelerate more liberally for the sake of caution. Conversely, the driver is also trained to accelerate more conservatively for the same reason. The overshoot seen in the acceleration and contributing to the large negative error at 325 can be attributed to a momentary lapse in the driver's tracking. Although overshoot was present, the driver was able to quickly make the necessary corrections and never exhibited speed error larger than ±4.5mph. The mean absolute speed error for the hairpin turn #2 section was found to be only 1.69mph.
Figure 5-56: Lateral lane and speed deviations over the hairpin turn #2 section of the DIS simulation course.

Figure 5-57: Reference speed generated by the ANVID and the associated driver response on the hairpin turn #2 section of the DIS simulation course.

Figure 5-58: Instantaneous speed error exhibited by the driver over the hairpin turn #2 section of the DIS simulation course.
The lateral lane deviations also shown in Figure 5-56 demonstrate that the driver was able to maintain deviations from the centerline within ±3ft over the entire hairpin turn #2 section. The largest deviation occurs at the 294s mark and can be observed in the trajectory plotted in Figure 5-55. Examination of the steering tracking and instantaneous error plots in Figure 5-59 and Figure 5-60 suggests that this deviation is not caused by a lapse in tracking performance from the driver. The timeframes before and at each of these deviation occurrences exhibit no significant length or amount of steering wheel angle error. It should be noted that the 42° tracking error occurs just after the deviation at 294s and is a result a sudden change in the reference trajectory as the AutonoNav™ system attempts to re-center the vehicle. The 3ft deviation before and at the 294s mark exhibits a constant deviation to the right of the centerline in the right-handed hairpin turn. Thus, this situation is once again an example of how the AutonoNav™ system tends to navigate inside of the centerline on turns.

Once the driver exited the turn into the straightaway, the driver demonstrated a small continuous left-sided deviation followed by a continuous right-side deviation. The tracking and error plots shown in Figure 5-59 and Figure 5-60 indicate no significant lapses in tracking performance during the final 30s of the course section. In fact, the reference signal generated by the ANVID and AutonoNav™ system request an steering angle of 0° for most of this timeframe with only three short 15-20° corrections. Therefore, it seems as if the AutonoNav™ becomes more liberal at lane centerline tracking at higher speeds and/or with straighter course sections.

![Graph showing steering wheel angle and driver response](image)

**Figure 5-59:** Reference steering wheel angle generated by the ANVID and the associated driver response on the hairpin turn #2 course section.
Figure 5-60: Instantaneous steering wheel angle error exhibited by the driver over the hairpin turn #2 section of the DIS simulation course.

Over the entire course section, the driver exhibits a mean absolute lateral deviation of 1.35ft and a mean absolute steering error of 6.02°. Despite the deviations previously discussed, the driver demonstrates the ability to track the reference steering wheel angle with satisfactory accuracy and is generally able to navigate the center of the driving lane. The only significant deviations occur as a result of the motion planning tendencies and still provide at least ±3.25ft of clearance between the lane edges and the vehicle tires.

5.6.3.2.6: Finish

The final section of the DIS simulation course, shown in Figure 5-61, consists of a sharp left-handed turn followed by a stop point located at the course finish line. The left-handed turn exhibits a relatively high curvature of 0.035rad (92ft turning radius) with an expansion of the lane width to 25ft just after the turn. The course map in Figure 5-24 indicates that the speed limit for this particular course section is specified as 5mph due to the increased curvature of the turn. The ultimate purpose of this relatively small course section is to investigate the driver’s ability to come to a complete stop at or before a stop point designated in the RNDF map file. The location of the stop point is indicated on the section map in Figure 5-61 by a red point located at -533m, 166m. A quick observation of the blue trajectory line indicates that the driver is able to navigate the sharp left-handed turn and is generally able to track the lane centerline. However, slight deviations that cut the inside of the left-hand turn can be observed and will be discussed in the following paragraphs.
Figure 5-61: Trajectory navigated by the driver at the finish section of the DIS simulation course.

A more detailed analysis of the driver’s navigation performance is conducted through the lateral lane and speed deviation plot provided by Figure 5-62. Observation of the speed deviation shows that the driver is able to decelerate from 20mph to the 5mph speed limit in 5s with negligible overshoot. The driver was then able to remain within ±1mph of the section speed limit for the remainder of the course section prior to the stop point. Figure 5-63 and Figure 5-64 clearly demonstrate this behavior. The data from these plots additionally indicate that the driver typically exhibited negative speed error and traveled slightly faster than the 5mph speed limit; however he remained well within the lowest error tolerances of the SpeedStrip interface and maintained an MASD of only 1.31mph.

Figure 5-62: Lateral lane and speed deviations at the finish line section of the DIS simulation course.
Figure 5-63: Reference speed generated by the ANVID and the associated driver response throughout the finish section of the DIS simulation course.

Figure 5-64: Instantaneous speed error exhibited by the driver at the finish section of the DIS simulation course.

The blue trajectory line shown in Figure 5-61 does not clearly indicate if the driver was successfully able to bring the vehicle to a complete stop before the stop point was reached. Furthermore, the trajectory data shown in this figure represents the position of the rear axle of the vehicle and does not communicate the distance between the front of the vehicle and the stop point. Figure 5-65 provides this information by plotting the distance between the front of the vehicle and the stop point as a function of time. The figure shows this same distance function at two different scales to display finer results near the actual stop point. Observation of this figure indicates that the driver does not begin to decelerate the vehicle from the initial 5mph speed until approximately 16ft from the stop point. Due to the relatively low initial speed of 5mph, the driver was able to
bring the vehicle to a complete stop in a period of only 1.8s with only 1.73ft of distance between the front bumper and the stop point. It is important for the driver to be able to stop at or before the stop point to ensure that the vehicle does not overshoot stop signs or traffic lights and endanger the driver and other traffic.

