3-D Point Cloud Generation from Rigid and Flexible Stereo Vision Systems

Nathaniel J. Short

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Engineering

A. Lynn Abbott

Kevin Kochersberger

Robert Broadwater

December 4th, 2009

Blacksburg, VA

Keywords: Stereo Vision, UAV, VTOL, Camera Calibration, Terrain Mapping

Copyright 2009
3-D Point Cloud Generation from Rigid and Flexible Stereo Vision Systems

Nathaniel Short

ABSTRACT

When considering the operation of an Unmanned Aerial Vehicle (UAV) or an Unmanned Ground Vehicle (UGV), such problems as landing site estimation or robot path planning become a concern. Deciding if an area of terrain has a level enough slope and a wide enough area to land a Vertical Take Off and Landing (VTOL) UAV or if an area of terrain is traversable by a ground robot is reliant on data gathered from sensors, such as cameras. 3-D models, which can be built from data extracted from digital cameras, can help facilitate decision making for such tasks by providing a virtual model of the surrounding environment the system is in. A stereo vision system utilizes two or more cameras, which capture images of a scene from two or more viewpoints, to create 3-D point clouds. A point cloud is a set of un-gridded 3-D points corresponding to a 2-D image, and is used to build gridded surface models.

Designing a stereo system for distant terrain modeling requires an extended baseline, or distance between the two cameras, in order to obtain a reasonable depth resolution. As the width of the baseline increases, so does the flexibility of the system, causing the orientation of the cameras to deviate from their original state. A set of tools have been developed to generate 3-D point clouds from rigid and flexible stereo systems, along with a method for applying corrections to a flexible system to regain distance accuracy in a flexible system.
ACKNOWLEDGEMENTS

I would like to thank my committee members for their help in writing this thesis; Dr. Abbott, Dr. Broadwater, and Dr. Kochersberger. Dr. Abbott first pointed me towards the Unmanned Systems Lab and his help with the stereo vision has been extremely helpful. I would especially like to thank Dr. K for providing me the opportunity to work at the Unmanned Systems Lab, where I was able to do research on some exciting projects in computer vision and without which I would not have been able to produce this thesis. In addition, this research would not have been possible without the support from Pacific Northwest National Laboratory.

I would also like to thank all the fellow research assistants at the lab who have been a pleasure to work with over the past year. I would especially like to thank Prather Lanier, Brian McCabe, and Jason Gassaway for their direct help on various aspects of research conducted in this thesis.

My close friends and family have always been there for every achievement in my life, and for that I am thankful. Their love, support, and direction have gotten me to and past many milestones in life. I’d especially like to thank my dad for his help and willingness to explain things to me that have helped with my coursework and particularly with some of the research for this thesis.
CONTENTS

ABSTRACT ........................................................................................................ ii

ACKNOWLEDGEMENTS ............................................................................... iii

CONTENTS ........................................................................................................ iv

LIST OF FIGURES .............................................................................................. vi

LIST OF TABLES ................................................................................................ x

Chapter One: Introduction ................................................................................. 1

1.1 Background .................................................................................................. 1

1.2 Problem Statement ....................................................................................... 3

1.3 Contribution ................................................................................................ 5

1.4 Organization of Thesis ................................................................................ 5

Chapter Two: Binocular Vision .......................................................................... 6

2.1 Stereo Vision .............................................................................................. 6

2.2 Camera Calibration ..................................................................................... 8

2.2.1 Overview ................................................................................................ 8

2.2.2 Camera Model ......................................................................................... 10

2.2.3 Rectification .......................................................................................... 12

2.3 Correlation .................................................................................................. 16

2.4 3-D Re-projection ...................................................................................... 19

2.5 Stereo vision considerations ...................................................................... 20

2.5.1 Camera Selection and System Design .................................................... 20

2.5.2 Stereo Procesre 14sing Latency .............................................................. 24

Chapter Three: Stereo Vision System Tools .................................................... 27

3.1 Stereo Vision System Tools Overview ....................................................... 27
3.2 Stereo Vision System Tools Process................................................................................. 29

Chapter Four: Custom Stereo Vision Systems........................................................................ 34

4.1 Stereo Systems .................................................................................................................. 34

4.2 Stereo Vision Intermediate Results.................................................................................. 38

4.3 Terrain Mapping Results .................................................................................................. 40

4.4 Object Detection and Distance Measurement Performance.............................................. 45

Chapter Five: Camera Pose Correction for Large Baseline Stereo System.......................... 50

5.1 Overview .......................................................................................................................... 50

5.2 Instrumentation ................................................................................................................ 51

5.3 Experimental Setup ........................................................................................................ 53

5.4 Test Procedures ............................................................................................................... 55

5.5 Results of Camera Motion Testing.................................................................................... 59

5.6 Real-time Camera Trigger ............................................................................................... 61

Chapter Six: Conclusions and Future Work ........................................................................... 65

WORKS CITED ...................................................................................................................... 67

Appendix A: Stereo Vision System Tools C++ source code.................................................... 69

Appendix B: Object extraction source code using Matlab...................................................... 109

Appendix C: Camera Calibration Guide using Matlab............................................................. 114
LIST OF FIGURES

FIGURE 1: RESULTS OF BEST-FIT PLANES FOR LANDING SITE ESTIMATION. (KLOMPARENS, 2008) ... 2
FIGURE 2: RESULTS OF HOUGH LINE DETECTION FOR DETECTING POWER LINES. (HAGER, 2009) ... 2
FIGURE 3: BUMBLEBEE STEREO SYSTEM FROM POINT GREY RESEARCH ........................................ 2
FIGURE 4: RMAX HELICOPTER USED FOR IMAGING FLIGHT, WITH STEREO VISION SYSTEM ATTACHED ...................................................................................................................... 3
FIGURE 5: EXAMPLE OF BINOCULAR VISION IN HUMANS (COOPER, 1995) ........................................... 6
FIGURE 6: STEREO VISION SYSTEM WITH 4 CAMERAS. THE USER HAS THE CHOICE OF 2 CAMERAS ON THE LEFT AND 2 CAMERAS ON THE RIGHT. THIS IMAGE WAS TAKEN OF A STEREO VISION SYSTEM BUILT AT THE UNMANNED SYSTEMS LAB AT VIRGINIA TECH ......................................................... 7
FIGURE 7: SIMPLE GEOMETRY FOR STEREO RANGING. THE USUAL GOAL IS TO FIND THE RANGE Z FROM THE CAMERAS TO A POINT P IN THE SCENE ................................................................. 8
FIGURE 8: CHECKERBOARD PATTERN USED AS CALIBRATION TARGET. THIS TARGET HAS A HEIGHT OF APPROXIMATELY 18 FEET .................................................................................................. 9
FIGURE 9: EXAMPLE OF BARREL DISTORTION EFFECT ......................................................................... 10
FIGURE 10: CALIBRATION TARGET WITH BARREL DISTORTION (LEFT) AND AFTER CORRECTION (RIGHT) ............................................................................................................................................. 11
FIGURE 11: RADIAL DISTORTION PLOT FOR A PARTICULAR LENS (BRADSKI & ADRIAN, 2008) .... 12
FIGURE 12: STEREO IMAGING GEOMETRY. (A) EPIPOLAR PLANE, WHICH IS THE PLANE FORMED BY THE 3D POINT P AND THE CENTER OF PROJECTIONS. (B) EPIPOLAR PLANE WITH MULTIPLE 3D POINTS ALONG RIGHT EPIPOLAR LINE CORRESPONDING TO ONE POINT ON THE LEFT EPIPOLAR LINE ........................................................................................................................................ 13
FIGURE 13: COLOR IMAGE OF SCENE (LEFT) AND DISPARITY MAP (RIGHT), INCREASING WINDOW SIZE WHEN COMPUTING SAD WILL PRODUCE SMOOTHER OBJECTS AND LESS DATA WHERE DECREASING THE WINDOW WILL HAVE MORE DATA BUT LESS DEFINED OBJECTS ................ 18
FIGURE 14: 3-D RE-PROJECTION (RIGHT) OF A 2-D IMAGE (LEFT) FROM STEREO PAIR ...................... 20
FIGURE 15: HORIZONTAL DISTANCE COVERED PER PIXEL AT VARIOUS DISTANCES AND VARIOUS CAMERA RESOLUTIONS USING A FOCAL LENGTH OF 3.8 MM .............................................................................. 21
FIGURE 16: DEPTH RESOLUTION VS. DISTANCE TO OBJECT OF TWO STEREO SYSTEMS WITH BASELINES OF 12” AND 18” ........................................................................................................... 23
FIGURE 17: DEPTH RESOLUTION (VERTICAL AXIS) VS. BASELINE DISTANCE (HORIZONTAL AXIS). . 24
FIGURE 18: GRAPH SHOWING CORRELATION ROUTINE EXECUTION TIME OF TWO IMAGES. ........... 25
FIGURE 19: STEREO VISION SYSTEM FLOW DIAGRAM. ............................................................ 28
FIGURE 20: STEREO CONTROLS USED FOR ADJUSTING CORRELATION VARIABLES. ............ 33
FIGURE 21: STEREO SYSTEM WITH 18’ PITCH USING TWO COLOR CAMERAS FROM WATEC. ....... 34
FIGURE 22: 18” BASELINE STEREO SYSTEM USING 2 B&W AND 1 COLOR SONY CAMERAS. ....... 35
FIGURE 23: 14” BASELINE STEREO VISION SYSTEM WITH LENSES VISIBLE THROUGH HOLE IN WHITE CRADLES AT THE ENDS .............................................................................................................. 36
FIGURE 24: 5’ STEREO VISION SYSTEM WITH SONY XCDU100 AND XCDU100CR CAMERAS .... 37
FIGURE 25: VIEW OF LEFT AND COLOR CAMERAS FROM 5’ STEREO SYSTEM. ...................... 37
FIGURE 26: VIEW OF RIGHT CAMERA FOR 5’ STEREO SYSTEM. ............................................. 37
FIGURE 27: LEFT AND RIGHT IMAGES TAKEN FROM 14” STEREO VISION SYSTEM BEFORE RECTIFICATION AND UN-DISTORTION .................................................................................................................... 38
FIGURE 28: LEFT AND RIGHT IMAGES TAKEN FROM 14” STEREO VISION SYSTEM AFTER RECTIFICATION AND UN-DISTORTION .................................................................................................................... 39
FIGURE 29: DISPARITY MAP OF THE SCENE FROM FIGURE 28 TAKEN BY 14” PITCH STEREO SYSTEM. ........................................................................................................................................ 40
FIGURE 30: TEST SITE AT UNMANNED SYSTEM LAB USED TO BUILD 3-D MODEL. .................... 41
FIGURE 31: UN-GRIDDED POINT CLOUD GENERATED FROM 18” STEREO VISION SYSTEM WITH HEIGHT COLORATION OVERLAY .......................................................................................... 41
FIGURE 32: WIRE MESH GRIDDED SURFACE MODEL GENERATED FROM 18” STEREO VISION SYSTEM WITH HEIGHT COLORATION OVERLAY .............................................................................. 41
FIGURE 33: MODIFIED TEST SITE AT UNMANNED SYSTEMS LAB. ............................................. 42
FIGURE 34: UN-GRIDDED POINT CLOUD GENERATED USING 18” STEREO SYSTEM USING HIGH RESOLUTION CAMERAS. HEIGHT OF TERRAIN IS MAPPED USING A COLOR MAP ................. 43
FIGURE 35: WIRE MESH GRIDDED SURFACE MODEL OF SAMPLE TERRAIN SITE WITH COLOR MAP CORRESPONDING TO HEIGHT OVERLAI D. ................................................................................. 43
FIGURE 36: 5’ STEREO VISION SYSTEM MOUNTED ONTO A YAMAHA RMAX VTOL UAV. ....... 44
FIGURE 37: LEFT, RIGHT, AND COLOR IMAGES TAKEN FROM 5’ STEREO VISION SYSTEM AT 60 METERS ............................................................................................................................................. 44
FIGURE 38: Gridded surface of model generated from left and right images taken with 5’ stereo system at 60 meters, shown with a color image texture overlay ........ 44
FIGURE 39: Left and right view of objects in a scene (further view). Storage tub set on a stool (green), newspaper lying on the floor (orange), end of a wooden shelf (red), and an orange cone (blue) ........................................................................ 45
FIGURE 40: Left and right view of objects in a scene (closer view) ...................... 46
FIGURE 41: 4 connected and 8 connected components .......................................... 47
FIGURE 42: Components extracted from point cloud, far view (left). Corresponding components extracted in rectified image (right) ................................................ 48
FIGURE 43: Components extracted from point cloud, close view (left). Corresponding Components extracted in rectified image (right) ................................................ 48
FIGURE 44: Stereo vision system with 10 inch pitch. The two lenses are visible near the ends of the metal bar. ...................................................................................... 51
FIGURE 45: Camera-centered coordinate system. This shows a Watec 660D G3.8 mono board camera, which is the model used for this project ....................................... 52
FIGURE 46: Stereo System with two Watec 660D cameras and Axis 241Q Video Server. ......................................................................................................................... 53
FIGURE 47: Shaker (left) is a Vibration System Model VG10054. Accelerometer (middle) is a PCB 35C65 and the analyzer is a Spectral Dynamics 20-24 (right) .... 54
FIGURE 48: View of scene used for measuring distances with stereo system .......... 55
FIGURE 49: Graph of acceleration data in m/s (top) and graph of accelerometer displacement (bottom) ............................................................................................. 56
FIGURE 50: Scaled mode shape plot based on measured acceleration at 87 mm .... 57
FIGURE 51: Scaled beam slope plot based on distance to center of camera lens at 97 mm .......................................................................................................................... 57
FIGURE 52: Disparity map from correlation without vibration (left) and with vibration (right) ............................................................................................................. 58
FIGURE 53: Disparity map resulting from applying angle calculated for camera corrections ............................................................................................................... 59
FIGURE 54: 3-D model of scene (left) with height profile (right) before correct angle applied .................................................................................................................. 60
Figure 55: 3-D model of scene (left) with height profile (right) after correct angle applied, the height profile is taken over the tall object in the scene, as indicated by the red line in the 3-D model.

Figure 56: Plot of acceleration measured from accelerometers (top) and least squares fit cosine wave (bottom).

Figure 57: Pseudo code for real-time camera correction using least square fit approximation.
LIST OF TABLES

TABLE 1: EXECUTION TIMES FOR CORRELATION ROUTINE ON A 704x480 AND 1600x1200 IMAGE.  ........................................................... 25

TABLE 2: SPECIFICATIONS OF WATEC CAMERAS USED IN 18” STEREO SYSTEM............................ 35

TABLE 3: CAMERA SPECIFICATIONS FOR SONY XCDU100 WITH PENTAX C60607KP - H612A(KP)
  LENS. (SONY, 2009) (PENTAX, 2009) .................................................................................................................. 36

TABLE 4: CAMERA SPECIFICATIONS FOR SONY XCDU100CR WITH PENTAX C60607KP -
  H612A(KP) LENS. (SONY, 2009) (PENTAX, 2009) .................................................................................................................. 36

TABLE 5: MEASURED DISTANCES FROM 14” STEREO SYSTEM TO OBJECTS IN FAR AND CLOSE TO
  SCENE. ......................................................................................................................................................... 46

TABLE 6: MEASURED AND CALCULATED DISTANCES FROM 14” STEREO SYSTEM TO OBJECTS IN FAR
  AND CLOSE TO SCENE. ......................................................................................................................................... 46

TABLE 7: MEASURED AND CALCULATED SIZE OF SHELF WITH PERCENT ERROR FROM SCENE VIEWED
  FROM FAR AND UP CLOSE. ..................................................................................................................................... 49

TABLE 8: CAMERA SPECIFICATIONS FOR LEFT AND RIGHT CAMERAS. (WATEC, 2008) .......... 54

TABLE 9: ENCODER SPECIFICATIONS FOR STEREO SYSTEM. (AXIS COMMUNICATIONS, 2009)...... 54

TABLE 10: RESULTS FROM STEREO MATCHING: STATIC (TOP) AND DYNAMIC (BOTTOM).
              MEASUREMENTS ARE DISTANCES TO THE REAL WORLD POINTS............................................ 58

TABLE 11: ERROR IN DISTANCE MEASUREMENT USING CAMERA CORRECTIONS. ............................. 59
Chapter One: Introduction

The concept of using two views of a scene from a camera to extract depth information has been around since 1832 when Sir Charles Wheatstone invented the stereoscope (Encyclopaedia Britannica, 1987). A stereoscope is simply a device that was designed to view stereographs, two images of the same scene on a card differing only in horizontal placement of the camera. When the stereoscope is used, the left and right eyes are presented with the two different views of the scene. The two images are blended together as if the user was actually viewing the scene, with all the lifelike depth information present. From this, modern stereo vision, as performed by a computer system utilizing differing views of a scene taken from digital cameras, was developed.

1.1 Background

For autonomous systems, being aware of the surrounding environment is important for decision making. For a computer system, having access to images of the environment obtained from digital cameras provides key data to determining its surroundings for decision making or to provide to a human user to help in decision making, as in a semi-autonomous system. Depth plays an important role in determining an environment around a system, as it can provide distances from the system to objects within the surroundings. Therefore, it is important to have a means to obtain this data without knowing it a priori, and one such way to calculate distance automatically is with a stereo vision system.

Using slope and best fit planes analysis, potential landing sites can be found from terrain models for an Unmanned Aerial Vehicle (Klomparens, 2008), shown in Figure 1. Using Hough transform line detection algorithms (Yang & Gillies, 2004), potential hazards above acceptable landing sites, such as power lines (Hager, 2009) shown in Figure 2, can also be found from data extracted from these models. In order to analyze the terrain in both of these scenarios, 3-D un-gridded surface points, known as point clouds, were generated from a stereo vision system and software developed by Point Grey Research (Point Grey Research, 2009), shown in Figure 3.
Figure 1: Results of best-fit planes for landing site estimation. (Klomparens, 2008)

Figure 2: Results of Hough line detection for detecting power lines. (Hager, 2009)

Figure 3: Bumblebee stereo system from Point Grey Research.
The key to stereo vision processing is that the relation of the left camera with respect to the right camera is known. This knowledge of positioning and characterization of a stereo system is known as a calibrated stereo system. Any deviation from the calibration state will introduce errors that will corrupt the data obtained from stereo processing.