Figure 5-65: Distance between the front of the Blind Driver Challenge® vehicle and the finish line stop point as a function of time at the finish section of the DIS simulation course.

The speed tracking data provided in Figure 5-63 demonstrates how the TORC AutonoNav™ system planned the vehicle stop as well as the driver's associated speed response. At approximately 40ft from the stop point, the motion planner and ANVID began to constantly decrease the reference speed until it reached zero shortly after the stop point. The driver response also shown in Figure 5-63 and the instantaneous error plot in Figure 5-64 indicate that the driver performed a single deceleration as commanded by the SpeedStrip interface and brought the vehicle to a smooth, controlled stop with low instantaneous error. The interaction between the driver and the AutonoNav™ motion planning actually enabled the driver to bring the vehicle to a smooth stop without the need to communicate a full stop stimulus on the SpeedStrip interface. The instantaneous error plot in Figure 5-64 shows that the instantaneous error exhibited by the driver never reached the Full Back vibration zone because he led the reference and stopped the vehicle before it was actually required by the reference. Because the driver stopped with 1.73ft of distance between the front of the vehicle and the stop point, this suggests that the AutonoNav™ system was originally planning a trajectory that would result in the vehicle partially overshooting the stop point.

The lateral deviation plot shown in Figure 5-62 indicates that the driver was able maintain at least 2.57ft of clearance between the lane boundaries and the vehicle tires for the entire duration of the course section. The maximum deviation from the lane
centerline never surpassed ±3.7ft and exhibited a mean absolute lateral deviation of 1.72ft. Closer examination of the lateral deviation and the vehicle trajectory shows that the driver tends to drive inside of the lane centerline over the length of the sharp left-handed turn, leading to an increased MALD. The steering wheel angle tracking and error plots given in Figure 5-66 and Figure 5-67 indicate that the driver was actually not turning the wheel counterclockwise enough during the turn and thus did not induce enough left-handed curvature. Since the lateral deviation was left of the lane centerline and the reference signal commanded a larger left-hand turn than the driver responded with, it can be concluded that the deviation is once again a result of the TORC AutonoNav™ planned trajectories. However, the AutonoNav™ system was able to guide the driver back to the lane centerline after exiting the turn and remain within ±2ft of the centerline until the stop point was reached at the finish line.

Figure 5-66: Reference steering wheel angle generated by the ANVID and the associated driver response at the finish course section.

Figure 5-67: Instantaneous steering wheel angle error exhibited by the driver over the finish section of the DIS simulation course.
Section 6: Conclusions and Future Research

It was stated in the introductory section that the purpose of this paper was to develop robust and intuitive driver assistance technologies that assist blind persons in safely and independently operating an automobile on standard public roads. Such technology would additionally benefit today’s sighted drivers in that it has the ability to augment their vision with suggestive cues during normal and low-visibility conditions. This work presented a non-visual human-computer interface system with two types of controlling software that realizes this type of technology. The research and development of the technologies presented in this work was made possible through the Blind Driver Challenge® effort initiated by the National Federation of the Blind.

Assistance technologies for blind drivers can be implemented in two separate paradigms: instructive and informative. Instructive technologies direct the driver with appropriate actions, while informative technologies provide the driver with critical information. The technologies presented in this work focus on developing instructive technologies that provide steering and speed cues to the driver. The development of these instructive technologies poses four principal problems. The first and most obvious problem is perception. Technology must be developed that can sense the environment for the driver in full, partial, or no-visibility driving scenarios. The second problem requires that motion planning be implemented to determine steering and speed actions that will safely instruct the blind driver through the driving environment. The third problem requires that reference transformations must be emplaced to transform the synthesized motion planning data into non-visual stimuli that can be understood and reliably interacted with by the driver. The fourth and final problem requires a human-computer interface system that is capable of communicating the transformed instructions to the driver in a dependable manner.

Fortunately, technology already exists that satisfies the first two principal issues behind the instructive technology paradigm. The TORC Robotics ByWire XGV™ and AutonoNav™ presented in Section 2 of this work is fully capable of perceiving the driving environment and planning safe trajectories through the environment towards a goal destination. The ByWire XGV™ and AutonoNav™ systems make up the Blind Driver Challenge® research platform and utilize a variety of environmental perception sensors with motion planning software to solve the first two principal issues. The last two principal issues are solved by technologies presented in this work.

In order to satisfy the reference transformation and communication principal issues, this work established three separate contributions. The first contribution proposed a Passive Non-Visual Interface Driver (PNVID) program that transforms TORC AutonoNav™ motion planning signals into a form that can be understood and reliably interacted with by the driver. The PNVID performs these transformations through a
method that is independent of the driver and thus interacts identically amongst all drivers. The second contribution of this work proposed an Adaptive Non-Visual Interface Driver (ANVID) program that also transforms the TORC AutonoNav™ motion planning signals into a form that can be understood and reliably interacted with by the driver. However, the ANVID differs from the PNVID in that the transformations are performed through a method that adapts each and every driver’s particular driving tendencies. The third and final of this work contribution proposed a Non-Visual Interface System (NVIS) comprised of non-visual human-computer interfaces that communicate transformed steering and speed instructions to the driver in a dependable manner. The system contains two separate non-visual interfaces (NVIs): the DriveGrip system for steering cues, and the SpeedStrip system for speed cues. Both the DriveGrip and SpeedStrip NVIs utilize vibro-tactile stimuli to physically communicate necessary instructions to the driver.