1.2 Problem Statement

The motivation behind this thesis is the development of a turnkey ‘in-house’ custom stereo vision system for real-time 3-D point cloud generation to build the models necessary for analysis. As mentioned earlier, the Bumblebee and other commercially available stereo vision systems currently exist. The problem is that these systems are designed at the company’s discretion and their software tools for stereo processing are proprietary. Having software tools built to control a static stereo vision design limits its usefulness. For more dynamic applications, it becomes necessary to have a set of tools that can be used with various system specifications which use a wide range of cameras and interfaces. The objective is to map terrain from high altitudes with a stereo vision system for use with a deployable ground robot for sampling. The ground robot would use the terrain maps to find suitable terrain to traverse. The VTOL UAV, a Yamaha RMAX, with a stereo system attached is shown in Figure 4.

![RMAX helicopter used for imaging flight, with stereo vision system attached.](image)
When dealing with high altitude terrain mapping, it becomes necessary to implement a wide baseline stereo system in order to recover depth resolution lost by using a smaller system, commercially available systems typically are not larger than 0.6 meters. Mounting a wide baseline stereo vision system to a vehicle, such as a VTOL UAV, can introduce errors from vibration to a non-rigid system. These errors arise from camera pose of the left and right cameras deflecting from that of their calibrated state. This error from vibration must be addressed in order to produce accurate terrain information for analysis. Inaccurate terrain information could lead to re-projection of a model larger than it really is, causing a landing site to be misclassified as suitable or an object being projected further from a system attached to a robot than it really is, potentially causing a collision.

Currently, there are a few methods for checking the internal parameters of a stereo system after it is active. Two methods are provided by Marita et al. (2006), one using infinity points and using epipolar line constraints to check drift of the parameters from the initial values, due to vehicle vibration and temperature. In the infinity points method, the authors consider points on the horizon (>100 m) to be points in a plane at infinity. With this assumption, the expected disparity of matching points lying on a plane at infinity can be calculated based on the intrinsic and extrinsic parameters of the system from calibration. In this method, if the difference between the calculated average disparity of infinity points and the expected disparity is below a certain threshold, than the system is considered to have been unchanged.

The epipolar method states that a vertical epipolar drift is present when changes to the focus of the camera occur, from road vibration. In this method, an epipolar drift is assumed and the value of the drift is found by moving the epipolar lines in the positive and negative direction, while computing the number of 3-D points that were found. Usually when a drift is occurring, the number of 3-D points found in re-projection is decreased, so the more 3-D points resulting from correlation, the closer the shift is to the correct state. The drift value in the system is the offset from the current state to the shift that produces the greatest number of 3-D points. Althought both of these methods were found to correctly determine if there is an error in a system, however neither provide a solution for correcting the errors.
1.3 Contribution

This thesis describes the process of building a custom stereo vision for real-time stereo processing, from camera selection to re-projecting points to 3-D point clouds. Using the Open Source Computer Vision Library (OpenCV), stereo vision algorithms have been implemented in order to produce 3-D point clouds from stereo images and a single calibration file containing the intrinsic parameters of the left and right cameras and the position of the left camera with respect to the left. From the resultant point clouds, various tasks such as landing site estimation and robot path planning can be performed. Results from stereo vision systems created and processed using the tools described and developed in this thesis are shown to be well representative of the 3-D scene they model.

Performances of custom systems of measuring distance to objects are characterized as well as size of objects in the scene. In addition, a method of object detection is shown that extracts objects within a certain range of a system mounted to a ground vehicle. The components can then be relayed to a ground control station for analysis of distance and size of the objects, in scenarios where the size of an opening for the robot to fit through is in question. Finally, a method for correcting camera pose deflection from their calibrated state by mounting accelerometers to the cameras is discussed. The angle of deflection of the cameras are found and applied to the images before processing. Experimental results are shown to reduce the distance measurement error when the corrected angle is applied.

1.4 Organization of Thesis

This thesis contains four chapters describing in detail the work involved with the contributions to the mission described in Section 1.2. Chapter 2 contains the stereo vision theory used in the functions to calibrate and process stereo images from custom built systems. The tools that were developed for stereo processing and analysis are described in Chapter 3. In Chapter 4, a variety of custom built stereo vision systems that were contributed to the mission are described with results from terrain mapping at various heights, object detection, and re-projection performance analysis. Chapter 5 contains the results from camera pose corrections resulting vibration of a flexible or large baseline stereo vision system.
Chapter Two: Binocular Vision

This section describes the theory behind stereo vision and the process of calibrating and rectifying images for correlation and 3-D re-projection.

2.1 Stereo Vision

Binocular vision is defined as vision from two eyes where the data being perceived from each is overlapped by some amount. The overlap from the two different views is used in biological vision to perceive depth. Stereoscopic vision is the use of binocular vision to perceive the three-dimensional structure of the world. Binocular disparity is the difference in the placement of objects as viewed by two eyes due to the different viewpoints from which each views the world. Stereopsis is referred to as the impression of depth extracted from binocular disparity (Howard & Rogers, 1995). A stereo vision system is a set of two or more cameras, used by machines to extract depth of a 3-D scene as viewed from different vantage points, as modeled after binocular vision in humans. Figure 5 demonstrates how stereo vision is present in humans through the use of two eyes viewing a scene from different vantage points to extract depth. In humans this is known as depth perception.

Figure 5: Example of binocular vision in humans (Cooper, 1995).

In a stereo vision system, cameras are horizontally aligned and separated by a distance known as the baseline. Figure 6 shows an example stereo vision system with four cameras
mounted on a bar. Using one of the cameras on the left and one of the cameras on the right will provide the two images necessary to extract a disparity map, which provides the data needed for 3-D reconstruction.

Stereo ranging is illustrated with the simple arrangement that is shown in Figure 7. In this ideal system, the optical axes of the two cameras are perfectly parallel, both image planes are coplanar, and no lens distortion is present. In this figure, the distance to point \( P \) in the scene is to be determined to provide its 3-D coordinate. In this case, the distance \( Z \) (also called the range or depth) can be found using the following equation:

\[
Z = \frac{Bf}{d}
\]

In this equation, \( f \) is the focal length, \( B \) is the baseline, and \( d \) is the disparity which is defined by \( d = x^l - x^r \).

Figure 6: Stereo vision system with 4 cameras. The user has the choice of 2 cameras on the left and 2 cameras on the right. This image was taken of a stereo vision system built at the Unmanned Systems Lab at Virginia Tech.
Figure 7: Simple geometry for stereo ranging. The usual goal is to find the range $Z$ from the cameras to a point $P$ in the scene.

2.2 Camera Calibration

In this section, the process of calibrating a camera to determine the intrinsic parameters and distortion vector is discussed.

2.2.1 Overview

In order to apply stereo ranging techniques with a reasonable level of accuracy, it is important to calibrate the camera system. The calibration process provides numerical values for intrinsic camera parameters such as focal length, and extrinsic parameters such as relative orientation angles between the two cameras. In principle, any characterized object can be used as a calibration target, as long as the 3-D world coordinates of the target are known in reference to the camera (Bradski & Adrian, 2008). There are calibration methods that utilize three-dimensional objects, such as a box covered with markers, but planar checkerboard patterns are the easiest to calibrate with.

We have used the camera calibration method that is described in (Heikkila & Silven, 1997). This method was implemented in a camera calibration toolbox developed for MATLAB.
(Bouguet, 2008) and is based on the OpenCV implementation developed in the C programming language. The MATLAB version was chosen based on the ease of use and better performance; the OpenCV implementation utilizes an automatic corner finder without any user input, which was found to be unpredictable at finding corners, with performance dropping drastically as the camera was moved further from the calibration target. This method appeared to be tailored for close range calibration. A guide for using the MATLAB toolbox for stereo calibration is listed in Appendix C: Camera Calibration Guide using Matlab. The calibration technique assumes the use of a checkerboard pattern as shown in Figure 8. The cameras are placed so that this pattern appears in both images. The system automatically detects corners, after the perimeter of the board is selected manually, within the checkerboard in both images, using an initial guess at radial distortion if needed. Multiple views of the same checkerboard are taken by either moving the cameras or moving the checker board. To achieve high quality results, it is recommended that at least ten images of a 7-by-8 or larger checkerboard is used. It is also necessary to move the checkerboard or camera to obtain images of the calibration target at multiple angles (Shapiro & Stockman, 2001).

The 3-D coordinates of the corners are provided to the system. Using knowledge of these 3-D locations along with the detected corner pixel locations in the images, the calibration procedure solves for the rotation and translation vectors that relate the left and right cameras as well as the intrinsic camera parameters.

![Figure 8: Checkerboard pattern used as calibration target. This target has a height of approximately 18 feet.](image-url)
2.2.2 Camera Model

Perspective projection can be represented as \( q = MQ \), where \( Q \) is a 3-D scene point given with camera-centered coordinates, \( q \) is the resulting 2D image point, and \( M \) is called the camera matrix. The 3\( \times \)3 matrix \( M \) contains the intrinsic parameters for the camera:

\[
M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}
\]

The parameters \( c_x \) and \( c_y \) represent the image center, which is determined by the optical axis of the lens, and is typically different from the center of the CCD sensor array. Ideally, the optical axis should intersect the exact center of the sensor, but in practice the sensor is displaced by a few pixels. The parameters \( f_x \) and \( f_y \) represent horizontal and vertical scale factors, respectively. Two parameters are needed because most sensors have rectangular (non-square) pixels. These two parameters combine the lens focal length, \( f \), with the physical size of the individual elements on the sensor array: \( f_x = f \ast s_x \) and \( f_y = f \ast s_y \), which is the actual physical focal length multiplied by the size each element on the sensor in pixels (pixels per mm).

When using a standard glass lenses, as opposed to a pinhole lens, lines that are straight in the 3-D world will sometimes appear to be curved in the image. The effect is more pronounced near the borders of the image. This is called barrel distortion or pincushion distortion or, more generally, radial distortion. Barrel distortion is illustrated in Figure 9. Figure 10 shows an actual image containing barrel distortion along with the corrected image.

![Figure 9: Example of barrel distortion effect.](image-url)
The corrected image in Figure 10 was obtained by “warping” or “stretching” the original image according to the following equations:

\[ x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \]
\[ y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \]

Here, \((x, y)\) is the location of a pixel in the original image and \((x_{\text{corrected}}, y_{\text{corrected}})\) is the new location of this pixel in the corrected image. The variable \(r\) represents distance from the image center. The constants \(k_i\) are taken from a Taylor series expansion around \(r = 0\), where only the even power coefficients are needed to keep the function symmetric. One goal of a camera calibration process is to solve for the \(k_i\) values.

An example of a radial distortion plot for a specific camera lens is shown in Figure 11. The blue arrows show how points are distorted and how they are shifted when correcting the image.
2.2.3 Rectification

Rectification of a stereo image pair is the process of transforming (“warping”) the two images so that corresponding points lie on the same image rows. More formally, the transformation causes epipolar lines (see below) to become collinear. Rectification is commonly performed to improve the speed of the search for stereo correspondences.

The stereo imaging geometry is illustrated in Figure 12a. A scene point $P$ projects onto $p_l$ and $p_r$ in the two images. The two points of projection, $O_l$ and $O_r$, together with $P$ determine an epipolar plane. The epipolar plane intersects each image plane to form two epipolar lines. The significance of the epipolar geometry is that any image point, say $X_L$ in the left image (Figure 12b), determines an epipolar line (shown in red) in the opposite image that must contain the corresponding point if it is present. This “epipolar constraint” is important for reducing computation time during the search for correspondences, because it effectively reduces the search to a one-dimensional search as opposed to searching 2D image regions. This constraint also reduces the number of incorrect matches.
Referring again to part (a) of the figure, the epipoles (marked as $e_l$ and $e_r$) are the points in the image planes that intersect the line $O_LO_R$ which connects the two camera centers. Notice that all epipolar lines in an image must pass through that image’s epipole. In Figure 12a, the two epipolar lines can be represented as $X_le_L$ and $X_re_r$.

Figure 12b also illustrates how multiple points in 3-D can be mapped as a single point in the left image and as multiple points along the epipolar line in the right image. Considering a point $X_L$ in the left image, it is not possible to determine which of the points, $X_3$, $X_2$, $X_1$, or $X$ it is corresponding to, since the depth of the point cannot be determined from a single vantage point. Finding the correct correspondence to $X_L$ in the right image, however, will actually determine the 3D location of the object.

Figure 12: Stereo imaging geometry. (a) Epipolar plane, which is the plane formed by the 3D point $P$ and the center of projections. (b) Epipolar plane with multiple 3D points along right epipolar line corresponding to one point on the left epipolar line.

In order to determine the epipolar lines, two matrices need to be defined, the essential matrix and the fundamental matrix. The difference is that the essential matrix, $E$, relates the left and right cameras in 3-D physical space (maps $p_l$ to $p_r$), whereas the fundamental matrix, $F$, relates the same two camera views, but in pixel coordinates (maps the variable $q_l$ to $q_r$). The essential matrix contains the rotation and translation information from the left camera to the right camera in physical coordinates. The fundamental matrix contains two parameters describing the
left and right epipoles as well as three parameters for the homography that relates the left image plane to the right.

Taking a point $P$, which is viewed as $P_l$ in the left camera and $P_r$ in the right, the relation of $P_l$ to $P_r$ can be obtained using the rotation and translation vectors. This relation is the following:

$$P_r = R(P_l - T)$$

In order to define the epipolar plane, the definition of a plane with respect to the normal to the plane $n$ and a point defined in that plane $a$ is used, which holds for all points on the plane $x$:

$$(x - a) \cdot n$$

Since the vectors $P_l$ and $T$ (the vector connecting the center of projections) are contained in the epipolar plane, the normal to these vectors can be used as the vector $n$. Simply taking the cross product of these vectors will give us a normal vector to the epipolar plane. Here all possible observed points for $P_l$ can be inserted, shown in Figure 11 on the right at various depths, through the point $T$, where the vector $O_lO_r$ intersects the epipolar plane is:

$$(P_l - T)^T (T \times P_l) = 0$$

To get rid of the dot product, the identity $a \cdot b = a^T b$ is used in order to perform matrix multiplication. Since the focus is with finding the fundamental matrix for rectification, the relation of $q_l$ and $q_r$ needs to be found. This can be accomplished by first relating the physical coordinates of the projection $P_l$ and $P_r$. $P_r$ can be substituted in by using the equation above relating $P_l$ to $P_r$. Solving for $P_l$, gives the following:

$$P_r R^T + T = P_l$$

Which can be substituted into the equation to give:

$$(P_r R^T + T - T) (T \times P_l) = (P_r R^T) (T \times P_l) = (P_r R) (T \times P_l) = 0$$

The final substitution made is for the cross product of $T$ (the vector $\{T_x, T_y, T_z\}$) and $P_l$, where $T$ is re-written as the matrix $S$ using the following relation:
\[ a \times b = \begin{bmatrix} 0 & -a_1 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \]

which defines \( S \) as:

\[
\begin{bmatrix} 0 & -T_x & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}
\]

and thus giving us a final result for our relation of \( P_l \) to \( P_r \) of:

\[ P_r^T R S P_l = 0 \]

Here the essential matrix \( E \), is defined as the product of \( R^*S \), which provides the following:

\[ P_r^T E P_l = 0 \]

Since the focus here is for the relation of \( q_l \) and \( q_r \), the relation of the points on the imagers \( p_l \) and \( p_r \), needs to be found. These can be substituted in using the projection equation; this provides the following for the relation:

\[ p_r^T E p_l = 0 \]

Since \( E \) does not contain information about the cameras intrinsic parameters, it can only be used to relate the points observed by the left camera to those observed by the right camera in physical coordinates, not in pixel coordinates, the fundamental matrix is needed in order to perform rectification.

Using the relation to project points in the physical world into camera coordinates, \( q=MQ \), where \( q \) are the physical coordinates, \( M \) is the camera matrix, and \( Q \) are the camera coordinates, the relation of the physical points in each image can easily be converted to the relation in camera coordinates. Taking \( q=MP \), gives \( p=M^l q \) and provides the following when substituting for \( p \):

\[
(q_r^T M_r)^T E (q_l^T M_l) = 0
\]
Defining the Fundamental matrix $F$ as

$$F = M_r E M_l^T$$

the following relation for the point $P$ in camera coordinates is given:

$$q_r^T F q_l = 0$$

Given two cameras where the intrinsic parameters are known, the images undistorted, and the image planes have been rectified, the cameras can be calibrated so that the rotation and translation of one camera is known with respect to the other. These relations are found in the rotation matrix and the translation vector. The rotation matrix contains parameters that rotate the left camera so that its image plane is mathematically coplanar with the right camera. The translation vector relates the positioning, or offset, of the left camera with respect to the right camera.

Using the relation of the two views of $P_r = R(P_l - T)$, and the relation of the object viewed in physical coordinates of and using the relations of $P_l = R_l P + T_l$ and $P_r = R_r P + T_r$, the following relation for rotation and translation is obtained:

$$R = R_r R_l^T,$$

$$T = T_r - R T_l$$

2.3 Correlation

During correlation, un-distorted and rectified images from the left and right camera are used to match points from one image to the other. In order to determine the distance from the camera, the disparity needs to be found, change in location of points in the left image to the right. It follows that there must be an overlap in the two images so that a point in the left image also exists in the right image and a correspondence can be found.

An algorithm that can be used to compute correlation and find disparities is a block matching technique where sum of absolute distances (SAD) windows are used to find correspondences. SAD windows are used as a scoring method for each pixel in the image based
on its surrounding neighbors. There are three steps to the block matching technique that OpenCV uses; prefiltering, correspondence search, and post filtering.

During this stage, the left and right images are normalized so that they have the same lighting levels. A window of variable size is placed over each pixel and the pixel is replaced using the following:

\[
\text{min} \left[ \text{max}(I_c - \bar{I}, -I_{\text{cap}}), I_{\text{cap}} \right]
\]

In this equation, \(\bar{I}\) is the average intensity value in the window and \(I_{\text{cap}}\) is an upper limit which is preset. \(I_c\) is the intensity of the pixel that the window is centered over, the one that will be changing.

Here the correspondences of the points in the left image to those in the right image are found. After rectification, the rows in the images are aligned so that the corresponding points should theoretically lie along the same line number in both images. The SAD window is set at a pixel in the left image and a score is calculated. The algorithm then searches the right image starting with the same coordinate as the left and moves to the left, along the \(x\)-axis, calculating scores for each pixel location until it reaches the maximum disparity. Disparity is the number of pixels offset from the initial pixel that is being looked at for a correspondence. The SAD value is calculated using the following equation:

\[
SAD(r, c) = \sum_{y=-w}^{y=w} \sum_{x=-w}^{x=w} |\text{Right}(y + r, x + c + d) - \text{Left}(y + r, x + c)|
\]

In this equation, \((r,c)\) is the point being searched for a correspondence in the right image, \(d\) is the disparity of the point in the right image from the original point, and \(w\) is the size of the window that is placed over each point. From this equation, it is shown that the scores are calculated based on the intensity values of the neighboring pixels surrounding the point. The point in the right image within the search area with the lowest score is considered the best match for the point in the left image. The offset of this point from the original point in the left image is taken as the disparity for that correspondence and from that the equations above are used to compute the depth. Points with large disparity values represent points that are closer to the camera and smaller disparities represent points that are farther away from the camera.
Post filtering is done to remove correspondences that are considered false matches. For this, OpenCV uses a uniqueness ratio as well as a texture threshold. The uniqueness ratio is used to make sure that the value that was calculated for the matched point is not just the closest score, but is an outlier score where it is surrounded by scores that are far from being a match. The texture threshold is set so that noise can be reduced during the matching process, not score that is below the texture threshold is considered.

The result of the correspondence routine is an image with each pixel being the disparity that was found from the left and right images, this image is called a disparity map. An example of a disparity map with the original image is shown in Figure 13. Brighter intensity values represent objects that are closer to the camera where darker objects are those farther away from the camera. Black pixels are those points where no correspondence was found between the images.

Figure 13: Color image of scene (left) and disparity map (right), increasing window size when computing SAD will produce smoother objects and less data where decreasing the window will have more data but less defined objects.
2.4 3-D Re-projection

Re-projecting 2-D points from a set of images to 3-D points is accomplished using the disparity values for each pixel. Each pixel has a corresponding $(X, Y, Z)$ coordinate that describes its position in the 3-D space. The $Z$ coordinate is found using the equation relating distance as a function of baseline, focal length, and disparity shown below:

$$Z = \frac{Bf}{d(\text{sensor element size})}$$

In this equation, the disparity value found by the correlation routine is multiplied by the actual size of an individual element on the imaging sensor, providing the disparity as a distance metric. The $X$ and $Y$ values are found by finding the size of a pixel at its calculated distance found for $Z$. First, the Field of View (FOV) must be known or calculated using the following equation:

$$FOV = 2 \cdot \arctan\left(\frac{\text{size of sensor}}{2f}\right)$$

The size of sensor value is the total vertical size for calculating vertical FOV and horizontal size for the horizontal FOV. Once the FOV is calculated, the size of a pixel can be found at a given distance using the following equation:

$$\text{actual distance covered by image} = 2 \cdot \sin\left(\frac{FOV}{2}\right) \cdot Z$$

Using the horizontal FOV or vertical FOV will give the horizontal and vertical distance covered by the image. Taking this measurement and dividing it by the total number of horizontal or vertical pixels provides a value for the distance covered per pixel at the distance $z$ from the camera. Finally, using the camera center coordinates from the left camera, $c_x$ and $c_y$ the real world $X$ and $Y$ coordinates can be determined from the pixel coordinates $x$ and $y$, using $(c_x, c_y)$ as the center of the projection. The following equations show how $x$ and $y$ are calculated:

$$X = \frac{2 \cdot \sin\left(\frac{\text{FOV}_{\text{horizontal}}}{2}\right) \cdot z}{\text{image width}} \cdot (x - c_x)$$

$$Y = \frac{2 \cdot \sin\left(\frac{\text{FOV}_{\text{vertical}}}{2}\right) \cdot z}{\text{image height}} \cdot (y - c_y)$$
Using these equations, a point cloud can easily be re-projected into any measurement system desired from the pixel coordinates and their corresponding disparity values. An example of a point cloud that was re-projected from a stereo pair is shown in Figure 14. The point cloud was viewed in a point cloud viewer developed in C, using the Open Graphics Library (OpenGL), by a graduate of the Unmanned Systems Lab, Dylan Klomparens (Klomparens, 2008).

![Figure 14: 3-D re-projection (right) of a 2-D image (left) from stereo pair.](image)

### 2.5 Stereo vision considerations

This section discusses the process of selecting cameras for a stereo vision system based on different characteristics that a camera and its lens provide as well as the effects of selecting a baseline for the system.

#### 2.5.1 Camera Selection and System Design

The first step in building a stereo system for 3-D reconstruction is the selection of a camera and a lens. The main things to consider are the resolution, field of view, and focal length since all these factors influence the area that can be used for stereo matching. It is not
recommended to use a lens with a focus control, because a change in focus setting throws off the calibration of the stereo system.

Resolution of the cameras plays an important role in the overall accuracy of the stereo system. Resolution often refers to the number of pixels in the sensor array of the camera, which is typically a CCD (charge-coupled device) containing a rectangular array of light-sensitive elements. For many applications the effective resolution is of interest, and this term refers to the area within the 3-D scene which projects onto a single pixel of the sensor array. Effective resolution therefore depends on the focal length of the lens and on the distance of the camera from the imaged surface in the scene. It is common to refer to the effective resolution using units of pixels per inch (PPI), and for convenience this may be considered in one dimension only. Figure 15 contains plots of effective resolution for several sensor array sizes and at several depths.

![Figure 15: Horizontal distance covered per pixel at various depths and various camera resolutions using a focal length of 3.8 mm.](image)

The field of view and focal length of the lens are inversely related. By increasing the focal length, the field of view can be reduced and simultaneously increase the effective resolution. Reducing the focal length will increase the amount of the scene that can be imaged,
but will reduce the effective resolution. When considering a stereo vision system, the field of
view is important as there needs to be a significant amount of overlap in the images from the two
cameras. Naturally, stereo ranging is possible only for 3-D scene points that appear in the
images of both cameras.

For a stereo ranging system, the resolution and accuracy of recovered depth values
depend in part on the imaging resolution described above. Accuracy further depends on
identifying the correct matching points in the images. Assuming that the correct
correspondences have been found, the depth resolution, $\Delta Z$, can be calculated using the
following equation:

$$\Delta Z = \frac{Z^2}{fB} \Delta d$$

In this equation, $Z$ is the distance from the camera, $f$ is the focal length of the lens, $B$ is
the baseline, and $\Delta d$ is the size of one pixel element on the sensor. In some lower end sensors,
the element size will be different for height and width. Since the correlation algorithm searches
along the horizontal axis, the width of the pixel element is used for this value. Using the values
of 3.8 mm for the focal length and 0.0049 mm for $\Delta d$, as found in the data sheet for the cameras
in the system from Figure 6, the graph in Figure 16 can be generated showing the depth
resolution at various distances using baselines of 18 and 12 inches.
Figure 16: Depth resolution vs. distance to object of two stereo systems with baselines of 12’’ and 18’’.

The pitch, or baseline, is the distance that the left camera is offset with respect to the right camera. The selection of this distance is important, and depends on the application. If the baseline is small, the system is more suited for modeling objects that are close to the camera, whereas a large baseline is more suitable for objects farther away. The graph in Figure 17 shows how the baseline affects the depth resolution of the stereo vision system. In this graph, the higher resolution can be seen when viewed with a wide baseline system when looking at close objects. However the problem that occurs is that overlap between the images is lost with a wide baseline and therefore correlation suffers when viewing close objects. Decreasing the baseline between cameras for closer objects fixes this problem.
2.5.2 Stereo Processing Latency

Another aspect in camera selection is its effect on the performance of the stereo vision correlation speed. Characteristics such as resolution of the camera have as much effect on the performance of the correlation process as does changing the number of disparities to search. A larger image will take longer to process than a smaller image; however a larger image will have more points per millimeter than will an image taken from a lower resolution image. In fact, a 1600 x 1200 pixel camera has 2.27 more pixels per mm than does a 704 x 480 image. However, a 1600 x 1200 pixel camera takes 1.2 seconds longer (3.6 times as long) to perform a correlation than a 704 x 480 camera. When considering a real time stereo system on a UAV, creating 3-D models in a timely manner is important. In some scenarios it might be beneficial to sacrifice quality of the models to increase the rate at which models can be generated. In Table 1 and Figure 18, a comparison between correlation times of a 704 x 480 and a 1600 x 1200 image are shown. The results were produced by running the correlation routine 100 times and computing the average execution time. Performance of execution time using an increasing number of disparities for correlation is also shown. This test was performed on an Intel Core 2 Duo processor running Windows 7 64-bit Operating System. This comparison also shows the effects of increasing the number of disparities that the correlation routine searches for each pixel.
Table 1: Execution times for correlation routine on a 704x480 and 1600x1200 image.

<table>
<thead>
<tr>
<th># of Disparities</th>
<th>704x480</th>
<th>1600x1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.16 (s)</td>
<td>0.98 (s)</td>
</tr>
<tr>
<td>128</td>
<td>0.46 (s)</td>
<td>1.69 (s)</td>
</tr>
<tr>
<td>256</td>
<td>0.69 (s)</td>
<td>4.93 (s)</td>
</tr>
</tbody>
</table>

Figure 18: Graph showing correlation routine execution time of two images.

Considering two systems designed with the same baseline and the same lens field of view, one using a set of 704x480 cameras (system A) and the other using a set of 1600x1200 cameras (system B). Looking at the execution times in Table 1 and taking into account that system B has 2.27 time as many pixels per meter as system A does, correlation on a set of images taken of the same scene at the same distance would require 2.27 time the number of disparities searched in system B as it would in system A. In short, the same disparity for a pixel point in system B would be 2.27 times the disparity in system A. In this scenario, system B would require a disparity search greater than that of system A. With this in mind, if system A is set to search a depth of 128 disparities then system B would need to be set to 256 disparities to search the same set of images of the same scene. From the execution times calculated in Figure 18, system B would take 4.47 seconds longer (10.7 times as long) to complete correlation as...
would system A. From these results, it is obvious the importance of taking resolution of the cameras selected for the stereo system into consideration and the need to weigh the costs of execution time versus quality of the 3-D model.
Chapter Three: Stereo Vision System Tools

The Open Computer Vision (OpenCV) Library is an open source computer vision library developed by Intel and is freely available for download and use in commercial and research areas. The libraries are written in C and C++ and run under Linux, Windows, and Mac OSx. OpenCV is actively being ported to other languages such as Matlab and C#. This library holds over 500 functions for many different areas in computer vision such as medical imaging, security, camera calibration, robotics, and computer vision. The three main components of OpenCV that are used in stereo vision research are the CV, HighGUI, and CXXCORE components. The CV component is used for image processing and contains all the computer vision algorithms that are used for stereo image processing. The HighGUI component is used for reading, writing, and displaying images as well as creating the slide bar tools used for adjusting parameters for the correlation routine shown in Figure 20. The CXXCORE contains data structures for images and matrices created in the StereoVisionSystemTools project in Appendix A: Stereo Vision System Tools C++ source code.

3.1 Stereo Vision System Tools Overview

The Stereo Vision System Tools found in Appendix A: Stereo Vision System Tools C++ source code were developed in C++ by the author to perform stereo vision tasks with the assistance of the OpenCV library. The tool kit accepts two images and a calibration data file as arguments. The calibration file was created using the Matlab toolkit during the calibration phase. The application is set up to either process the images automatically and output a point cloud, or it can be set to load the images, process them and then display the disparity maps and rectified pair. A slide bar window was created so that various parameters dealing with the correlation process can be adjusted. This process is shown in a flow chart in Figure 19.
This figure illustrates the work flow of the Stereo Vision System Tools that were developed. The stereo system sends a left and right image of the 3-D scene it is viewing to the Stereo Vision System Tools program. With these images, the calibration data, and re-projection data from calibration, the program rectifies the images, correlates the pixels, and re-projects the 2-D points to a 3-D point cloud. A point cloud is simply a set of X, Y, Z coordinates extracted from the x and y pixel coordinates and the disparity associated with each point. This point cloud is then imported to Quick Terrain Modeler (APL, 2003) where it can be viewed, edited, and analyzed. In addition, the color image taken by the stereo camera can be overlaid onto of the point cloud creating a texture map. The following section describes some of the OpenCV functions that are used in the Stereo Vision System tools found in Appendix A: Stereo Vision System Tools C++ source code.
3.2 Stereo Vision System Tools Process

In order to rectify the two images sent to the toolkit, there must be a calibration file present that was created from calibrating the same stereo system that captured the left and right images. This file contains the following information: left and right camera matrices, left and right distortion vectors, rotation matrix, and translation vector. Once this data is loaded, the OpenCV function `cvRectify()` is used to find the 3x3 rectification rotations for the left and right image planes and the 3x4 projection equations that were described in 2.1.2. The `cvRectify()` function is as follows:

```c
void cvStereoRectify(
    const CvMat* cameraMatrix1,
    const CvMat* cameraMatrix2,
    const CvMat* distCoeffs1,
    Const CvMat* distCoeffs2,
    CvSize imageSize,
    const CvMat* R,
    const CvMat* T,
    CvMat* Rl,
    CvMat* Rr,
    CvMat* P1,
    CvMat* Pr,
    CvMat* Q=0,
    Int flags=CV_CALIB_ZERO_DISPARITY
);
```

The rectification function takes in 13 parameters, five of which are returned by the function call. The first two parameters are the camera matrices for the left and right cameras. The next two are the distortion coefficients for the left and right camera lenses, to correct for barrel distortion. The function also takes in the rotation matrix and the translation vector; all of these variables were loaded from the calibration file sent to the program corresponding to the model of the stereo system cameras. The return variables are the rectification rotations for the left and right image planes (`Rl` and `Rr`), the projection matrices for the left and right projections (`Pl` and `Pr`). These variables are used in the next function, `cvInitUndistortRectifyMap()`, which is
called on the left and right images individually. This function is used to define an image to image mapping function, which simply determines where pixels in the original image go in the undistorted, rectified images. This function takes on the following form:

\[
\text{void cvInitUndistortRectifyMap}(
    \text{const CvMat* cameraMatrix},
    \text{const CvMat* distCoeffs},
    \text{const CvMat* Rrect},
    \text{const CvMat* Prect},
    \text{CvArr* mapx},
    \text{CvArr* mapy},
    \text{cv})
\);

This function accepts four parameters as inputs and returns two. The first and second variables are those corresponding to the camera matrix and distortion coefficients of the left or right camera, whichever image the mapping is being computed for. \text{Rrect} and \text{Prect} correspond to the rotation and project matrices that were found in the \text{cvStereoRectify()} function. The return values are vectors of the pixel mapping of the image in the x and y directions. These return variables are used in the \text{cvRemap()} function in order to adjust the pixels from the original image to produce the rectified image for correlation. The \text{cvRemap()} function can be called using the \text{mapx} and the \text{mapy} vectors for each image that is sent to the Stereo Vision System Tools application, provided the correct vectors for the corresponding left and right images are used. The \text{cvRemap()} function is defined here:

\[
\text{void cvRemap}(
    \text{const CvArr* src},
    \text{const CvArr* dst},
    \text{const CvArr* mapx},
    \text{const CvArr* mapy},
    \text{int flags=CV_INTER_LINEAR},
    \text{CvScalar fillval=cvScalarAll(0)},
    );
\]
The first two parameters are the source and destination images; here the destination image is the undistorted, rectified image. The next two arguments are the mapping vectors that describe where each pixel is supposed to be shifted (x and y direction) from the source to the destination image. Since the two mapping vectors do not have to be integer type mappings, an interpolation technique is used to determine where in the destination image the point needs to be placed, and since it could be a floating point value, it would be rounded to an integer value and placed at that pixel location, causing gaps in the destination image. Another reason interpolation is needed is that not all pixels in the destination image have a pixel mapped to them, which also causes gaps in the destination image. The fifth argument describes which interpolation technique is to be used, the default is bilinear interpolation.

Interpolation is a tool used in image tasks such as zooming, shrinking, rotation, and geometric corrections. Defined, interpolation is the process of using known data to estimate values at unknown locations (Gonzalez & Woods, 2008). Here bilinear interpolation is being used for un-distorting and rectifying left and right images. In bilinear interpolation uses the four nearest neighbors of the unknown pixel are used to estimate the intensity of the gaps in the destination image. The four horizontal and vertical neighbors of the unknown coordinate are the 4-neighbors of a pixel p. This type of interpolation simply takes the average of the four neighbors to determine the output intensity for the unknown pixel.