A significant amount of research, development, and experimentation was performed to realize the three contributions proposed in this paper. The following subsections provide detailed conclusions focused on each of the three proposed contributions. An additional subsection is subsequently included to discuss areas of future research that may further improve and build upon the contributions and findings of this work.

6.1: The Non-Visual Interface System (NVIS)
Section 3 presented a Non-Visual Interface System (NVIS) designed to realize the third contribution of this work: the communication of steering and speed instructions to the driver in a non-visual and dependable manner. The NVIS was presented first in this work because the PNVID and ANVID software both heavily depend on the functionality of the NVIS. The NVIS is comprised of two non-visual interfaces (NVIs) as well as their associated control and mounting hardware. The first NVI, called DriveGrip, is utilized to communicate steering instructions to the driver through a pair of vibro-tactile gloves worn on the driver’s hands. The second NVI, called SpeedStrip, is utilized to communicate speed instructions through a vibro-tactile cushion overlaid on the driver’s seat.

6.1.1: DriveGrip
The DriveGrip glove interface uses vibro-tactile motors placed on the driver’s left and right-hand fingers to communicate the instantaneous steering wheel angle error to the driver. The instantaneous steering wheel angle error is determined by the difference between the instantaneous actual steering wheel angle and the instantaneous reference steering wheel angle defined by the PNVID or ANVID. Communicating the instantaneous error to the driver enables him or her to perform closed-loop control of the steering wheel angle. The driver essentially attempts to minimize the steering wheel error communicated over the DriveGrip interface.
The interface associates each finger with a range of error magnitude range and each hand with an error direction. In respect to magnitude, vibrations on the little finger represent the largest error range while vibrations on the index finger represent the smallest error range. In respect to direction, the left and right hands represent counterclockwise and clockwise error, respectively. No vibrations indicate that the error has fallen within an acceptable range and no correction is required. Different DriveGrip Operation Configurations (DOCs) were presented that utilize various combinations of vibration and magnitude assignments to communicate the instantaneous error. Significant amounts of testing were performed to analyze the tracking abilities elicited by each DOC and it was found that the Four-Finger-Lenient-Tolerance (FFLT) DOC resulted in the most desirable driver tracking responses.

The DriveGrip gloves themselves were constructed from half-finger gloves with small vibration motors attached on top of the proximal phalanges of the index, middle, ring and little fingers. Special considerations were taken in the design of the interface’s wiring harnesses to minimize fatigue damage due to constant and repetitive use. The gloves are connected to the vehicle through Ethernet cords that enable quick and simple connection/disconnection. An aluminum mount was fabricated to attach to the headrest of the driver’s seat and position the Ethernet wires over the driver’s shoulders. The mount included a DriveGrip Interface Box that converts the cabling from the Non-Visual Interface Controller (NVIC) into the separate Ethernet wires.

An extensive experiment was conducted to measure the steering wheel angle tracking performance exhibited by drivers while using the DriveGrip non-visual interface. Drivers were asked to track step, ramp, and arbitrary type reference steering wheel angle signals using a variety of DriveGrip Operation Configurations (DOCs). It was found that the driver was able to accurately track step signals using all four DOCs, however the TFLT configuration provided the best response characteristics with an 840ms rise time, 0° overshoot, 630ms settling time, and 0.2° steady state error on average. The experiments showed that the driver was able to track ramp signals using all four DOCs; however with decreased accuracy due to the fact that the interface communicates instantaneous error rather than error rate. The TFST configuration induced the best response characteristics with only 1.56° of mean absolute error when compared to the reference ramp signal. It was finally found that the driver could track arbitrary reference signals with the DriveGrip interface also using all four DOCs. The FFST and FFLT configurations produced similar response characteristics of 400-460ms pure time delay and 1.06-1.10° mean absolute error. However, the FFLT was chosen as the nominal configuration because the lenient tolerances required much less controller effort by the driver for similar response results.

One final experiment was conducted to examine if desensitization over extended periods of use would interfere with the driver’s ability to perceive steering instructions.
from the DriveGrip interface. The experiment leveraged 7hr driving periods conducted during practice for a demonstration to determine the effects of desensitization. Tracking responses to the same arbitrary type reference signal were recorded before and after the 8hr constant driving period. Average results from two separate drivers indicated that the drivers were able to track the reference signal with near-identical response characteristics even after 7hrs of continuous interface usage.

6.1.2: SpeedStrip

The SpeedStrip interface uses vibro-tactile motors positioned beneath the thighs and on the back of the driver to communicate the instantaneous speed error to the driver. The instantaneous speed error is determined by the difference between the instantaneous actual vehicle speed and the instantaneous reference speed defined by the PNVID or ANVID. Communicating the instantaneous error to the driver enables him or her to perform closed-loop control of the vehicle’s speed. The driver essentially attempts to minimize the speed error communicated over the SpeedStrip interface.

The interface associates leg and back positions with a range of error magnitude and direction. In respect to magnitude, vibrations closest to the buttocks of the driver indicate the lowest range of error magnitude while vibrations furthest from the buttocks indicate the highest range of error magnitude. In respect to direction, vibrations on the legs and back indicate positive and negative error, respectively. No vibrations indicate that the error has fallen within an acceptable range and no correction is required. Different SpeedStrip Operation Configurations (SOCs) were presented that utilize various combinations of vibration and magnitude assignments to communicate the instantaneous error. Significant amounts of testing were performed to analyze the tracking abilities elicited by each SOC and it was found that the Two-Element (TE) SOC resulted in the most desirable driver tracking responses.