Next the rectified pair of images is sent to the correlation routine to find matching pixels in the image. To do this, the function \texttt{cvFindStereoCorrespondenceBM()} is used, which is defined here:

\begin{verbatim}
void cvFindStereoCorrespondenceBM(
    const CvArr* leftImage,  
    const CvArr* rightImage,  
    const CvArr* disparityImage,  
    const CvStereoBMState* BMState,
    const CvStereoBMState* BMState,
)
\end{verbatim}

The first two arguments are the left and right rectified images, returned from individual calls to the \texttt{cvRemap()} routine. The next parameter is the output disparity image, which is the disparity map containing the offset of a pixel in the left image with respect to its position in the right image.
image. OpenCV employs a 16 bit precision correlation routine, meaning that each pixel has 16 sub pixel locations to which the correlation of a point is checked. Due to this, the values stored in the output disparity image are not the correct disparity values and need to be divided by 16 to obtain the proper disparity for re-projection and determining depth. The final parameter is the state for the parameters used by the correlation routine. This holds values for variables such as the Sum of Absolute Differences window size, which was discussed in Section 2.1.1. In addition, it holds the values for the number of disparities to search and the uniqueness ratio for matched points to accept. The number of disparities variable is used to determine the maximum search length of the correlation routine. This value has different effects on the correlation routine, where lowering it provides a quicker correlation process, as it is searching through fewer potential matches for each pixel then if it was a higher value. Increasing the number of disparities increases the search area, which slows the process, but also provides the possibility of matching points that might have a correspondence that is outside of a lower range. This value should be changed based on the scenario of the scene that is presented to the correlation routine. A scene where objects are far away most likely does not need to have a very high number of disparities to check as does a stereo system with a short baseline where the images are taken from cameras that are close to each other. The uniqueness ratio variable is used to reject matches that do not meet a certain level of uniqueness; this variable is important in order to remove as many falsely matched points as possible. Potential matched points are rejected if the following is satisfied over the correlation window area, determined by the number of disparities:

\[ SAD_{match} - SAD_{min(\text{match \ h})} < uniqRatio \times SAD_{min(\text{match \ h})} \]

A tool was built in the Stereo Vision Toolkit in order to adjust values of certain variables found in the \textit{BMState} variable during analysis mode, variables change immediately and images are correlated again with results updated, eliminating the need to set these variables before running the program. This tool is shown in Figure 20 with the variables for the SAD window size, the uniqueness ratio, and the number of disparities values represented with slider bars. The slider bars can be adjusted to set new values for the parameters and the correlation routine will be executed again on the image set.
After viewing the results of the correlation process, a point cloud can be created and saved for viewing in a 3-D modeler. A point cloud is created by a set of \((X, Y, Z)\) points determined by the pixel location in the image and the disparity found by the correlation process. The disparity for each point is used to first determine the \(Z\) distance of the point to the camera. Using the measured distance \(Z\), the \(X\) and \(Y\) values are also determined for each pixel using the equations defined in Section 2.4. The \(X\) and \(Y\) values correspond to the real world distance of a point from the camera center in meters.

Once the point cloud is generated and each point is saved to a text file, it is loaded into a viewer where it can be edited and viewed. In a viewer such as Quick Terrain Modeler, the color image that was taken with the left and right black and white images can be overlaid as a texture map on top of the point cloud, to provide a realistic looking model of the terrain viewed from the stereo system.
Chapter Four: Custom Stereo Vision Systems

From the stereo vision theory and the stereo vision tools outlined in Chapters 2 and 3, a number of stereo vision systems were designed, built, and calibrated for terrain mapping and for object detection. The design specifications for these systems as well as the results obtained from them are described in this section. Each system will be described in Section 4.1 and the results from all the systems will be shown in Section 4.2. All of the systems were calibrated following the calibration guide found in Appendix C: Camera Calibration Guide using Matlab. All images obtained from the calibrated stereo systems were processed using some version of the stereo system tools developed in Appendix A: Stereo Vision System Tools C++ source code.

4.1 Stereo Systems

The first stereo system shown was with an 18” baseline using two Watec color cameras with a focal length of 3.8mm. The specifications for the cameras are listed in Table 2 and the stereo system is shown in Figure 21.

![Stereo system with 18” pitch using two color cameras from Watec.](image)

Figure 21: Stereo system with 18” pitch using two color cameras from Watec.
Table 2: Specifications of Watec cameras used in 18” Stereo System.

<table>
<thead>
<tr>
<th>Watec 230 Vivid G3.8 mono board</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensor Type</strong></td>
</tr>
<tr>
<td><strong>Unit Cell Size</strong></td>
</tr>
<tr>
<td>(physical pixel size on sensor)</td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
</tr>
<tr>
<td><strong>Minimum illumination</strong></td>
</tr>
<tr>
<td><strong>Focal Length</strong></td>
</tr>
<tr>
<td><strong>Connection</strong></td>
</tr>
</tbody>
</table>

The second system that was designed also has an 18 inch baseline, but utilizes cameras with much higher resolution. This system was built for testing and to demonstrate the quality of a point cloud generated using higher resolution cameras versus one with lower end cameras. This system also uses two black and white cameras for stereo processing and a third color camera to obtain images for a color texture overlay onto the point cloud. Since the stereo vision routines process images in a grayscale format, this has no effect on the performance of the stereo processing. Figure 22 shows the left and right cameras and the color camera used for this system. The camera specifications for each of the cameras in this stereo system are shown in Table 3 and Table 4. The cameras are from Sony and the lenses mounted on the cameras are from Pentax.

![Figure 22: 18” baseline stereo system using 2 B&W and 1 Color Sony cameras.](image-url)
Table 3: Camera Specifications for Sony XCDU100 with Pentax C60607KP - H612A(KP) lens. (Sony, 2009) (Pentax, 2009)

<table>
<thead>
<tr>
<th>Sony XCDU100 with Pentax C60607KP lens</th>
<th>Sony XCDU100CR with Pentax C60607KP lens</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensor Type</strong></td>
<td><strong>Sensor Type</strong></td>
</tr>
<tr>
<td>1/1.3-type Progressive Scan IT CCD</td>
<td>1/8-type Progressive Scan IT CCD</td>
</tr>
<tr>
<td><strong>Unit Cell Size</strong></td>
<td><strong>Unit Cell Size</strong></td>
</tr>
<tr>
<td>(physical pixel size on sensor)</td>
<td>(physical pixel size on sensor)</td>
</tr>
<tr>
<td>4.4 µm x 4.4 µm</td>
<td>4.4 µm x 4.4 µm</td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
<td><strong>Resolution</strong></td>
</tr>
<tr>
<td>1600 x 1200</td>
<td>1600 x 1200</td>
</tr>
<tr>
<td><strong>Focal Length</strong></td>
<td><strong>Focal Length</strong></td>
</tr>
<tr>
<td>6 mm</td>
<td>6 mm</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td><strong>Connection</strong></td>
</tr>
<tr>
<td>IEEE 1394</td>
<td>IEEE 1394</td>
</tr>
</tbody>
</table>

The third system uses the same cameras as described in Figure 21 and Table 2. For this system, the baseline between the left and right cameras is 14.5”. This system was used for object detection and to measure distances to objects. The results from this system, shown in Figure 23, are listed in Section 4.3.

![Figure 23: 14” baseline stereo vision system with lenses visible through hole in white cradles at the ends.](image_url)

The final stereo system that will be discussed is shown in Figure 24, which uses the same cameras as described in Table 3 and...
Table 4. The much wider baseline of this system was selected for mapping terrain at an altitude of 60 m from a VTOL UAV. A system with a smaller baseline would lack in differing perspective views, at such a height and would suffer in depth resolution as shown in the graph of Figure 16.

![Figure 24: 5' Stereo Vision System with Sony XCDU100 and XCDU100CR cameras.](image)

In Figure 25 and Figure 26, the three cameras used in the system are shown. The cameras are linked by fire wire cables by ‘daisy chaining’, where the left camera is connected to the computer, the color camera is connected to the left camera, and the right camera is connected to the color camera. With this set up, all cameras are linked through one connection. Figure 25 shows the left and color camera, where the color is flipped and placed over the left camera. The reason it is placed over the left camera is that when the point cloud is generated, it is computed off of correlations relating the right to the left camera, so the left camera image is the reference image. Orienting the color camera over top of the reference camera is the easiest way to match up a color texture map to a generated point cloud. The same idea holds for the placement of the color camera of the system designed in Figure 22.

![Figure 25: View of left and color cameras from 5' stereo system.](image)  ![Figure 26: View of right camera for 5' stereo system.](image)
4.2 **Stereo Vision Intermediate Results**

In this section, the results from un-distorting, rectifying, and generating disparity maps of a stereo vision system will be discussed. The images obtained from the system in Figure 23 will be used for stereo processing. The first step is using the calibration file that was built from calibration to un-distort and rectify the images using the stereo vision system tools outlined in Chapter 3. The images were gathered of objects set up in the Unmanned Systems Lab. Figure 27 and Figure 28 show the images before and after un-distortion and rectification have been applied.

In Figure 27, left and right images are shown before un-distortion or rectification has been applied. From Chapter 2, it is necessary to rectify the images so that they are row aligned, as the correlation routine searches along the x axis for pixel matches. Using the green lines representing the epipolar lines in this figure, it can easily be seen that the objects are not horizontally aligned. Looking and towards the edges of the two images, the barrel distortion present in the left and right images can be seen.

![Figure 27: Left and Right images taken from 14” stereo vision system before rectification and un-distortion.](image-url)
Figure 28: Left and Right images taken from 14” stereo vision system after rectification and un-distortion.

In Figure 28 the undistorted, rectified left and right images are shown. In this figure, it is obvious how the distortion has been corrected by looking at the edges of the image, namely the lines indicating the edge of the bay door. The effects of the rectification process are also easy to see by the objects in the left and right image being horizontally aligned, lined up on the green lines marking the epipolar lines.

From the rectified images, the correlation routine can be run on the image set, computing disparity values for each pixel. The disparity map for the rectified images in Figure 28 is shown below in Figure 29, where brighter intensities represent objects that are closer to the camera. From the disparity map, the depth of each pixel can be computed and re-projected into a 3-D point cloud representing the 3-D scene using the equations described in Chapter 2. The results from re-projection are shown in Section 4.4.
4.3 Terrain Mapping Results

The first stereo vision system described in Section 4.2 was used to take images of a test site that was built at the Unmanned Systems Lab to replicate actual terrain. The test site is shown below in Figure 30, which consists of several objects (a barrel and a cinder block) and areas with elevated and lowered terrain. The system was placed on a fork lift and was lifted 16 feet above the site. A 3-D point cloud is shown in Figure 31 and a wire mesh model in Figure 32 using Quick Terrain Modeler from Applied Imagery, with a color map overlaid based on distance of the points from the stereo system.

Figure 29: Disparity map of the scene from Figure 28 taken by 14” pitch stereo system.
Figure 30: Test site at Unmanned System Lab used to build 3-D model.

Figure 31: Un-gridded Point Cloud generated from 18” stereo vision system with height coloration overlay.

Figure 32: Wire mesh gridded surface model generated from 18” stereo vision system with height coloration overlay.
Using the 18” shown in Figure 22, images were taken of a modified test site as shown above. The model created by this system serves as a comparison of a lower resolution camera system and a higher resolution camera stereo system. The modified test site is shown in Figure 33, where a bridge was added and a pipe setting on two objects above the ground. From the point cloud generated and shown in Figure 34, immediately it can be noted that the model has substantially more points than the version created by the 18” system with lower resolution cameras. The number of points in the point cloud went from 255,640 to 1,444,064 when created from the stereo system with the higher resolution cameras. Looking at the point cloud shown in Figure 34 and the gridded surface in Figure 35, the objects appear much smoother and have a more defined shape than those generated by the lower resolution stereo system. The reason for this is that there are more points per area (higher density of points) in the point cloud generated by the high resolution system. The trade off, as explained in Section 2.5.1, is the amount of time taken to generate the model. With higher resolution cameras, the number of disparities searched has to be increased which adds to the search time in addition to the added number of pixels in the image.

![Figure 33: Modified test site at Unmanned Systems Lab.](image)
Figure 34: Un-gridded Point cloud generated using 18'' stereo system using high resolution cameras. Height of terrain is mapped using a color map.

Figure 35: Wire mesh gridded surface model of sample terrain site with color map corresponding to height overlaid.

The final system that was tested for mapping terrain is the 5’ baseline system shown in Figure 24. This stereo vision camera was mounted on a VTOL UAV and flown to an altitude of 60 meters, as shown in Figure 36. The images shown in Figure 37 are the left, right, and color images of the terrain 60 meters below the system. These images show a scene of a road and a group of trees at the middle of the image. The model generated from the set of images, with the color image overlaid on top of the model, is shown in Figure 38. From this model, the trees that are in the middle of the image can be seen modeled sticking out above the flat terrain below it.
Figure 36: 5’ stereo vision system mounted onto a Yamaha RMAX VTOL UAV.

Figure 37: Left, Right, and Color images taken from 5’ stereo vision system at 60 meters.

Figure 38: Gridded surface of model generated from left and right images taken with 5’ stereo system at 60 meters, shown with a color image texture overlay.
4.4 Object Detection and Distance Measurement Performance

In Section 4.3 results from terrain mapping were discussed with models shown of the point clouds and resulting gridded surfaces that were generated from a variety of stereo systems. One thing that was not covered was the accuracies of $x$, $y$, and $z$ distance measurements that were produced by re-projection. This section will describe another stereo system and its performance with object detection. This section will also cover the performance of the system at measuring objects and distances to the objects. An application was developed in MATLAB to extract objects in a point cloud that are within a certain distance of the camera. Also, distances and sizes of objects can be extracted by clicking on a point in the extracted components.

Using the 14.5” calibrated system shown in Figure 23, images were taken of a scene that was set up inside at the Unmanned Systems Lab. The objects that were set up in this scene were a blue tub set on a stool, a newspaper sitting on the floor, the end of storage shelf, and an orange cone. Two sets of images were taken at different distances and are shown in Figure 39 and Figure 40. Actual distances were measured from the stereo system to the objects for both sets of images and are listed in Table 5 below.

![Figure 39: Left and right view of objects in a scene (further view). Storage tub set on a stool (green), newspaper lying on the floor (orange), end of a wooden shelf (red), and an orange cone (blue).](image)

| Figure 39: Left and right view of objects in a scene (further view). Storage tub set on a stool (green), newspaper lying on the floor (orange), end of a wooden shelf (red), and an orange cone (blue). |
Figure 40: Left and Right view of objects in a scene (closer view).

Table 5: Measured distances from 14.5'' stereo system to objects in far and close to scene.

<table>
<thead>
<tr>
<th>Object</th>
<th>Measured Distance to Object</th>
<th>Far</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tub</td>
<td>3.074 m</td>
<td>2.337 m</td>
<td></td>
</tr>
<tr>
<td>Newspaper</td>
<td>3.734 m</td>
<td>2.921 m</td>
<td></td>
</tr>
<tr>
<td>Shelf</td>
<td>4.242 m</td>
<td>3.354 m</td>
<td></td>
</tr>
<tr>
<td>Cone</td>
<td>4.853 m</td>
<td>4.014 m</td>
<td></td>
</tr>
</tbody>
</table>

Both sets of images were processed using the Stereo Vision System Tools outlined in Chapter 3. From the generated point cloud, the distance from the object to the stereo system that was calculated can be extracted. The measured and calculated distances for both sets of images are shown in Table 6 with the percent error between the two.

Table 6: Measured and calculated distances from 14.5'' stereo system to objects in far and close to scene.

<table>
<thead>
<tr>
<th>Object</th>
<th>Scene viewed from far.</th>
<th>Scene viewed from close.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Measured</td>
<td>Calculated</td>
</tr>
<tr>
<td>Tub</td>
<td>3.07 m</td>
<td>3.19 m</td>
</tr>
<tr>
<td>Newspaper</td>
<td>3.73 m</td>
<td>3.76 m</td>
</tr>
<tr>
<td>Shelf</td>
<td>4.24 m</td>
<td>4.24 m</td>
</tr>
<tr>
<td>Cone</td>
<td>4.85 m</td>
<td>4.89 m</td>
</tr>
</tbody>
</table>
Using MATLAB a program was developed that, given a maximum distance, will extract all objects that are within that range of the camera and label them as individual components, the code is shown in Appendix B: Object extraction source code using Matlab. From these components, size and distance of the object can be extracted for each component. This application would be useful in an environment where the camera was mounted on a robot and could be used to determine if the robot was getting close to an object. With this, objects that are beyond the maximum distance can be discarded and objects within can be extracted and examined more carefully.

The object extraction algorithm sorts through the point cloud and gathers all the points that are within the maximum distance and compresses them to a single x,y plane. A graph theory technique called connected components analysis is used on the resulting image to find the components in the image. In a connected components search, pixels are considered ‘connected’ if they are either 4-connected neighbors or 8-connected neighbors of each other. In a 4-connected component search, connections are only made between pixels that are horizontally or vertically neighbors. An 8-connected component search looks for pixels that are diagonally neighbors in addition to horizontally or vertically. An example of 4-connected and 8-connected pixels is shown in Figure 41. The pixels labeled ‘8’ are 8 connected and the pixels labeled ‘4’ are 4 connected with the reference pixel in the middle.

A connected component is the group of all the pixels that are connected by either 4 connection or 8 connection. In this algorithm, the method used is 4 connection, which produces components that are more strongly connected than an 8 connection algorithm. Figure 42 shows the 4 connected components that are found within the point cloud generated from the set of
images taken far from the scene. Here only the objects that are within 4.5 meters of the camera are extracted, which includes all the objects but the cone. Colors are used to represent the different connected components that were found in the scene.

Figure 42: Components extracted from point cloud, far view (left). Corresponding components extracted in rectified image (right).

Using the same maximum distance, 4.5 meters; the same analysis is run on the set of images taken from close to the scene. These results are shown in Figure 43, where the cone now becomes visible as it is within the 4.5 meter range due to the system approaching the objects. In addition to the cone appearing, some objects in the background appear, where distances were not measured. It can be assumed that the calculated distance to these objects lie within the 4.5 meter range in the close view of the scene.

Figure 43: Components extracted from point cloud, close view (left). Corresponding Components extracted in rectified image (right).
Another feature of this application is the ability to extract information about the length and width of the objects in addition to the distance from the camera. By clicking on a point in the figure of the component view, the \( (X, Y, Z) \) information about that point is extracted. By clicking on the left and right side of the shelf and subtracting the \( X \) value, a calculation for width of the shelf can be found. Performing the same at the top and bottom of the shelf and subtracting the \( Y \) value will give a calculation for the height. In Table 7, the results of the measured and calculated size of the end of the shelf are shown with the percent error from the far view.

**Table 7: Measured and calculated size of shelf with percent error from scene viewed from far and up close.**

<table>
<thead>
<tr>
<th>Object</th>
<th>Measured</th>
<th>Calculated</th>
<th>Percent Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width (widest)</td>
<td>0.635 m</td>
<td>0.619 m</td>
<td>2.52 %</td>
</tr>
<tr>
<td>Width (narrowest)</td>
<td>0.419 m</td>
<td>0.404 m</td>
<td>3.57 %</td>
</tr>
<tr>
<td>Height</td>
<td>1.93 m</td>
<td>1.97 m</td>
<td>2.07 %</td>
</tr>
</tbody>
</table>
Chapter Five: Camera Pose Correction for Large Baseline Stereo System

5.1 Overview

In Chapter 2, the importance of calibrating a stereo vision system, so that the relation of the left and right cameras is known, was discussed. With a calibrated system such as this, there is a known equation relating the disparity of the scenes captured by the two imagers of the calibrated system to depth of objects to the cameras. Any motion of the cameras after calibration will result in error when row aligning the image planes during the rectification process. Most stereo vision systems are built with a relatively small baseline so as to reduce the amount of relative camera. Commercially available systems are rarely found with a baseline exceeding 0.6 m, which limits the range of the system. To calculate depth from a greater distance, such as from an aircraft, it is necessary to use a larger baseline system to obtain good results. The problem with larger systems is maintaining relative camera position, after calibration, since the accuracy of distance measurement is directly related to camera orientation/position accuracy.

In order to solve for the flexible camera problem, a method was implemented using a modal analysis to correct for relative camera motion. Using a calibrated stereo vision system such as in Figure 44, camera motion can be accurately described in the modal domain, depending on system configuration and the nature of disturbances to the system. Using analytical techniques, rotations of the cameras can be extracted from measured accelerations, measured in the frequency domain, at the cameras. With an accurate camera rotation calculated while the beam is bending, a rotation matrix with the correct angle between the two cameras can be applied during the rectification process to offset for the difference in angle from that of the calibrated system. This process will rotate the image planes according to the updated angle, so that they are at the same orientation as they would be from the calibrated state, providing accurate stereo images for correlation.

This paper considers situations in which two cameras are separated by a relatively large baseline distance, using a structural support that can flex when the system is subjected to vibration. The primary concern is rotation by the cameras in opposite directions about their respective y axes. Vibration will tend to cause the cameras to converge (rotate toward each other) and diverge (rotate away from each other) repeatedly. At maximum deflection during
convergence, the disparity values for corresponding points will be smaller than for the calibrated (stationary) system. Based on the analysis given above, the estimated range values would therefore be smaller than the true values. Conversely, at maximum deflection during divergence, the disparity values will be larger, and the estimated range values would be larger than the calibrated values. Also, maximum deflection corresponds to minimum camera motion, which is the best time to capture images in an effort to reduce the effects of motion blur.

Accelerometers can be attached to the cameras for estimating deflection angles, and therefore to correct for small rotational changes of the cameras. Assuming periodic motion, accelerometers can also be used to predict the instants of maximum deflection, which is when the camera should be triggered.

5.2 Instrumentation

Figure 44 shows the stereo system used in this experiment, it has two Watec black and white cameras with 3.8mm focal length lenses. The system was calibrated using the Stereo Vision Tools outlined in Chapter 3.

A rotation matrix can describe the orientation of the left camera with respect to the right, using the camera-centered coordinate system shown in Figure 45. The origin is located at the point of projection, and the $z$ axis coincides with the optical axis. The $x$ and $y$ axes are parallel to
the image plane. Rotation about the $x$, $y$, and $z$ axes will be represented as $\theta_x$ (pitch), $\theta_y$ (yaw), and $\theta_z$ (roll) of the camera, respectively.

![Camera-centered coordinate system](image)

Figure 45: Camera-centered coordinate system. This shows a Watec 660D G3.8 mono board camera, which is the model used for this project.

The composite rotation vector $RV$ for one camera can be expressed as the row vector

$$RV = [\theta_x, \theta_y, \theta_z]$$

If $RV$ represents a fixed axis through the origin, then the angle of rotation about this axis is given by the vector norm

$$\theta = ||RV||$$

and the corresponding unit vector is

$$\Omega = \frac{RV}{\theta}$$

with components $\Omega = [\Omega_x, \Omega_y, \Omega_z]$. Now, defining the antisymmetric matrix as

$$\Omega_v = \begin{bmatrix}
    0 & -\Omega_z & \Omega_y \\
    \Omega_z & 0 & -\Omega_x \\
    -\Omega_y & \Omega_x & 0
\end{bmatrix}$$

then the rotation matrix is given by

$$R = I + \Omega_v \sin \theta + \Omega_v \Omega_v^T (1 - \cos \theta)$$
where \( I \) is the 3×3 identity matrix. This is known as Rodrigues' rotation formula, and it can be rewritten as

\[
R = \begin{bmatrix}
c \theta + \Omega_x^2 (1 - c \theta) & -\Omega_x s \theta + \Omega_x \Omega_y (1 - c \theta) & \Omega_y s \theta + \Omega_x \Omega_z (1 - c \theta) \\
\Omega_x s \theta + \Omega_x \Omega_y (1 - c \theta) & c \theta + \Omega_y^2 (1 - c \theta) & -\Omega_x s \theta + \Omega_y \Omega_z (1 - c \theta) \\
-\Omega_y s \theta + \Omega_x \Omega_z (1 - c \theta) & \Omega_x s \theta + \Omega_y \Omega_z (1 - c \theta) & c \theta + \Omega_z^2 (1 - c \theta)
\end{bmatrix}
\]

Where ‘c’ is cosine and ‘s’ is sine. For both cameras, the combined rotation matrix is

\[
R = R_r R_l^T
\]

where \( R_r \) and \( R_l \) are the rotation matrices for the right and left cameras, respectively.

### 5.3 Experimental Setup

The stereo system was built using two Watec 660D cameras, as shown in Figure 46 and described in Table 8, networked with an Axis 241Q Video Server, shown in Figure 46 and described in Table 9. The cameras were mounted on a beam at a baseline of 10 inches, which was placed on a 50 lb. shaker. The cameras were connected to the video server through two coax video cables and powered by 9 volts through a bench top power supply.

![Stereo System with two Watec 660D cameras and Axis 241Q Video Server.](image)
Table 8: Camera Specifications for Left and Right Cameras. (Watec, 2008)

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>¼” interline transfer CCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Cell Size</td>
<td>7.15 µm x 5.55 µm</td>
</tr>
<tr>
<td>Resolution</td>
<td>704 x 480</td>
</tr>
<tr>
<td>Minimum Illumination</td>
<td>0.06lx.</td>
</tr>
<tr>
<td>Focal Length</td>
<td>3.8mm</td>
</tr>
</tbody>
</table>

Table 9: Encoder Specifications for stereo system. (Axis Communications, 2009)

<table>
<thead>
<tr>
<th>Connection Type</th>
<th>Network Cameras</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>Transmits simultaneous</td>
</tr>
<tr>
<td></td>
<td>streams up to 704 x 576</td>
</tr>
<tr>
<td>Video Frame Rate</td>
<td>Up to 30 frames per second</td>
</tr>
<tr>
<td>Number of Channels</td>
<td>4</td>
</tr>
<tr>
<td>Video Format</td>
<td>Motion JPEG and MPEG-4</td>
</tr>
</tbody>
</table>

The data acquisition system was a Spectral Dynamics model 20-24 digital signal processing (DSP) unit running the SigLab analysis software. Figure 47 shows the components that were used in the experiment.

Figure 47: Shaker (Left) is a Vibration System Model VG10054. Accelerometer (Middle) is a PCB 35C65 and the analyzer is a Spectral Dynamics 20-24 (Right).

The 3-Dimensional scene that was analyzed was a set of objects placed on the ceiling above the shaker, on which the stereo system was mounted. This scene is shown in Figure 48.
newspaper was used as a backdrop to provide a textured background for correlation purposes in the stereo matching routine.

![Figure 48: View of scene used for measuring distances with stereo system.](image)

### 5.4 Test Procedures

Using the SigLab virtual network analyzer function, a continuous sinusoidal input signal was applied to excite the stereo vision system at its first fundamental frequency of 20 Hz. Acceleration data was then sampled at a rate of 512 Hz in 20 second intervals. From the acceleration date, the beam displacement (deflection of the cameras) can be calculated using the following relationship:

\[ Y_{disp} = \frac{\ddot{y}}{\omega^2} \]
where $\ddot{Y}$ is the acceleration measured by the accelerometer and $\omega$ is the frequency in radians per second. Figure 49 shows a graph of acceleration vs. time plot and the derived displacement vs. time plot.

![Graph of Acceleration Data](image1)

![Graph of Accelerometer Displacement](image2)

**Figure 49:** Graph of Acceleration Data in m/s (Top) and graph of Accelerometer Displacement (Bottom).

From Figure 49, the point where the accelerometer (camera) is at its maximum acceleration corresponds to the maximum deflection point. The deflection measured from the accelerometer data is about 0.995 mm. Figure 50 shows a plot of the mode shape which is used to determine the displacement angle of the camera from 0. The accelerometer was placed 87 mm from the middle of the stereo vision system and the camera was placed at 97 mm from the end. Using the accelerometer data and the scaled mode shape, the displacement of the cameras at their maximum deflection point during vibration can be found.
To find the camera rotation angles, the 1\textsuperscript{st} mode beam slope, $\frac{\partial Y}{\partial x}$, can be found from the mode shape. Here, the beam slope corresponds to the slope of the stereo vision beam from bending during vibration. From the experimental data, the peak camera rotation was found to be 0.0265 radians, this is shown in Figure 51.
Before the experiment was conducted, a stereo pair was taken while the cameras were motionless in order to compare the corrected results to the ideal results from the system. The disparity map for the system in a state of rest is shown in Figure 52. This figure shows the relative distance of the objects from the camera. Table 10 shows the results of distance calculations before (top two rows) the shaker is turned on, and after (bottom two rows) the shaker is turned on where images are obtained at the peak acceleration (peak displacement and rotation). Note that distance errors of 19% represent the maximum error in the oscillatory cycle for an uncorrected set of images, but they also represent the highest quality images because the cameras are momentarily motionless at the peaks of the cycle. It is also of note that the cameras have moved 1 mm. further from the objects while at maximum deflection.

Table 10: Results from stereo matching: static (top) and dynamic (bottom). Measurements are distances to the real world points, found using a measuring tape from the object to the system.

<table>
<thead>
<tr>
<th>Pixel</th>
<th>Calculated</th>
<th>Measured</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>{263, 345} White</td>
<td>1.344 m</td>
<td>1.371 m</td>
<td>1.98%</td>
</tr>
<tr>
<td>{222, 367} Black</td>
<td>1.168 m</td>
<td>1.168 m</td>
<td>0.39%</td>
</tr>
<tr>
<td>{274, 350} White</td>
<td>1.109 m</td>
<td>1.371 m</td>
<td>19.08%</td>
</tr>
<tr>
<td>{220, 341} Black</td>
<td>0.954 m</td>
<td>1.168 m</td>
<td>18.31%</td>
</tr>
</tbody>
</table>

Figure 52: Disparity map from correlation without vibration (left) and with vibration (right).
Using the displacement angle of the camera calculated from the modal analysis of the stereo vision system, the corrected rotation matrix can be calculated using the Rodrigues rotation transform. The results of this application are shown in Section 5.4.

5.5 Results of Camera Motion Testing

In the experiment, the camera angle at peak motion was calculated to be 0.0261 rad., resulting in a \( \theta_y \) of 0.0522 rad. which is used to obtain the corrected camera pose. The disparity map in Figure 53 and the calculated error in Table 11 show the improvements in calculating depth using the angle calculated from the accelerometer data.

Table 11: Error in distance measurement using camera corrections.

<table>
<thead>
<tr>
<th>Pixel</th>
<th>Calculated</th>
<th>Measured</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>{265, 334} White</td>
<td>1.309 m</td>
<td>1.371 m</td>
<td>4.55 %</td>
</tr>
<tr>
<td>{230, 322} Black</td>
<td>1.093 m</td>
<td>1.168 m</td>
<td>6.37 %</td>
</tr>
</tbody>
</table>

Figure 53: Disparity map resulting from applying angle calculated for camera corrections.

Comparing the results of Table 11 to the results found without applying the calculated displacement angle in Table 10, a significant improvement in distance measurements is seen.
when the angle corrections are applied to the stereo rectification. The error in measured distance dropped 14.5% for the point corresponding to the black marker and 11.94% for the white marker, providing an average reduction in distance measurement error of 13.22%.

Figure 54: 3-D Model of scene (left) with height profile (right) before correct angle applied.

Figure 55: 3-D Model of scene (left) with height profile (right) after correct angle applied, the height profile is taken over the tall object in the scene, as indicated by the red line in the 3-D model.
In Figure 54 and Figure 55, the 3-D models produced from the point clouds generated from the stereo vision system designed for this experiment are shown, while in a vibrating state. The results in Figure 54 are from processing the stereo image pair before applying the angle calculated for the camera deflection from the modal analysis. In the height profile, the values on the left correspond to the distance from the stereo system. Examining the height profile, which was taken across the tallest object in the scene and corresponds to the object that the black marker is placed on in Figure 53, it can be seen that the distance measurement is off from the actual measurement. In Figure 55, which is the model corresponding to the point cloud generated from correlation performed after applying the corrected angle; the height profile shows the improved results of distance measurements.

5.6 Real-time Camera Trigger

From the results in the previous sections, applying a corrected angle improves the distance measurement when the camera angles are deflecting from the calibrated state. However, the data from the experiment conducted to produce these results was processed after the experiment. A method for camera trigger when the cameras are at maximum deflection needs to be implemented for a real-time system.

Using a least squares fit of a sample from the acceleration from the accelerometer input; an estimate of the acceleration can be made. Using this estimate for acceleration, the cameras can be triggered by the acceleration being read from the accelerometers and the equation approximating the input signal. The least squares approximation is found by using the following equation for acceleration:

\[ \ddot{Y} = c_1 \sin(\omega t) + c_2 \cos(\omega t) \]

Where \( t \) is the time of the samples, \( \omega \) is the frequency, and \( \ddot{Y} \) is a vector of acceleration values used to estimate the signal. To solve for the best estimate of the coefficients \( c_1 \) and \( c_2 \) from

\[ Y(t) = \begin{bmatrix} \sin(\omega t) & \cos(\omega t) \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \]

gives
\[ C = ([H]^T[H])^{-1}[H]^TY(t) \]

where
\[
\begin{bmatrix}
\sin(\omega t) & \cos(\omega t) \\
\vdots & \vdots \\
\sin(\omega T) & \cos(\omega T)
\end{bmatrix} = H
\]

And
\[
\begin{bmatrix}
c_1 \\
c_2
\end{bmatrix} = C
\]

Solving for \( C \) results in the coefficient vector that contains \( c_1 \) and \( c_2 \). Using the phase angle \( \phi \), an approximation to the acceleration signal can be found. The phase angle \( \phi \) is found from the following relation:

\[
\phi = tan^{-1}\left(\frac{c_1}{c_2}\right)
\]

Using this phase angle, the maximum deflection can be found when

\[ |\cos(\omega t + \phi)| = 1 \]

Using this approximation for the input acceleration signal, the camera can be triggered to take images and the measured angle can be applied to correct the images to obtain accurate distance measurements.

This method for finding the phase angle and determining the maximum deflection was implemented in MATLAB using the acceleration plot found in Section 5.3. The program first samples the first 1000 acceleration values for \( \ddot{Y} \). The sample rate \( \omega \) is the frequency of the system, 19 hz, or 119.38 radians per second. Solving for \( c_1 \) and \( c_2 \) provides the phase angle \( \phi = 0.07747 \) radians. Once the phase angle is found, the equation below can be used to trigger the camera at time \( t \) where,

\[ |\cos(119.38 \times t + 0.07747)| = 1 \]

The graph for the acceleration found in Section 5.3 and the corresponding least squares fit are shown below in Figure 56. The points where the camera needs to be triggered are shown in both graphs, and as shown in the initial experiment, the peak accelerations correspond to the
maximum deflection of the cameras. Pseudo code for an implementing this algorithm for real-
time camera correction is shown in Figure 57.

```plaintext
void main()
{
    // Find Least Squares Fit of acceleration plot
    readNVoltageSamplesFromDAQ();
    accelData = convertVoltSamplesToAccel();
    approxSignal = computeLeastSquaresFit(accelData);
    correctedAngle = findMaxDeflectionAngle();

    // Sample accel data at time t and trigger camera if approximate cosine signal equals to
    // 1 or -1
}
```

Figure 56: Plot of acceleration measured from accelerometers (Top) and least squares fit cosine wave (bottom).
while(Sampling)
{
  if(approxSignal(time) == 1)
  {
    grabImagesFromCamera();
    stereoProcessing(correctedAngle);
  }
}

Figure 57: Pseudo code for real-time camera correction using least square fit approximation.
Chapter Six: Conclusions and Future Work

In this thesis, the development of a number of stereo vision systems has been described. The results from 3-D re-projection for terrain mapping and object detection from these systems have been characterized, with the importance of accurate and real-time terrain data for a robot deployed from a VTOL UAV in mind. A tool for detecting objects within the vicinity of a stereo system has been shown with favorable results in measuring the size and distance of objects and openings between objects through which a ground robot could fit. Finally, a method for correcting errors in correlation from camera angle deflection has been described. The results from applying this correction have been shown to improve the percent in error from calculated distance to actual measured distance to objects.

From the work performed in this thesis, various issues have surfaced that still need to be addressed. An easier method for calibrating large baseline stereo systems needs to be developed. Currently, the only methods of doing this are using a large checkerboard or manually extracting points from a large object with feature points of known dimensions. A simpler method might be looked at of projecting a grid, where feature points can be extracted easily, onto the side of a building or floor and taking images for calibration. The problem that arises from calibrating a system to be used at 60 meters, is finding an object large enough to fill the majority of the frame. One avenue of research might be looking at calibrating at a smaller scale and characterizing the performance of the close range calibration to long range stereo correlation.