The SpeedStrip interface itself was constructed from a cushion implanted with eight rows of vibrational motors. Four rows of motors were placed down the legs of the driver while the other four rows were placed up the back. The cushion was designed to fit onto the driver’s seat. Special considerations were taken in the design of the interface’s wiring harnesses to minimize fatigue damage due to constant and repetitive use. The interface was connected directly to the Non-Visual Interface Controller (NVIC) through a single nine-conductor cable. The interface attached to the same aluminum headrest mount employed by the DriveGrip interface to ensure that the cushion remained in the correct position on the driver’s seat.

An extensive experiment was conducted to measure the speed tracking performance exhibited by drivers while using the SpeedStrip non-visual interface. Drivers were asked to track step, ramp, and arbitrary type reference speed signals using a variety of SpeedStrip Operation Configurations (SOCs). It was found that the driver was able to
accurately track step signals using both SOCs, however the TE configuration provided marginally better response characteristics with a 0.222m/s overshoot, 8586ms settling time, and 0.28m/s steady state error on average. The experiments showed that the driver was able to track ramp signals using both SOCs; however with severely decreased accuracy due to the fact that the interface communicates instantaneous error rather than error rate. The two configurations produced similar tracking results of 0.61-0.59m/s mean absolute error; however the TE configuration was chosen as the nominal configuration due to the decreased cognitive load imposed on the driver. It was finally found that the driver could track arbitrary reference signals with the SpeedStrip interface also using both SOCs. Once again, both configurations produced similar tracking results with 1740-2000ms of pure time delay and 0.202-0.188m/s of mean absolute error on average. Advantages were found on both configuration responses, however the TE was ultimately chosen as the best configuration again due to the reduced cognitive load imposed on the driver.

An additional experiment was performed to examine the deceleration and braking behavior of the driver while tracking reference speed signals. In this experiment, drivers were subjected to a reference signal composed of successive downward ramps increasing in ramp rate. It was found that drivers respond to smaller error magnitudes with initial decelerations followed by brake tapping if necessary. Larger error magnitudes induced significant braking and often exhibited overshoot of the reference signal. The full-back stimulus was found to be effective in communicating the need for a complete stop and enabled the driver to bring the vehicle to more controlled stops.

One final experiment was conducted to examine if desensitization over extended periods of use would interfere with the driver’s ability to perceive speed instructions from the SpeedStrip interface. The experiment leveraged 3hr driving periods conducted during testing for the ANVID and PNVID programs. Tracking responses to the same arbitrary type reference signal were recorded before and after the 3hr constant driving period. Average results indicated that the driver was able to track the reference signal with near-identical response characteristics even after 3hrs of continuous interface usage.

6.1.3: Simultaneous Interface Usage

Further experiments were conducted to analyze the change in tracking performances associated with simultaneous usage of the DriveGrip and SpeedStrip interfaces. The experiment required a driver to simultaneously track arbitrary steering and speed reference signals with both interfaces using the nominal FFLT and TE operation configurations. Results indicated that the driver was able to track the steering and speed reference signals simultaneously with negligible effects on pure time delay and increased mean absolute errors for both responses. Aside from the increased mean absolute errors, sudden temporary lapses in speed tracking performance occurred
when complex steering actions were required. This proved that positioning the vibro-tactile elements on the primary and non-primary locations of the human body enabled simultaneous, yet coupled, interaction with the DriveGrip and SpeedStrip interfaces.

6.1.4: Associated Control Hardware

Hardware for controlling the vibro-tactile elements of the DriveGrip and SpeedStrip interfaces was also presented in this work. The Non-Visual Interface Controller (NVIC) is essentially a switchbox responsible for physically controlling the power of each vibro-tactile element. The NVIC is comprised of a USB Digital Input/Output (DIO) board and a custom Darlington Array PCB for performing the actual power switching. The NVIC connects to the Non-Visual Interface Computer (NVICPU) through a USB connection. The NVICPU is a laptop computer responsible for hosting the PNVID or ANVID software and controlling the Non-Visual Interface System in coordination with the TORC ByWire XGV™ and AutonoNav™ systems.

6.2: The Passive Non-Visual Interface Driver (PNVID)

Section 4 presented the Passive Non-Visual Interface Driver (PNVID), a software program designed to realize the second contribution of this work: the driver-independent transformation of motion planning data into signals that can be understood and reasonably interacted with by the driver. Three functional requirements were imposed upon the PNVID software to perform these transformations. The first requirement specified a need for smoothing the sawtooth patterns that appear in the reference steering and speed signals due to the rapid regeneration of motion profiles from the TORC AutonoNav™ system. The second requirement indicated that situations where the driver leads a ramping reference signal must be mitigated to avoid significant deviations from the intended vehicle trajectory. The third and final requirement specified a need for offsetting the pure time delay exhibited by drivers when tracking reference steering and speed signals.

Generally speaking, the PNVID calculates and generates the DriveGrip and SpeedStrip non-visual stimuli in parallel. The calculation process accepts motion profiles from the TORC AutonoNav™ system and transforms them into reference steering and speed signals that can be understood and reliably interacted with by the driver. This process uses a series of specialized algorithms to perform the transformations in accordance with the functional requirements previously described. After performing the necessary transformations, the instantaneous steering and speed errors are calculated using measurements of the steering wheel angle and vehicle speed through the TORC ByWire XGV™. The instantaneous steering and speed errors are then converted into non-visual stimuli for the DriveGrip and SpeedStrip interfaces. As the non-visual stimuli are calculated, they are generated in a parallel loop that interfaces directly with the Non-Visual Interface Controller (NVIC). The NVIC subsequently switches the power to the
vibro-tactile elements of the non-visual interfaces as instructed by the non-visual stimuli generation process of the PNVID software.