Future development for implementing the method proposed for extracting objects within a certain distance of the camera system into a point cloud viewer developed by past students of the Unmanned Systems Lab is a possible area for improvement. This application would be useful for a ground station controller who is interested in extracting just the objects that the system is in immediate danger of collision with. Integration of this application into the viewer was not accomplished by this thesis due to a lack in time to incorporate the algorithm. However, the methods used are feasible to port from MATLAB code to C++, the native language of the viewer. In addition, integrating a 3-D connected component search could be explored instead of the 2-D approach which suffers when edges separating the components are not well defined.
Dealing with vibration correction, the pseudo code for the implementation of the least squares analysis for camera trigger at peak phase angle should be implemented for real time analysis. Possible problems arising from this experimentation might be latency in the detection of the peak angle to the triggering of the camera. This issue might be solved by a predictive camera trigger, with knowledge of the delay between trigger and execution of image capturing.
WORKS CITED


Appendix A: Stereo Vision System Tools C++ source code

A. StereoMain.cpp

The StereoMain.cpp file defines the entry point for the stereo vision system tools. From the main, the stereo functions for rectification, correlation, and re-projection are called.

```cpp
#include "stdafx.h"
#pragma warning( disable: 4996 )
#include <vector>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <iostream>
#include <ctype.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"
```
#include "StereoGrabber.h"
#include "StereoFunctions.h"

using std::string;
using std::vector;
using namespace std;

typedef unsigned char GLubyte;

StereoGrabber* grab = new StereoGrabber();
StereoFunctions* stereoFunc = new StereoFunctions();

static struct SwsContext *img_convert_ctx1 = NULL;
static struct SwsContext *img_convert_ctx2 = NULL;

CV_EXTERN_C_FUNCPTR(void(*CvMouseCallback)(int event, int x, int y, int flags, void* param));
static void cvSetMouseCallback(const char* window_name, CvMouseCallback* on_mouse, void* param=NULL);

void mouseHandler(int event, int x, int y, int flags, void* param);
//handling mouse events
bool fexists(const char *filename);
void matchingAlgoTime();
//method for measuring correlation matching times

//functions for handling stereo correlation tools events
void onWindowBarSlide(int pos);
void onTextureBarSlide(int pos);
void onUniquenessBarSlide(int pos);
void onPreFilterSizeBarSlide(int pos);
void onPreFilterCapBarSlide(int pos);
void onSaveDataSlide(int pos);
void onSaveOriginalsSlide(int pos);
void stereoCreateCorrelationControls();
void loadCorrelationParams();
int connectToIEEE();

int main(int argc, char* argv[])
{
    //load images manually from files provided in command line arguments
    if(!GRAB_IMAGES)
    {
        stereoFunc->img1 = cvLoadImage(argv[2],0);
        stereoFunc->img2 = cvLoadImage(argv[3],0);

        //create variables for stereo processing
        stereoFunc->imageSize = cvSize(stereoFunc->img1->width,stereoFunc->img1->height);
        stereoFunc->img1r = cvCreateMat( stereoFunc->imageSize.height, stereoFunc->imageSize.width, CV_8U );
        stereoFunc->img2r = cvCreateMat( stereoFunc->imageSize.height, stereoFunc->imageSize.width, CV_8U );
        stereoFunc->disp = cvCreateMat( stereoFunc->imageSize.height, stereoFunc->imageSize.width, CV_16S );
        stereoFunc->vdisp = cvCreateMat( stereoFunc->imageSize.height, stereoFunc->imageSize.width, CV_8U );
        stereoFunc->depthImage = cvCreateMat( stereoFunc->imageSize.height, stereoFunc->imageSize.width, CV_8U );

        //for grabbing images and reading in for processing, if not, images have already been loaded
        else if(GRAB_IMAGES){
            if(CAMERA_TYPE == NET)
                int setupCam = grab->stereoGrabberOpenVideoStream();
            else if(CAMERA_TYPE==IEEE1394){
                //int i = connectToIEEE();
            }
        }
    }
}
else if(CAMERA_TYPE==IMAGE_GRABBER_EXEC)
{
    grab->imageGrabber = argv[2];
    grab->grabImagesFromCameras();
}
}
//Look for Calibration File if it's there skip the calibration procedure.  NOTE: I have not been able to run the calibration routine on my laptop with Vista
//If the calibration file is found, read it and jump into the undistortion, rectification and correlation process.
if( !fexists(argv[1]) )
{
    printf("Calibration File not found!\n");
    stereoFunc->stereoCalibrate("Calib_Images.txt", 8, 6, 1, argv[1]);
    cvWaitKey(0);
} else
{
    printf("Calibration File Found!\nReading Calibration File then Process images...\n");
    if( !fexists("CorrelationParams.txt"))
    {
        printf("Correlation Parameters File not found!\n");
    } else{
        loadCorrelationParams();
    }

    //create controls for stereo correlation if in analysis mode
    if(ANALYSIS_MODE)
        stereoCreateCorrelationControls();

    //load calibration file
    stereoFunc->stereoReadCalibFile(argv[1], grab);
}
return 0;
```c
void onWindowBarSlide(int pos)
{
    stereoFunc->stereoDispWindowSize = cvGetTrackbarPos("SAD Window Size", "Stereo Controls");
    if((stereoFunc->stereoDispWindowSize%2!=0 && stereoFunc->stereoDispWindowSize>=5)
        stereoFunc->stereoCorrelation();

    if(stereoFunc->stereoDispWindowSize%2==0 && stereoFunc->stereoDispWindowSize>=5)
    {
        stereoFunc->stereoDispWindowSize++;
        stereoFunc->stereoCorrelation();
    }
    if(stereoFunc->stereoDispWindowSize<5)
    {
        stereoFunc->stereoDispWindowSize = 5;
        stereoFunc->stereoCorrelation();
    }
}

void onTextureBarSlide(int pos)
{
    stereoFunc->stereoDispTextureThreshold = cvGetTrackbarPos("Texture Threshold", "Stereo Controls");
    if(stereoFunc->stereoDispTextureThreshold)
        stereoFunc->stereoCorrelation();
}

void onUniquenessBarSlide(int pos)
{
    stereoFunc->stereoDispUniquenessRatio = cvGetTrackbarPos("Uniq. Ratio", "Stereo Controls");
    if(stereoFunc->stereoDispUniquenessRatio>=0)
```
stereoFunc->stereoCorrelation();
}

void onPreFilterSizeBarSlide(int pos)
{
  stereoFunc->stereoPreFilterSize = cvGetTrackbarPos("Pre-Filter Size", "Stereo Controls");
  if((stereoFunc->stereoPreFilterSize%2!=0 && stereoFunc->stereoPreFilterSize>=5)
    stereoFunc->stereoCorrelation();
}

void onPreFilterCapBarSlide(int pos)
{
  stereoFunc->stereoPreFilterCap = cvGetTrackbarPos("Pre-Filter Cap", "Stereo Controls");
  if((stereoFunc->stereoPreFilterCap%2!=0 && stereoFunc->stereoPreFilterCap>=5)
    stereoFunc->stereoCorrelation();
}

void onNumDisparitiesSlide(int pos)
{
  stereoFunc->stereoNumDisparities = cvGetTrackbarPos("Num. Disparities", "Stereo Controls");
  while((stereoFunc->stereoNumDisparities%16!=0 || stereoFunc->stereoNumDisparities==0)
    stereoFunc->stereoNumDisparities++;

  stereoFunc->stereoCorrelation();
}

void onSaveDataSlide(int pos)
{  
  cvSaveImage("Disparities//DisparityMap.jpeg",stereoFunc->vdisp);

  stereoFunc->stereoSavePointCloud();
}
void onSaveOriginalsSlide(int pos)
{
    cvSaveImage("Left.jpeg", stereoFunc->img1);
    cvSaveImage("Right.jpeg", stereoFunc->img2);
}

void stereoCreateCorrelationControls()
{
    cvNamedWindow("Stereo Controls", 0);
    cvCreateTrackbar("SAD Window Size", "Stereo Controls", &stereoFunc->stereoDispWindowSize, 255, onWindowBarSlide);
    //cvCreateTrackbar("Texture Threshold", "Stereo Controls", &stereoFunc->stereoDispTextureThreshold, 25, onTextureBarSlide);
    cvCreateTrackbar("Uniq. Ratio", "Stereo Controls", &stereoFunc->stereoDispUniquenessRatio, 25, onUniquenessBarSlide);
    //cvCreateTrackbar("Pre-Filter Size", "Stereo Controls", &stereoFunc->stereoPreFilterSize, 101, onPreFilterSizeBarSlide);
    //cvCreateTrackbar("Pre-Filter Cap", "Stereo Controls", &stereoFunc->stereoPreFilterCap, 63, onPreFilterCapBarSlide);
    cvCreateTrackbar("Num. Disparities", "Stereo Controls", &stereoFunc->stereoNumDisparities, 640, onNumDisparitiesSlide);
    cvCreateTrackbar("Save Point Cloud and Images", "Stereo Controls", &stereoFunc->stereoSavePointCloudValue, 1, onSaveDataSlide);
    //cvCreateTrackbar("Save Original Images", "Stereo Controls", &stereoFunc->stereoSaveOriginal, 1, onSaveOriginalsSlide);
}

void loadCorrelationParams()
{
    ifstream params;
    params.open("CorrelationParams.txt");
if (!params) {
    cerr << "Unable to open file Correlation Params file!";
    exit(1);  // call system to stop
} else{
    string temp;
    int value;
    while (params>>temp)
    {
        params>>value;
        stereoFunc->stereoDispWindowSize = value;
        params>>temp;
        params>>value;
        stereoFunc->stereoDispTextureThreshold = value;
        params>>temp;
        params>>value;
        stereoFunc->stereoDispUniquenessRatio = value;
        params>>temp;
        params>>value;
        stereoFunc->stereoNumDisparities = value;
    }
    stereoFunc->stereoPreFilterSize = 41;

    stereoFunc->stereoPreFilterCap = 31;
    stereoFunc->stereoSavePointCloudValue = 0;
    stereoFunc->stereoSaveOriginal = 0;

    params.close();
}

//Routine to calculate the CPU time needed to perform correlation on a set of images.
void matchingAlgoTime()
IplImage* img1 = cvLoadImage("SampleImages\left.png", 0);
IplImage* img2 = cvLoadImage("SampleImages\right.png", 0);

CvMat* disp = cvCreateMat( 512, 512, CV_16S );

CvStereoBMState *BMState = cvCreateStereoBMState();
assert(BMState != 0);
BMState->preFilterSize=41;
BMState->preFilterCap=31;
BMState->SADWindowSize=41;
BMState->minDisparity=-64;
BMState->numberOfDisparities=128;
BMState->textureThreshold=15;
BMState->uniquenessRatio=10;

long time_tot =0;
for(int i=0;i<200;i++)
{
  clock_t start = clock(), diff;
  cvFindStereoCorrespondenceBM( img1, img2, disp, BMState);
  diff = clock() - start;
  int msec = diff * 1000 / CLOCKS_PER_SEC;
  time_tot += (msec%1000);
}

printf("Time taken %d milliseconds",(double)(time_tot/200));
void mouseHandler(int event, int x, int y, int flags, void *param) {
    stringstream originalLeft, originalRight;
    string leftFile, rightFile;
    originalLeft<<"Originals/leftImage_"<<stereoFunc->fileNO<<".jpeg";
    originalRight<<"Originals/rightImage_"<<stereoFunc->fileNO<<".jpeg";
    leftFile = originalLeft.str();
    rightFile = originalRight.str();
    stereoFunc->fileNO++;
    switch(event) {
    case CV_EVENT_LBUTTONDOWN:
        // l = cvGet2D(depthImage, x, y);
        // printf("Disparity at x: %d y: %d Distance: %f \n", x, y, l.val[0]);
        cvSaveImage(&leftFile[0], grab->imageLeft);
        cvSaveImage(&rightFile[0], grab->imageRight);
        break;
    }
}

bool fexists(const char *filename) {
    ifstream ifile(filename);
    return ifile;
}
B. stdafx.h

The stdafx.h file contains the values which define the work flow of the stereo vision system tools.

```cpp
#pragma once

#include "targetver.h"
#include <string>
#include <stdio.h>
#include <tchar.h>

#define CV_EVENT_LBUTTONDOWN 1
#define CV_EVENT_RBUTTONDOWN 2
#define FIND_CORNERS 1    //1 Find corners during camera calibration
                         or 0 load points in manually from a text file

#define SHOW_SAVE_DATA 1    //show and save intermediate results from stereo
correlation, set this to 0 when just producing a point cloud

#define POINT_CLOUD_FMT_DYLAN 0    //output point cloud in format to read into
dylan's or dan's point cloud viewer

#define ANALYSIS_MODE 1    //set to 1 to enter analysis mode, where
                         intermediate results are shown and a correlation variable tool can be used
                         //extra data can still be saved, but is
done manually through a button on the tool
```
#define GRAB_IMAGES 0  //set to 1 to grab images from a camera automatically to be processed using an image grabber tool designed for the cameras
//a path to the grabber executable must be supplied in the command prompt at run time

#define SAVE_DISTANCES 0 //saves depth information of each pixel in a separate file

#define WRITE_BINARY 0 //write the point cloud data to a binary file, a binary converter must be used to convert back to ascii for integration with applied imagery

#define CAMERA_TYPE 2 //Determines the method for grabbing images from camera
#define NET 0 //uses network cameras with ffmpeg (right now only works in linux)
#define IEEE1394 1 //connect to fire wire cameras (this is not implemented yet) could be accomplished by building executable for fire wire cameras and loading images
#define IMAGE_GRABBER_EXEC 2 //that are saved

#define CS_NET 2 //executes an image grabber for cameras connected to video encoder

#define METERS2FEET 3.2808399 //conversion from meters to feet

#define PI 3.141592654 // TODO: reference additional headers your program requires here
C. StereoGrabber.h

StereoGrabber.h is the header file which contains the definition for variables and functions used to obtain images for processing.

```cpp
#pragma once

#include <iostream>
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"

/*
extern "C"
#include "ffmpeg/libavcodec/avcodec.h"
#include <ffmpeg/avformat.h>
#include <libswscale/swscale.h>
*/

using std::string;
using namespace std;

struct StereoGrabber{
    int stereoGrabberOpenVideoStream(); //open video stream from network cameras using FFmpeg
    void stereoGrabberGrabFrames();    //grab a frame from the stream
    void stereoGrabberInitFrames();
    void stereoGrabberConvertFrames();  //convert frames from FFmpeg format to OPENCV format
    void connectToIEEE1394();          //connect to fire wire cameras
};
```
void grabImagesFromCameras(); //run executable to grab images from network cameras
    bool BMPSave24( CString FName, BYTE *Buffer, DWORD BufferSize, int Width, int Height, int BitCount );
    IplImage* imageLeft;
    IplImage* imageRight;
    string imageGrabber;
};
D. StereoGrabber.cpp

StereoGrabber.cpp contains the algorithms for obtaining images for processing from various cameras, including networked cameras and fire wire cameras.

```cpp
#include "StereoGrabber.h"

//Grabs images from video encoder using Axis libraries
void StereoGrabber::grabImagesFromCameras()
{
    system(imageGrabber.c_str());
    imageLeft=cvLoadImage("Image_Left_Raw.jpeg");
    imageRight=cvLoadImage("Image_Right_Raw.jpeg");
}
```

AVFormatContext *pFormatCtx1, *pFormatCtx2;
AVCodecContext *pCodecCtx1,*pCodecCtx2;
AVCodec *pCodec1, *pCodec2;
AVFrame *pFrame1,*pFrame2, *pFrameRGB1,*pFrameRGB2;
AVPacket packet1,packet2;
int frameFinished1,frameFinished2, numBytes, i, videoStream1, videoStream2;
uint8_t *buffer1,*buffer2;
int m=0, n=0;
char *ptr1, *ptr2, *ptr3, *ptr4;

//This method only works with FFMPEG library
int StereoGrabber::stereoGrabberOpenVideoStream()
{
    fileNO =0;

    av_register_all();
    ```
int vid1=av_open_input_file(&pFormatCtx1, "http://192.168.1.3/mjpg/1/video.mjpg", NULL, 0, NULL),
vid2=av_open_input_file(&pFormatCtx2, "http://192.168.1.3/mjpg/2/video.mjpg", NULL, 0, NULL);

// Open video file Left and file right
if(vid1!=0 || vid2!=0)
    return -1; // Couldn't open file

if(av_find_stream_info(pFormatCtx1)<0 || av_find_stream_info(pFormatCtx2)<0)
    return -1; // Couldn't find stream information

// Dump information about file onto standard error
// dump_format(pFormatCtx1, 0, "http://192.168.1.3/mjpg/video.mjpg", 0);
// dump_format(pFormatCtx2, 0, "http://192.168.1.3/mjpg/2/video.mjpg", 0);

// Find the first video stream
videoStream1=-1, videoStream2=-1;
for(i=0; i<pFormatCtx1->nb_streams; i++)
{
    if(pFormatCtx1->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO)
    {
        videoStream1=i;
        break;
    }
}

for(i=0; i<pFormatCtx2->nb_streams; i++)
{
    if(pFormatCtx1->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO) {
        videoStream2=i;
        break;
    }
if(videoStream1==-1||videoStream2==-1)
    return -1; // Didn't find a video stream

// Get a pointer to the codec context for the video stream
pCodecCtx1=pFormatCtx1->streams[videoStream1]->codec;
pCodecCtx2=pFormatCtx2->streams[videoStream2]->codec;

// Find the decoder for the video stream
pCodec1=avcodec_find_decoder(pCodecCtx1->codec_id);
pCodec2=avcodec_find_decoder(pCodecCtx2->codec_id);

if(pCodec1==NULL||pCodec2==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
}

// Open codec
int openCodec1 = avcodec_open(pCodecCtx1, pCodec1), openCodec2 =
    avcodec_open(pCodecCtx2, pCodec2);

if(openCodec1<0 || openCodec2<0)
    return -1; // Could not open codec

// Allocate video frame
pFrame1=avcodec_alloc_frame();
pFrame2=avcodec_alloc_frame();

// Allocate an AVFrame structure
pFrameRGB1=avcodec_alloc_frame();
pFrameRGB2=avcodec_alloc_frame();
if(pFrameRGB1==NULL||pFrameRGB2==NULL) /*
    return -1;
}

void StereoGrabber::stereoGrabberInitFrames()
{
    // Determine required buffer size and allocate buffer
    numBytes=avpicture_get_size(PIX_FMT_RGB24, pCodecCtx1->width, pCodecCtx1->height);
    buffer1=(uint8_t *)av_malloc(numBytes*sizeof(uint8_t));
    buffer2=(uint8_t *)av_malloc(numBytes*sizeof(uint8_t));