### 6.2.1: Functional Requirement Satisfaction

The PNVID software utilizes several different algorithms to satisfy the described functional requirements. The reference signal smoothing requirement was solved with the FilterSawtooth() algorithm. This algorithm rids the reference signals of the sawtooth patterns by essentially "riding" the peaks of the sawtooth patterns through a series of specialized case structures. Experiments showed that the reference signals generated through this algorithm were not only smooth, but also improved the accuracy of the reference signals in describing the desired steering and speed actions specified by the motion profiles. The ramp lead mitigation requirement was solved with the CheckRampLead() algorithm. This algorithm detects when the driver is leading a ramp in one of the reference signals and reports zero error to avoid corrections in the wrong direction and to allow the reference signal to essentially "catch up" to the response.

The third and final requirement associated with the driver pure time delay was mitigated through the TORC AutonoNav™ system rather than the PNVID software. The AutonoNav™ system includes a "planning horizon" parameter that may be adjusted to offset the pure time delay exhibited by the driver. This parameter effectively reported future motion profiles as current motion profiles and provided the PNVID with futuristic trajectory data to offset the driver's pure time delay.

### 6.2.2: Software Implementation

The algorithms of the PNVID program are implemented in software designed with the LabVIEW Development System and TORC JTK™ add-on module. The software was written using LabVIEW code due to the numerous advantages LabVIEW provides in both parallel programming and hardware integration. The software is additionally implemented as a JAUS interoperable component so that it may interact with the JAUS-based Blind Driver Challenge® research platform. JAUS, or the Joint Architecture for Unmanned Systems, specifies the hierarchy and communication protocol of software components within an unmanned system. The Blind Driver Challenge® research platform and the PNVID software utilize JAUS to maintain an organized software design and standardize the method of communication between TORC and Virginia Tech designed software components.

### 6.2.3: Comprehensive System Performance

An extensive experiment was conducted to analyze the overall performance of the Blind Driver Challenge® system and PNVID software as blind driver assistance technologies. The experiment required several drivers to navigate a complex road course utilizing the technologies presented in this paper and investigated the resulting trajectories. The
particular road course for this experiment was the Patriot course of the Virginia International Raceway (VIR). This course offered a diverse composition of roadways and enabled a wide array of driving scenarios to be tested. The course included snaking curves with various curvatures, straightaways, and locations for static and dynamic obstacle testing.

Overall, the driver remained within 1.76ft from the lane centerline and within 1.79mph of the specified speed limits on average over the entire length of the 1.1mi long VIR patriot course. The driver never exited the driving lane and maintained a clearance of at least 1ft between the lane boundaries and the vehicle tires. The driver reached speeds of up to 25mph and was able to decelerate to lower speeds within 4s. Within the obstacle fields, the driver was able to avoid statically and dynamically placed objects with at least 5ft of clearance while traveling at 15mph. Finally, the driver was able to come to a complete stop from 15mph at the finish line with 1.21ft of distance between the front bumper and the finish line stop point. These results proved that the driver was able to use the Non-Visual Interface System and PNVID software to successfully navigate a vehicle on a complex road course in a safe and reliable manner.

6.2.4: Rolex 24 Demonstration

On January 29th, 2011, the first public demonstration of the Blind Driver Challenge® technologies was given at the Rolex 24 GRAND-AM race at the Daytona International Speedway (DIS). The demonstration included a completely blind driver independently navigating the Infield course of the DIS and performing several complex navigation tasks to demonstrate the capabilities of the Blind Driver Challenge® system and the capacity of the blind. A significant amount of preparation was required for this high scale demonstration. The actual development process of a significant amount of technologies used in this demonstration did not start until seven months ahead of time. Within this seven-month timeframe, the Non-Visual Interface System and PNVID software was developed and tested to ensure proper functionality for the demonstration. The comprehensive Blind Driver Challenge® system was finalized two weeks before the demonstration date; however blind driver training and practice had already been occurring for several months beforehand.

The significant amount of work performed in preparation for the Rolex 24 demonstration enabled Mark Riccobono to navigate around the DIS demonstration course under complete blindness with perfect execution. Mark stayed well within the lane boundaries and reached speeds of up to 27mph on the high-speed straightaway. He was also able to avoid static obstacles and obstacles thrown from the rear of a lead vehicle and even passed the lead vehicle on the left before the finish line. The successful demonstration was publicized internationally and proved to people everywhere that a blind person could successfully drive a vehicle using the technologies presented in this work.
6.3: The Adaptive Non-Visual Interface Driver (ANVID)

Section 5 presented the Adaptive Non-Visual Interface Driver (ANVID), a software program designed to realize the third and final contribution of this work: the driver-adaptive transformation of motion planning data into signals that can be understood and reasonably interacted with by the driver. Three functional requirements were imposed upon the ANVID software to perform these transformations. The first requirement specified a need for smoothing the sawtooth patterns that appear in the reference steering and speed signals due to the rapid regeneration of motion profiles from the TORC AutonoNav™ system. The second requirement indicated a need for offsetting the varying pure time delays exhibited by different drivers when tracking reference steering and speed signals. The third and final requirement specified a need for adapting to each driver’s particular tendencies in responding to non-visual stimuli from the DriveGrip and SpeedStrip interfaces.

The ANVID is able to satisfy these functional requirements with the use of Model Predictive Control (MPC). MPC is a method that seeks suboptimal control inputs to a dynamical system that are determined by predicting the systems output. Essentially, MPC is an optimization process that incorporates a model of the system within the objective function. The ANVID software utilizes MPC to determine a finite horizon of suboptimal future reference steering and speed signals based on the commanded steering and speed actions specified by the TORC AutonoNav™ motion profiles. The suboptimal reference signals adapt to each driver through model of that particular driver. The ANVID software supplements the MPC process with a driver model that is continuously trained online in real-time and in parallel with the operation of the vehicle.

6.3.1: Driver Modeling

The ANVID software continuously models the driver using a set of Time Series Prediction Neural Networks (TSPNNs). A separate model is maintained to predict future steering and speed actions taken by the driver in response to a particular reference signal. The models input 0.75s of past actions and 0.75s of future reference signals to predict 0.75s of future actions. Each TSPNN model contains 102 neurons with two hidden layers and an output layer. Predictions can be made with the models in real-time using FIFO buffers that store the past 0.75s of driver actions. The MPC process uses these predictions to determine a set of reference signals that will force the driver to recreate the commanded actions of the motion profile.

The models are trained online and in real-time with supervised learning to adapt to the driver’s dynamics as they vary with time. A specialized training process, called the Multiple Extended Kalman Algorithm (MEKA), is utilized to perform rapid training of the neural network model weights. The MEKA process implements a local Extended Kalman Filter (EKF) at each neuron to perform the weight updates in a recursive
manner. The EKF enables the weights to be trained rapidly but restricts the size of the neural network in terms of processing time as each neuron must maintain a separate inverse covariance matrix. The recursive nature of the EKF allows the training of the TSPNN driver models to be performed online using supervisory data measured from driver operating in parallel. A time-shifted scheme is used to perform the training with supervised learning in real-time even though the output of the models is the prediction of future actions that have not yet occurred.

### 6.3.2: Reference Optimization

The main functionality of Model Predictive Control is provided through an optimization process. It was described that the ANVID uses MPC to optimize the reference steering and speed signals based on predicted actions from the steering and speed driver models. A Quasi-Newton Optimization (QNO) process is implemented to determine these suboptimal reference signals. The QNO process is a gradient descent-based objective function minimizer that iteratively finds stationary points using first and second derivatives of the objective function surface. The use of second derivatives allows the descent to take a more direct path down the objective function surface and quickly converge. The QNO also allows the inverse of the Hessian matrix containing the second derivatives of the objective function to be estimated rather than explicitly calculated.

A separate QNO process is conducted to calculate the steering and speed reference signals each time a new motion profile is received from the TORC AutonoNav™ system. Each process utilizes an objective function that describes the mean-squared-error (MSE) between the predicted driver response from the TSPNN driver model and the commanded action specified by the current motion profile. Thus, the QNO iteratively determines new reference signals until the MSE between the predicted and commanded actions is minimized. A back-tracking line search using the Armijo Rule and Wolfe/Strong Wolfe conditions is used to determine appropriate optimizer step sizes. In order to increase the efficiency of the QNO process, the gradient of the objective function is algorithmically calculated. This algorithm determines the gradient of the objective function with respect to the reference signal even though the objective function contains the TSPNN driver model in which the gradient calculation is not straightforward.

### 6.3.3: Software Implementation

The algorithms of the ANVID program are implemented in software designed with the LabVIEW Development System and TORC JTK™ add-on module. The software was written using LabVIEW code due to the numerous advantages LabVIEW provides in both parallel programming and hardware integration. The software is additionally implemented as a JAUS interoperable component so that it may interact with the JAUS-
based Blind Driver Challenge® research platform. JAUS, or the Joint Architecture for Unmanned Systems, specifies the hierarchy and communication protocol of software components within an unmanned system. The Blind Driver Challenge® research platform and the ANVID software utilize JAUS to maintain an organized software design and standardize the method of communication between TORC and Virginia Tech designed software components.

6.3.4: MPC and Driver Modeling Performance
Experiments were conducted to analyze the performance and timing statistics of the Model Predictive Controller (MPC) and Driver Modeling (DM) processes. The DM process was able to predict driver steering and speed actions with mean absolute errors of 2.06° and 0.051m/s, respectively. The predictions could be calculated in 237μs of processing time on the Non-Visual Interface Computer and thus could be rapidly evaluated within the MPC process without injecting substantial delay into the Blind Driver Challenge® system. The DM training process executes in parallel with the operation of the vehicle at an average rate of 62.7Hz for each driver model. These results prove that the DM process is capable of accurately predicting the driver's response in real-time.

The MPC process is able to determine suboptimal steering and speed reference signals only 38.38ms after receiving the most recent motion profile from the AutonoNav™ system. Each QNO process requires only 14.343ms of processing time and is designed to run in parallel to decrease the pure time delay of the system. However, overhead and a lack of enough processor cores created a tendency of the parallel 14.343ms processes to take 38.38ms overall. The evaluation of the objective function and its gradient with respect to a reference signal required only 854μs of processing time. Since these evaluations can occur up to 50 times per optimization iteration, the low processing time becomes extremely important.

6.3.5: Comprehensive System Performance
An extensive experiment was conducted to analyze the overall performance of the Blind Driver Challenge® system and ANVID software as blind driver assistance technologies. The experiment required a driver to navigate a complex simulated road course utilizing the technologies presented in this paper and investigated the resulting trajectories. The particular simulated road course for this experiment was taken from the Infield course of the Daytona International Speedway (DIS). This course offered a diverse composition of roadways and enabled a wide array of driving scenarios to be tested. The course included medium and high-curvature turns, hairpin turns, and high-speed straightaways.

Overall, the driver remained within 1.33ft from the lane centerline and within 1.51mph of the specified speed limits on average over the entire length of the DIS simulation course. The driver never exited the driving lane and maintained a clearance of at least
2.6ft between the lane boundaries and the vehicle tires. The driver reached speeds of up to 20mph and was able to decelerate to lower speeds within 4s. At the end of the simulation course, the driver was able to come to a complete stop from 5mph with 1.73ft of distance between the front bumper and the finish line stop point. These results proved that the driver was able to use the Non-Visual Interface System and ANVID software to successfully navigate a vehicle on a complex road course in a safe and reliable manner.