    // Assign appropriate parts of buffer to image planes in pFrameRGB
    // Note that pFrameRGB is an AVFrame, but AVFrame is a superset
    // of AVPicture
    avpicture_fill((AVPicture *)pFrameRGB2, buffer2, PIX_FMT_BGR24, pCodecCtx2->width, pCodecCtx2->height);
    avpicture_fill((AVPicture *)pFrameRGB1, buffer1, PIX_FMT_BGR24, pCodecCtx1->width, pCodecCtx1->height);

    pFormatCtx2->index_built = 0;

    imageLeft = cvCreateImage(cvSize(pCodecCtx1->width, pCodecCtx1->height), IPL_DEPTH_8U, 3);
    imageRight = cvCreateImage(cvSize(pCodecCtx2->width, pCodecCtx2->height), IPL_DEPTH_8U, 3);
}

void StereoGrabber::stereoGrabberGrabFrames()
{
    int cam2=av_read_frame(pFormatCtx2, &packet2), cam1=av_read_frame(pFormatCtx1, &packet1);
    if(cam1 >= 0 && cam2 >= 0)
// Is this a packet from the video stream?
if(packet1.stream_index == videoStream1 && packet2.stream_index == videoStream2)
{
    // Decode video frame
    avcodec_decode_video(pCodecCtx1, pFrame1, &frameFinished1, packet1.data, packet1.size);
    avcodec_decode_video(pCodecCtx2, pFrame2, &frameFinished2, packet2.data, packet2.size);

    // Did we get a video frame?
    if(frameFinished1 && frameFinished2)
    {
        StereoGrabber::convertFrames();
    }
}
// Free the packet that was allocated by av_read_frame
av_free_packet(&packet1);
av_free_packet(&packet2);
}

void StereoGrabber::stereoGrabberConvertFrames()
{
    img_convert_ctx1 = sws_getContext(pCodecCtx1->width, pCodecCtx1->height, pCodecCtx1->pix_fmt, pCodecCtx1->width, pCodecCtx1->height, PIX_FMT_BGR24, SWS_BICUBIC, NULL, NULL, NULL);
    img_convert_ctx2 = sws_getContext(pCodecCtx2->width, pCodecCtx2->height, pCodecCtx2->pix_fmt, pCodecCtx2->width, pCodecCtx2->height, PIX_FMT_BGR24, SWS_BICUBIC, NULL, NULL, NULL);

    sws_scale(img_convert_ctx1, pFrame1->data, pFrame1->linesize, 0, pCodecCtx1->height, pFrameRGB1->data, pFrameRGB1->linesize);
sws_scale(img_convert_ctx2, pFrame2->data, pFrame2->linesize, 0, pCodecCtx2->height, pFrameRGB2->data, pFrameRGB2->linesize);

//store the frame in an IplImage

for(m = 0; m < pCodecCtx1->height; m++)
{
    ptr1 = (char*)(pFrameRGB1->data[0] + m*pFrameRGB1->linesize[0]);
    ptr2 = imageLeft->imageData + m*imageLeft->widthStep;
    ptr3 = (char*)(pFrameRGB2->data[0] + m*pFrameRGB2->linesize[0]);
    ptr4 = imageRight->imageData + m*imageRight->widthStep;

    for(n = 0; n < 3*pCodecCtx1->width; n++)
    {
        ptr2[n] = ptr1[n];
        ptr4[n] = ptr3[n];
    }
}

//connects to left, right, and color fire wire cameras and saves images.
void StereoGrabber::connectToIEEE1394()
{
    HCAMERA rightCamera, leftCamera, colorCamera, hCamera;
    UINT64 UID = -1, rightUID = 576537786841918118, leftUID = 576537786841917361,
    colorUID = 576537786845369806;
    char *ptr1, *ptr2, *ptr3, *ptr4;
    CString filename("right.bmp");
    ZCL_LIST *pList;
    ULONG Count;
    BOOL Exist;

    Count = 0;
if(!ZCLOpen( UID, &hCamera))
{
    // Failed to open the camera
} else{
    BYTE *pData[ 3 ];
    ZCL_GETIMAGEINFO Info, rightInfo;
    int idx;
    ZCLIsoAlloc( hCamera );
    ZCLGetImageInfo( hCamera, &rightInfo );
    for( idx = 0; idx < 3; idx++ )
    {
        pData[ idx ] = ( BYTE *)malloc( Info.Image.Buffer );
        ZCLImageReq( hCamera, pData[ idx ], Info.Image.Buffer );
    }
    ZCLIsoStart( hCamera, 0 );
    for( idx = 0; idx < 3; idx++ )
    {
        ZCLImageCompleteWait( hCamera, pData[ idx ], 0, 0, 0 );
    }
    ZCLIsoStop( hCamera );

    bool result = BMPSave24( filename,
                pData[0],
                rightInfo.Image.Width * rightInfo.Image.Height*3,
                rightInfo.Image.Width,
                rightInfo.Image.Height,
                16 );
}

for(int m = 0; m < rightInfo.Image.Height; m++)
{
    ptr1 = (char*)(pData[0]+ m*4800);
}
ptr2 = imageRight->imageData + m*imageRight->widthStep;
//ptr3 = (char*)(pFrameRGB2->data[0]+ m*pFrameRGB2->linesize[0]);
//ptr4 = imageRight->imageData + m*imageRight->widthStep;

for(int n = 0; n < 3*rightInfo.Image.Width; n++)
{
    ptr2[n] = ptr1[n];
    //ptr4[n] =  ptr3[n];
}

for (idx = 0; idx < 3; idx++)
{
    free( pData[ idx ] );
}

cvSaveImage("LeftImage.jpeg",imageLeft);

} //save images in BMP format

bool StereoGrabber::BMPSave24(CString FName, BYTE *Buffer, DWORD BufferSize, int Width, int Height, int BitCount )
{
    HANDLE hFile;
    BITMAPFILEHEADER FileHeader;
    BITMAPINFOHEADER InfoHeader;
    DWORD WSize, WriteSize;
    DWORD Error;
    int ByteCount;

    ZeroMemory( &FileHeader, sizeof( BITMAPFILEHEADER ) );
FileHeader.bfType = 'MB';
FileHeader.bfOffBits = sizeof( BITMAPFILEHEADER ) + sizeof( BITMAPINFOHEADER );

ZeroMemory( &InfoHeader, sizeof( BITMAPINFOHEADER) );
InfoHeader.biSize = sizeof( BITMAPINFOHEADER);
InfoHeader.biWidth = Width;
InfoHeader.biHeight = Height;
InfoHeader.biPlanes = 1;

ByteCount = BitCount / 8;

//32bit color
BOOL DIB32Flg = FALSE;
if( ByteCount == 4 )
{
    InfoHeader.biBitCount = 24;
    BitCount = 24;
    ByteCount = 3;
    DIB32Flg = TRUE;
}

FileHeader.bfSize = sizeof( BITMAPFILEHEADER ) + sizeof( BITMAPINFOHEADER ) +
Width * ByteCount * Height;
InfoHeader.biBitCount = BitCount;

hFile = CreateFile( (LPCWSTR)FName, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL , NULL );
if( !hFile )
    return FALSE;

WriteSize = 0;
// BMP File Header Write
WriteFile( hFile, &FileHeader, sizeof( BITMAPFILEHEADER ), &WSize, NULL );
WriteSize+=WSize;
WSize = 0;

// BMP Bitmap Header Write
WriteFile( hFile, &InfoHeader, sizeof( BITMAPINFOHEADER ), &WSize, NULL );
WriteSize+=WSize;
WSize = 0;

// BMP Bitmap Data Write
    if !( WriteFile( hFile, Buffer, BufferSize, &WSize, NULL ) )
    {
        Error = GetLastError();
    }
    WriteSize += WSize;

CloseHandle( hFile );

if( WriteSize != sizeof( BITMAPFILEHEADER ) + sizeof( BITMAPINFOHEADER ) + BufferSize )
{
    DeleteFile((LPCWSTR)FName);
    return FALSE;
}
return TRUE;
E. StereoFunctions.h

StereoFunctions.h contains the definitions for variables and functions used to rectify, correlate, and re-project stereo images.

```cpp
#pragma once
#include "stdafx.h"
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <fstream>
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"
#include "StereoGrabber.h"
#include <time.h>
using namespace std;

struct StereoFunctions{
    void stereoSavePointCloud();
    void stereoCorrelation();
    void stereoReadCalibFile(char *, StereoGrabber*);
    void stereoRectify(CvMat* _M1, CvMat* _M2, CvMat* _D1, CvMat* _D2, CvMat* _R, CvMat* _T, CvMat* CamData, StereoGrabber* stereoGrabber);
    void stereoCalibrate(const char* imageList, int nx, int ny, int, char *calibFileName);
    CvMat* depthImage; //3-D point image
    CvSize imageSize; //size of input images
    CvMat* img1r; //rectified left image
```
CvMat* img2r;  //rectified right image
CvMat* disp;  //disparity map
CvMat* vdisp;  //scaled disparity map for viewing
CvMat _Q;  //reprojection matrix

double reprojectionVars[7];

int stereoDispWindowSize,  //41
    stereoDispTextureThreshold,  //15
    stereoDispUniquenessRatio,  //10
    stereoPreFilterSize,
    stereoPreFilterCap,
    stereoSavePointCloudValue,
    stereoSaveOriginal,
    stereoNumDisparities;

IplImage *img1,  //left image
    *img2,  //right image
    *rgb;

int fileNO;
};
StereoFunctions.cpp

StereoFunctions.cpp contains all the algorithms for rectifying, correlating, and re-projecting stereo images to 3-D point clouds.

```cpp
#include "StereoFunctions.h"
#include "math.h"

int fileNO = 0;

//variables for correlation routine, some of them can be changed in analysis mode with
the correlation tools window
int stereoDispWindowSize, //41
    stereoDispTextureThreshold, //15
    stereoDispUniquenessRatio, //10
    stereoPreFilterSize,
    stereoPreFilterCap,
    stereoSavePointCloudValue,
    stereoSaveOriginal,
    stereoNumDisparities;

//Correlates pixels from the left and right images, results in a disparity map which is
stored in the variable 'disp'
void StereoFunctions::stereoCorrelation()
{
    //Setup for finding stereo correspondences
    CvStereoBMState *BMState = cvCreateStereoBMState();
    assert(BMState != 0);
    BMState->preFilterSize= stereoPreFilterSize;
    BMState->preFilterCap= stereoPreFilterCap;
    BMState->SADWindowSize= stereoDispWindowSize;
    BMState->minDisparity=0;
```
BMState->numberOfDisparities = stereoNumDisparities;
BMState->textureThreshold = stereoDispTextureThreshold;
BMState->uniquenessRatio = stereoDispUniquenessRatio;

// timing for stereo correspondence can be assessed by turning on the timer
double time_tot = 0;
clock_t start, diff;
double difference;
for (int i = 0; i < 100; i++)
{
    start = clock(); /*
    cvFindStereoCorrespondenceBM(img1r, img2r, disp, BMState); // performs stereo correspondence on the rectified left and right images
    */
    diff = clock() - start;
difference = (double)diff/CLOCKS_PER_SEC;
time_tot += difference;
}
printf("Time taken %f seconds", (double)((double)time_tot/100)); /*

cvNormalize( disp, vdisp, 0, 256, CV_MINMAX );

// save rectified images and disparity map if SHOW_SAVE_DATA is set to 1
if (SHOW_SAVE_DATA){
cvShowImage("Disparity Map", vdisp);
    string dispFile;
    stringstream str, str2, str3;
    string left, right;
    str<<"Disparities/DisparityMap"<<"-"<<fileNO<<".jpeg";
    dispFile=str.str();
    cvSaveImage(&dispFile[0], vdisp);
    str2<<"Rectified/Left"<<"-"<<fileNO<<".jpeg";
}
str3<<"Rectified/Right"<<"-"<<fileNO<<".jpeg";
left=str2.str();
right=str3.str();
cvSaveImage(&left[0],img1r);
cvSaveImage(&right[0],img2r);
}
cvReleaseStereoBMState(&BMState);
fileNO++;

//Saves point clouds to Dylan's and Dan's viewer format as well as ascii format for
//applied imagery integration

//void StereoFunctions::stereoSavePointCloud()
{
    //used to store reprojected 3-D image using open cv reprojection routine
    CvMat* threeDimage = cvCreateMat( imageSize.height, imageSize.width, CV_32FC3);
    FILE * threeFile,* distanceFile;

    //number of points in the point cloud, needed for dylans viewer
    long pointCount=0;
    double depth=0, horiz=0, vert=0;

    //vectors to store 3-D points calculated from reprojection
    CvMat *depthM = cvCreateMat(imageSize.height, imageSize.width, CV_32F),
          *horizM = cvCreateMat(imageSize.height, imageSize.width, CV_32F),
          *vertM = cvCreateMat(imageSize.height, imageSize.width, CV_32F);

    //create strings for point cloud files
    stringstream strThreeD;
    stringstream strDistance;
    if(POINT_CLOUD_FMT_DYLAN)
strThreeD<<"Image-"<<fileNO<<".ppm";
else
strThreeD<<"PointClouds/Image-"<<fileNO<<".pts";

string threeDFile = strThreeD.str();
threeFile = fopen(&threeDFile[0],"w");

//if SAVE_DISTANCES is set to 1, a separate file will be created to store distance measurements
if(SAVE_DISTANCES)
{
strDistance<<"Distance-"<<fileNO<<".txt";
string distancesFile = strDistance.str();
distanceFile = fopen(&distancesFile[0],"w");
}

//REPROJECTION VARIABLES NEED TO BE CHANGED FOR EACH SYSTEM
double cx= (-1)*reprojectionVars[0];
double cy= (-1)*reprojectionVars[1];
double baseline = reprojectionVars[2]; //1536.47571;
double focal = reprojectionVars[3];
double sensorSize = reprojectionVars[4];
double xfov = reprojectionVars[5]; //60.79745543 //field of view x
double yfov = reprojectionVars[6]; //47.498; //field of view y

//calculates reprojection of points in meters
for(int y=0;y<vdisp->rows;y++){
    for(int x=0;x<vdisp->cols;x++){
        if(cvGet2D(vdisp,y,x).val[0]>0){
            pointCount++;
            depth=(double)(((baseline*focal)/(double)(cvGet2D(disp,y,x).val[0]/16)*sensorSize));
        }
    }
    horiz=
\begin{verbatim}
(double)((2*sin((xfov/2)*(PI/180))*depth)/imageSize.width)*(double)(-1)*(x+cx);
    vert =
(double)((2*sin((yfov/2)*(PI/180))*depth)/imageSize.height)*(double)(y+cy);
else
    depth=0;
    cvmSet(depthM,y,x,depth);
    cvmSet(horizM,y,x,horiz);
    cvmSet(vertM,y,x,vert);
}

//disparity is found with 16-bit depth resolution, so all values need to be rescaled
to 1/16
IplImage* real_disparity= cvCreateImage( imageSize, IPL_DEPTH_32F, 1 );
cvConvertScale( disp, real_disparity, 1.0/16, 0 );

//opencv routine for reprojecting to 3-D, results found that the first method
performed better
cvReprojectImageTo3D(real_disparity,threeDimage, &_Q);

//output point cloud data to file (Dan's Format)
if(POINT_CLOUD_FMT_DYLAN)
{
    fprintf(threeFile, "P6\n");
    fprintf(threeFile, "#time= 12345\n" );
    fprintf(threeFile, "#pointcount= %d\n",pointCount);
}

ofstream myFile ("Points.bin", ios::out | ios::binary);

vector<CvPoint3D32f> PointArray;
CvPoint3D32f Point;
int i=0,j=0;
int intensity;
\end{verbatim}
for (int y = 0; y < threeDimage->rows; y++){
    float *data = (float *)(threeDimage->data.ptr + y * threeDimage->step);
    for (int x = 0; x < threeDimage->cols * 3; x = x + 3){
        //leave out points where the disparity was found to be zero (no correlation)
        if(cvGet2D(vdisp,i,j).val[0]>0){
            Point.x = data[x];
            Point.y = data[x+1];
            Point.z = data[x+2];
            PointArray.push_back(Point);

            intensity = (int)cvGet2D(img1r,i,j).val[0];

            if((POINT_CLOUD_FMT_DYLAN)\n            {\n                //output points projected from opencv routine\n                //fprintf(threeFile, "# %f %f %f %d %d %d %d %d %f\n", (double)(-1*Point.x/1000), (double)(-1*Point.y/1000), (double)(-1*cvmGet(depthM,i,j)/1000), (int)cvGet2D(img1r,i,j).val[0], (int)cvGet2D(img1r,i,j).val[0], (int)cvGet2D(rgb,i,j).val[0], (int)i, (int)j, (float)cvGet2D(real_disparity,i,j).val[0]);\n
                fprintf(threeFile, "# %f %f %f %d %d %d %d %d %f\n", (double)(-1*Point.x/1000), (double)(-1*Point.y/1000), (double)(-1*cvmGet(depthM,i,j)/1000), (int)cvGet2D(img1r,i,j).val[0], (int)cvGet2D(img1r,i,j).val[0], (int)cvGet2D(rgb,i,j).val[0], (int)i, (int)j, (float)cvGet2D(real_disparity,i,j).val[0]);\n
                if(SAVE_DISTANCES)\n                    fprintf(distanceFile, "%d %d %f\n", (int) i, (int) j, cvGet2D(depthM,i,j).val[0]);\n            }\n        }
    }
}
if (WRITE_BINARY)
{
    myfile.write ((char*)&point.x, sizeof (float));
    myfile.write ((char*)&point.y, sizeof (float));
    myfile.write ((char*)&point.z, sizeof (float));
    myfile.write ((char*)&intensity, sizeof (int));

    if (SAVE_DISTANCES)
        fprintf(distanceFile, "%d %d %f\n", (int) i, (int) j, cvGet2D(depthM,i,j).val[0]);
} else
{
    //output points projected from opencv routine
    //fprintf(threeFile, "%f %f %f %d %d %d %d %d %f\n", (double)(-1*point.x/1000), (double)(-1*point.y/1000), (double)(-1*cvmGet(depthM,i,j)/1000), (int)cvGet2D(img1r,i,j).val[0], (int)i, (int)j, (float)cvmGet2D(real_disparity,i,j).val[0]);