In comparison with the PNVID comprehensive performance analysis, the ANVID induced lower average lateral deviations and speed deviations over the length of the courses experimented upon. However, the decreased deviations while using the ANVID software came with a price of increased controller effort. The ANVID was able to determine more complex reference signals that forced the driver to more accurately recreate the desired trajectories planned by the TORC AutonoNav™ system. A side-effect of this process is that the controller effort required by the driver is increased. The mean absolute error between the driver’s steering and speed responses and the reference signals is also increased in the ANVID implementation. However, the MPC process leverages the response-reference error to force the driver into recreating the trajectory planned by the AutonoNav™ system. The larger response-reference error effectively causes the improved trajectory tracking performance through the MPC process.

6.4: Future Research

This subsection defines avenues of research that may be conducted in the future to improve or extend the driver assistance technologies presented in this work. Considerations for the improvement of the Non-Visual Interface System and the ANVID software will first be discussed. Next, a shift to an informational paradigm of driver assistance technologies will be outlined. Finally, the discussion of future research will conclude with the extension of the Non-Visual Interface System to other research areas.

6.4.1: Improvement of the Non-Visual Interface System

Several improvements can be made to refine the hardware and performance capabilities of the Non-Visual Interface System presented in Section 3. The first improvement suggests the construction of DriveGrip interface gloves that are wireless and do not tether the driver’s hands to the vehicle. Currently, the Ethernet wires connecting the gloves to the vehicle are positioned to minimize interference during complex steering maneuvers; however the wiring can still be cumbersome and complicates entering and exiting the vehicle. Wireless gloves would enable true freedom of the driver’s hands and will simplify complex maneuvers such as hand-over-hand turning. Furthermore, the driver will be able to perform other tasks in the car, such
as fastening a seatbelt and adjust the air conditioning, without interference from the wires.

There are several considerations that must be taken with respect to the implementation of wireless gloves, however. Firstly, the gloves must be powered with an attached battery. The battery will increase the weight of the gloves, which may or may not impose another form of interference on the driver's hands. The battery must be kept charged at all times, which unfortunately causes the possibility of reliability issues. The batteries may be charged constantly and wirelessly through electromagnetic induction between conductor coils in the palm of the glove and on the steering wheel. The second issue to consider is that the non-visual stimuli must be communicated to the gloves in a wireless fashion. This method would further increase the possibility of reliability issues by increasing the complexity of the gloves. The gloves would have to contain a wireless radio and microcontroller (for example a XBee) that would also increase the weight of the gloves and possibly interfere with the driver's motions.

The second improvement to the Non-Visual Interface System suggests that the SpeedStrip interface be integrated into the actual driver's seat of the vehicle. Currently, the interface is composed of a cushion that fits over the driver's seat and poses several menial issues that can be resolved by integrating the vibro-tactile elements into the seat. The first issue is that, despite the significant amount of padding, the cushion becomes uncomfortable to sit on for prolonged periods of time. The second issue is that the position of the cushion shifts each time drivers enter and exit the driver's seat. By integrating the vibro-tactile elements into the driver's seat, the comfort of the seat may be increased and the position of the elements on the seat will remain constant. This improvement would additionally increase the reliability of the system as the interface will be contained and protected within the driver's seat.

The third improvement aims to elicit better reference signal tracking performance from the driver using the DriveGrip and SpeedStrip interfaces. The operation configurations defined for the two non-visual interfaces utilize error tolerances that are independent from speed. However, tolerances that adjust with speed will provide more accurate control when required and can decrease the controller effort when accurate control is not needed. Therefore this improvement suggests that both the DriveGrip and SpeedStrip operation configurations be defined as functions of speed. The steering error tolerances should tighten as speed increases since relatively small deviations in the steering wheel angle lead to large deviations in trajectory at higher speeds. The speed error tolerances should loosen as speed increases as fine speed control is typically only required when traveling at slow speeds for safety or bringing the vehicle to a controlled stop. An example of speed-dependent operation configurations has been provided with more detail at the end of Section 3.1.4.3.
The fourth and final improvement to the Non-Visual Interface System aims to better quantify the results of interface desensitization over periods of prolonged usage. This work analyzed the effects of desensitization by examining the difference in tracking results after prolonged periods of driving. However, vibro-tactile desensitization may be better analyzed by conducting experiments at the interface level rather than the reference tracking level. It is therefore suggested that an additional experiment should be conducted that measures interface desensitivity through singular stimulations and reactions. The experiment can activate a single vibro-tactile element and have the driver respond by pressing a button or key that is associated with that particular element. Software can generate these stimuli randomly and record the driver's response time and if the correct response was made by the driver. Such an experiment would be able to statistically determine the driver's sensitivity to the vibro-tactile elements and prove if desensitization occurs over prolonged periods of interface usage.

6.4.2: Improvement of the ANVID Software

Two improvements can be made to refine the software and performance capabilities of the Adaptive Non-Visual Interface Driver (ANVID) presented in Section 5. The first improvement suggests that the separate steering and speed TSPNN driver prediction models discussed in this work be combined into a single TSPNN prediction model. The ANVID decoupled the steering and speed responses of the driver to decrease the computational time required for evaluating and training the model and improve the execution performance of the software. While the satisfactory modeling accuracy and comprehensive system results proved that decoupling the models was valid, the accuracies of these results could be improved by implementing a single coupled model. The interaction between the driver's steering and speed responses may be relatively small but is still present and can be accounted for to increase the overall system performance.

The second improvement aims to reduce the processing times required for training and evaluating the TSPNN driver models as well as for the optimization of the steering and speed reference signals. Reducing the processing time required for a single driver model training iteration increases the rate at which online, real-time training may occur and enables faster convergence of the model to the driver's dynamics. Reducing the processing time of the model evaluation and reference optimization processes will equally reduce the amount of pure time delay injected into the Blind Driver Challenge® system and improve navigational performance. Therefore, it is suggested that the model training and evaluation as well as the optimization processes be implemented on an FPGA dedicated to these calculations. An FPGA can truly harness the advantages of parallel computations and would dramatically reduce the processing time required for processes that can be decoupled into parallel subsections. The evaluation and training of TSPNNs can benefit from this as both processes are performed locally at each
neuron within the layers of the neural network. The optimization process may also benefit from an FPGA implementation because it must conduct many iterations between a single input and output of data. The MPC could provide the FPGA optimization implementation with an initial guess and let the FPGA dedicatedly perform the required optimization iterations. Once the iterations are complete, the FPGA would provide the final suboptimal solution back to the MPC process.

6.4.3: Development of Informational Interfaces

It was described in the Problem Statement of this paper that driver assistance technologies may be implemented in either instructive or informational paradigms. While this work focused on the development of instructive technologies, it is still suggested that further research be conducted to develop informational technologies to provide perception data directly to the driver. This paradigm shift would enable the driver to make conscious driving decisions based on the environmental information that is provided. Essentially, the motion planning of the vehicle will be offloaded from the Blind Driver Challenge® system onto a more reliable human operator. Work has already begun that outlines an informational interface called the Kinesthetic Tactile Display in [141]. This referenced work also provides a guide and considerations that should be taken in the development of non-visual interfaces. This helpful paper can be utilized to streamline the creation of additional informative interfaces in the future.

6.4.4: Extensions of the Non-Visual Interface System

The final avenue of suggested future research is the extension and application of the Non-Visual Interface System developed in this work to other real-world problems. The DriveGrip and SpeedStrip interfaces can easily be extended to serve as assistance technologies in any mobility situation. For example, similar technology may be used to help guide a blind person on foot through heading and speed adjustments without the requirement for a cane or guide dogs. The same technology may be applied to blind persons who are constrained to a wheelchair. Such applications could greatly improve the safety and quality of life of the blind population.

The non-visual technologies can also be applied to applications completely unrelated to mobility. The DriveGrip and SpeedStrip interfaces are designed to communicate numbers through descriptions of magnitude and direction. Extensions of this technology could use vibro-tactile element arrays to communicate mathematical signal or function graphs to blind students, engineers, scientists, etc. It is therefore suggested that future research be conducted to examine and design extended applications for the many different uses that the Non-Visual Interface System may provide.
Section 7: References


Appendix A: Non-Visual Interface Performance Analysis Data Sets

A.1: DriveGrip Performance Analysis Full Data Set

A.1.1: Step Reference Steering Wheel Angle Signals

Figure A-1: Average driver response to a 30° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.

Figure A-2: Average driver response to a 30° step reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-3: Average driver response to a 30° step reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2.

Figure A-4: Average driver response to a 30° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.
Figure A-5: Average driver response to a 60° step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.

Figure A-6: Average driver response to a 60° step reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-7: Average driver response to a 60° step reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2.

Figure A-8: Average driver response to a 60° step reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-9: Average driver response to a $180^\circ$ step reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.

Figure A-10: Average driver response to a $180^\circ$ step reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-11: Average driver response to a 180° step reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2.

Figure A-12: Average driver response to a 180° step reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2.
A.1.2: Ramp Reference Steering Wheel Angle Signals

Figure A-13: Average driver response to a $5^\circ$/s ramp reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.

Figure A-14: Average driver response to a $5^\circ$/s ramp reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-15: Average driver response to a 5°/s ramp reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2.

Figure A-16: Average driver response to a 5°/s ramp reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-17: Average driver response to a $10^\circ$/s ramp reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.

Figure A-18: Average driver response to a $10^\circ$/s ramp reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-19: Average driver response to a 10°/s ramp reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2.

Figure A-20: Average driver response to a 10°/s ramp reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-21: Average driver response to a 30°/s ramp reference steering signal utilizing the FFST DriveGrip operation configuration defined in Table 3-2.

Figure A-22: Average driver response to a 30°/s ramp reference steering signal utilizing the FFLT DriveGrip operation configuration defined in Table 3-2.
Figure A-23: Average driver response to a 30°/s ramp reference steering signal utilizing the TFST DriveGrip operation configuration defined in Table 3-2.

Figure A-24: Average driver response to a 30°/s ramp reference steering signal utilizing the TFLT DriveGrip operation configuration defined in Table 3-2.
A.2: SpeedStrip Performance Analysis Data Set

A.2.1: Step Reference Speed Signals

Figure A-25: Average driver response to a 5m/s step reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7.

Figure A-26: Average driver response to a 5m/s step reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7.
Figure A-27: Average driver response to a 10m/s step reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7.

Figure A-28: Average driver response to a 10m/s step reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7.
A.2.2: Ramp Reference Speed Signals

Figure A-29: Average driver response to a \(0.3\text{m/s}^2\) ramp reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7.

Figure A-30: Average driver response to a \(0.3\text{m/s}^2\) ramp reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7.
Figure A-31: Average driver response to a 0.5m/s² ramp reference speed signal utilizing the FE SpeedStrip operation configuration defined in Table 3-7.

Figure A-32: Average driver response to a 0.5m/s² ramp reference speed signal utilizing the TE SpeedStrip operation configuration defined in Table 3-7.