    //output points projected from first routine
    fprintf(threeFile, "%f %f %f %d %d %d %f\n", (float)cvmGet(horizM,i,j)/1000, (float)cvmGet(vertM,i,j)/1000, (float)cvmGet(depthM,i,j)/1000, intensity, (int)i, (int)j, (float)cvGet2D(real_disparity,i,j).val[0]);
    if (SAVE_DISTANCES)
        fprintf(distanceFile, "%d %d %f\n", (int) i, (int) j, cvGet2D(depthM,i,j).val[0]);
}
        j++;
    }
PointArray.clear();
i++;
j=0;
myFile.close();

if (POINT_CLOUD_FMT_DYLAN)
{
    fprintf(threeFile, "%d %d\n", imageSize.width, imageSize.height);
    fprintf(threeFile, "255");
}

fclose(threeFile);

if (SAVE_DISTANCES)
    fclose(distanceFile);

// rectify the left and right images and undisort then call correlation function
void StereoFunctions::stereoRectify(CvMat* _M1, CvMat* _M2, CvMat* _D1, CvMat* _D2,
    CvMat* _R, CvMat* _T, CvMat* _CamData, StereoGrabber* grab){

    int j, nframes=1, showUndistorted = 1, N=0, saveDisparityFiles=1, medianFilterWindow = 0;

    int useSampleImages = 1, findThreeDPoints=1, delay=1;
    fileNO=0;

    // 0: cx, 1: cy, 2: B, 3: f, 4: d, 5: xfov, 6: yfov
    reprojectionVars[0] = cvmGet(_M1,0,2);
    reprojectionVars[1] = cvmGet(_M1,1,2);
    reprojectionVars[2] = (-1)*cvmGet(_T,0,0);

    // READ In FOCAL LENGTH, SENSOR ELEMENT SIZE, XFOV, YFOV
    reprojectionVars[3] = cvmGet(_CamData, 0, 0);
    reprojectionVars[4] = cvmGet(_CamData, 0, 1);
    reprojectionVars[5] = cvmGet(_CamData, 0, 2);
reprojectionVars[6] = cvmGet(_CamData, 0, 3);

//if we are manually loading images, if images are loaded automatically, they are already in the proper variables
if(!ANALYSIS_MODE||GRAB_IMAGES)
{
    img1 = cvCreateImage(cvSize(grab->imageLeft->width,grab->imageLeft->height),
IPL_DEPTH_8U, 1);
    cvCvtColor(grab->imageLeft, img1, CV_BGR2GRAY);
    img2 = cvCreateImage(cvSize(grab->imageRight->width,grab->imageRight->height),
IPL_DEPTH_8U, 1);
    cvCvtColor(grab->imageRight, img2, CV_BGR2GRAY);

    imageSize = cvSize(img1->width,img1->height);
    img1r = cvCreateMat(imageSize.height,imageSize.width, CV_8U );
    img2r = cvCreateMat(imageSize.height,imageSize.width, CV_8U );
    disp = cvCreateMat(imageSize.height,imageSize.width, CV_16S );
    vdisp = cvCreateMat(imageSize.height,imageSize.width, CV_16S );
    depthImage = cvCreateMat(imageSize.height, imageSize.width, CV_8U );
}

//COMPUTE AND DISPLAY RECTIFICATION and find disparities
if( showUndistorted ){
    CvMat* mx1 = cvCreateMat( imageSize.height,
imageSize.width, CV_32F );
    CvMat* my1 = cvCreateMat( imageSize.height,
imageSize.width, CV_32F );
    CvMat* mx2 = cvCreateMat( imageSize.height,
imageSize.width, CV_32F );
    CvMat* my2 = cvCreateMat( imageSize.height,
imageSize.width, CV_32F );
    CvMat* pair;
    double R1[3][3], R2[3][3], P1[3][4], P2[3][4], Q[4][4];
CvMat _R1 = cvMat(3, 3, CV_64F, R1);  
CvMat _R2 = cvMat(3, 3, CV_64F, R2);  
CvMat _P1 = cvMat(3, 4, CV_64F, P1);  
CvMat _P2 = cvMat(3, 4, CV_64F, P2);  
_Q = cvMat(4, 4, CV_64F, Q);  

//compute variables needed for rectification using camera matrices, distortion  
vectors, rotation matrix, and translation vector  
cvStereoRectify(_M1, _M2, _D1, _D2, imageSize,  
_R, _T, &_R1, &_R2, &_P1, &_P2, &_Q, 0 );  

//Precompute maps for cvRemap()  
cvInitUndistortRectifyMap(_M1,_D1,&_R1,&_P1,mx1,my1);  
cvInitUndistortRectifyMap(_M2,_D2,&_R2,&_P2,mx2,my2);  

// RECTIFY THE IMAGES FOR CORRELATION  
pair = cvCreateMat(imageSize.height, imageSize.width*2, CV_8UC3 );  

cvRemap( img1, img1r, mx1, my1 );  
cvRemap( img2, img2r, mx2, my2 );  

if(SHOW_SAVE_DATA)  
{
    cvNamedWindow( "rectified", 0);  
    cvNamedWindow( "Disparity Map",0 );  
}

//run correlation routine  
stereoCorrelation();  

//if not analysis mode, save point cloud automatically for viewing  
if(!ANALYSIS_MODE)
stereoSavePointCloud();

if ( img1 && img2 ){
    CvMat part;

    // shows the left and right images in a window with epipolar lines showing
    features horizontally aligned
    if(SHOW_SAVE_DATA){
        cvGetCols( pair, &part, 0, imageSize.width );
        cvCvtColor( img1, &part, CV_GRAY2BGR );
        cvGetCols( pair, &part, imageSize.width,
                   imageSize.width*2 );
        cvCvtColor( img2, &part, CV_GRAY2BGR );
        for( j = 0; j < imageSize.height; j += 16 )
            cvLine( pair, cvPoint(0,j), cvPoint(imageSize.width*2,j),
                     CV_RGB(0,255,0));
        cvShowImage( "rectified", pair );
    }

    cvWaitKey(0);

    cvReleaseImage( &img1 );
    cvReleaseImage( &img2 );
}

// color files can be loaded to be rectified as well so they line up when
overlying texture on the point cloud
string colorFile;
cout<<"Enter color file name to rectify: ";
cin>>colorFile;

    // create images for color rectification, each band has to be remapped
    individually
    IplImage *colorImage = cvLoadImage(colorFile.c_str(),1);
CvSize colorSize = cvSize(colorImage->width, colorImage->height);
IplImage *bandB = cvCreateImage(colorSize, colorImage->depth, 1),
    *bandG = cvCreateImage(colorSize, colorImage->depth, 1),
    *bandR = cvCreateImage(colorSize, colorImage->depth, 1),
    *bandBr = cvCreateImage(colorSize, colorImage->depth, 1),
    *bandGr = cvCreateImage(colorSize, colorImage->depth, 1),
    *bandRr = cvCreateImage(colorSize, colorImage->depth, 1);

if (colorImage){
    CvScalar s;
    for (int j=0; j<colorImage->height; j++){
        for (int i=0; i<colorImage->width; i++){
            s.val[0] = cvGet2D(colorImage, j, i).val[0];
            cvSet2D(bandB, j, i, s);
            s.val[0] = cvGet2D(colorImage, j, i).val[1];
            cvSet2D(bandG, j, i, s);
            s.val[0] = cvGet2D(colorImage, j, i).val[2];
            cvSet2D(bandR, j, i, s);
        }
    }
    cvRemap(bandB, bandBr, mx1, my1);
    cvRemap(bandG, bandGr, mx1, my1);
    cvRemap(bandR, bandRr, mx1, my1);

    for (int j=0; j<colorImage->height; j++){
        for (int i=0; i<colorImage->width; i++){
            s.val[0]=cvGet2D(bandBr, j, i).val[0];
            s.val[1]=cvGet2D(bandGr, j, i).val[0];
            s.val[2]=cvGet2D(bandRr, j, i).val[0];
            cvSet2D(colorImage, j, i, s);
        }
    }
    cvSaveImage("Color_Rect.jpeg", colorImage);
}
else

    cout<<"Color image not found."

    cvReleaseMat( &mx1 );
    cvReleaseMat( &my1 );
    cvReleaseMat( &mx2 );
    cvReleaseMat( &my2 );
    cvReleaseMat( &img1r );
    cvReleaseMat( &img2r );
    cvReleaseMat( &disp );
}

void StereoFunctions::stereoReadCalibFile(char *calibFileName, StereoGrabber* grab){
    // Read in Calibration File
    CvFileStorage* fs = cvOpenFileStorage(calibFileName,0,CV_STORAGE_READ);

    // Information for Calibration File.
    // Calibration File Format.
    // (1) M1: Camera Matrix for camera 1 (left) [3][3] row by row
    // (2) M2: Camera Matrix for camera 2 (right) [3][3] row by row
    // (3) D1: Distortion Vector for camera 1 (left) [5]
    // (4) D2: Distortion Vector for camera 2 (right) [5]
    // (5) R: Rotation Matrix for camera 1 (left) wrt camera 2 (right) [3][3] row by row
    // (6) T: Translation Vector for camera 1 (left) wrt camera 2 (right) [3]

    CvMat* _M1 = (CvMat*) cvReadByName(fs,0,"M1");
    CvMat* _M2 = (CvMat*) cvReadByName(fs,0,"M2");
    CvMat* _D1 = (CvMat*) cvReadByName(fs,0,"D1");
    CvMat* _D2 = (CvMat*) cvReadByName(fs,0,"D2");
    CvMat* _R = (CvMat*) cvReadByName(fs,0,"R");
    CvMat* _T = (CvMat*) cvReadByName(fs,0,"T");
    CvMat* _CamData = (CvMat*) cvReadByName(fs,0,"CamData");
cvReleaseFileStorage(&fs);
stereoRectify(_M1, _M2, _D1, _D2, _R, _T, _CamData, grab);
}
Appendix B: Object extraction source code using Matlab

The code shown below is used with a GUI also developed in MATLAB, slider.m, for extracting objects from a point cloud that are within a certain distance of the camera system. Size and distance of the components can also be found using mouse clicks.

```matlab
function objectDetect(distance_in,points,image)

%variables for extracting distance measurements of components
B=379.29881;
f=3.8;
d=0.0049;
distance=distance_in*1000;
minDisp=B*f/distance/d;

PC=points;

rect=image;
disparity = reshape(PC(:,7),704,480)';
x=reshape(PC(:,1),704,480)';
y=reshape(PC(:,2),704,480)';
z=reshape(PC(:,3),704,480)';

%find all pixels that are above a minimum disparity (relating to the maximum distance
imagebw=disparity>minDisp;
```
% find the connected components
cconn = bwconncomp(imagebw, 4);
cconn2 = bwlabel(imagebw, 4);

area = zeros(1, cconn.NumObjects);

% find the size of each component
for i = 1:cconn.NumObjects
    % area(i) = bwarea(L == i);
    a = cconn.PixelIdxList(i);
    area(i) = size(a{1, 1}, 1);
end

sarea = sort(area, 'descend');

% keep objects that are over a particular size
objects = find(area > 1000);

M = zeros(337920, 1);

% put only components over a certain size in a filter matrix to apply to the
% rectified image and point cloud models
for i = 1:size(objects, 2)
    % M = M | (L == objects(i));
    a = cconn.PixelIdxList(objects(i));
M(a{1,1})=1;
end

M = reshape(M,480,704);

%apply connected components filter to remove smaller unwanted components
conn3=conn2.*M;

%apply filter to x,y,z point cloud data
X=x.*M;
Y=y.*M;
Z=z.*M;

%apply extracted component filter to rectified image
rect2=im2double(rect).*M;
figure,
subplot(2,1,1);
imshow(rect2);
subplot(2,1,2);
imshow(rect);

Dx=X;
Dy=Y;
Dz=Z;

X=reshape(X,size(X,1)*size(X,2),1);
Y=reshape(Y,size(Y,1)*size(Y,2),1);
Z=reshape(Z,size(Z,1)*size(Z,2),1);

for i=1:size(objects,2)
    a=conn.PixelIdxList(objects(i));
    xdist(i) = abs(max(X(a{1,1}))) + abs(min(X(a{1,1})));
    ydist(i) = abs(max(Y(a{1,1}))) + abs(min(Y(a{1,1})));
    zdist(i) = abs(max(Z(a{1,1}))) + abs(min(Z(a{1,1})));
end

dist(1,:)=xdist;
dist(2,:)=ydist;
dist(3,:)=zdist;
%dist

%save new point cloud with extracted objects shown
X((X==0))=[];
Y((Y==0))=[];
Z((Z==0))=[];
Points=zeros(size(X,1),3);
Points(:,1)=X;
Points(:,2)=Y;
Points(:,3)=Z;
save('objects.txt','Points','-ASCII');

figure('color','k'),
imagesc(conn3);
colormap(jet(256));

xdist=zeros(1,size(objects,2));
ydist=zeros(1,size(objects,2));
zdist=zeros(1,size(objects,2));

%display x,y,z reprojection for the pixel location of a mouse click on the
%figure
while(1)
    [x y] = ginput(1);
    D(1)=Dx(uint32(y),uint32(x));
    D(2)=Dy(uint32(y),uint32(x));
    D(3)=Dz(uint32(y),uint32(x));
    D
end

%figure,
%scatter3(X,Y,Z);

return
Appendix C: Camera Calibration Guide using Matlab

1) Download Camera Calibration toolkit from http://www.vision.caltech.edu/bouguetj/calib_doc/

2) Extract file to Matlab toolbox folder

3) Add Calibration Toolbox to search path in Matlab
   File->Set Path->Add Folder

4) type “calib_gui” at command line and hit enter
   GUI will pop up for calibration process

5) Select appropriate option for loading images

6) Navigate the Current Working Directory to the directory with the calibration images

7) Select “Image Names” from GUI

8) Enter base filename “left” / “right”

9) Select format of image
Images will be loaded and displayed

10) Select “Extract Grid Corners”

11) “Enter” to process all images

12) “Enter” (2x) to use default window size x and y to find checkerboard corners

13) “Enter” to determine number of squares automatically

14) Select Corners

   a. Select the first corner in the top left of the checkerboard pattern. This corner is set as the origin. Always select this corner first in each image and select the other three corners in the same order for each image
15) If red ‘Xs’ are not on the corners of each of the squares, follow directions to guess initial distortion to correct for radial distortion in the lens. Enter values (i.e. “-0.5”) to apply distortion to the grid until ‘Xs’ are placed correctly.
16) Repeat step 15 for all images in set

17) Select “Calibration”

Calibration routines are run on the data extracted

18) Select “Save”

Rename files to “Calib_Results_left” / “Calib_Results_right”

19) Repeat the entire process for other set of images

20) Exit the Camera Calibration Toolbox

21) Enter “stereo_gui”

Stereo Camera Calibration Toolbox GUI will display
22) Select “Load Left and Right Calibration files”
23) Copy Camera Matrices stored in variables “KK_left”, “KK_right” to the camera matrices “M1” and “M2” in the stereo calibration file
24) Copy Distortion vectors “kc_left”, “kc_right” to the distortion vectors “D1” and “D2” in the stereo calibration file
25) Find the Rodrigues rotation transform of the variable \( om \) to obtain the Rotation matrix.

\[ R=Rodrigues(om); \]

26) Copy Rotation Matrix “R” and Translation Vector “T” to the rotation matrix “R” and translation vector “T” in the calibration file
Appendix D: OpenCV Copyright Agreement

OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions for real-time computer vision.

OpenCV is released under an **BSD** license, it’s free for both academic and commercial use.

Example applications of the OpenCV library are: Human-Computer Interaction (HCI); Object Identification, Segmentation and Recognition; Face Recognition; Gesture Recognition; Camera and Motion Tracking, Ego Motion, Motion Understanding, Structure From Motion (SFM); Stereo and Multi-Camera Calibration and Depth Computation; Mobile Robotics.

Status

- The current release version is 2.0 (Sept. 30th 2009).
- Ongoing Announcements.

Documentation

- The OpenCV book is: *Learning OpenCV: Computer Vision with the OpenCV Library*
- The complete online reference manual for OpenCV: [C++, C, and Python]

Downloads

- You can download the release version (2.0) for *Linux* or *Windows*.
- To get the *OpenCV source code*, visit: [https://sourceforge.net/projects/opencvlibrary](https://sourceforge.net/projects/opencvlibrary). To build, see the *Install Guide*.

Support

- Send your question to the [OpenCV User Group](https://sourceforge.net/projects/opencvlibrary/)
- *Report a bug* or request new features or raise a documentation issue.
- *The OpenCV Wiki* is the OpenCV community's wiki.
Appendix E: Optometrists Network Copyright Agreement

If you've got Stereo Vision, Count Your Blessings!
According to the Web site of the American Academy of Optometry, September 1996: "many occupations are not open to people who have good vision in one eye only [that means people without stereo vision]."

Here are a few examples of occupations that depend heavily on stereo vision:

- Baseball player
- Waitress
- Driver
- Architect
- Surgeon
- Dentist

Here are just a few examples of general actions that depend heavily on stereo vision:

- Throwing, catching or hitting a ball
- Driving and parking a car
- Planning and building a three-dimensional object
- Threading a needle and sewing
- Reaching out to shake someone's hand
- Pouring into a container
- Stepping off a curb or step

Are you sure you've got Stereo Vision?
It's hard to know what you've missing if you've never had it. Do you see with both your eyes? Are your two eyes similar or different in sight?

This Is Just A Test -- of Your Stereo Vision System
Are both your eyes turned on and working together as a team? Try this easy test and find out if you are a good candidate for 3D viewing. It's The Stereo Test and it only takes a minute!

* To read more about the difference between the side vision of horses and the frontal vision of humans, check out Dr. Cooper's cool explanation about the evolution of non-eye vision in our ancestors (humans, lions, tigers, sharks, etc.).

You may have arrived at this page because you were searching for an explanation as to why you can't see Spy Kids 3D or The Adventures of Lava Girl and Shark Boy 3D. Can you see 3D? If you can't, keep reading and find out why!!!

---

Two Eyes = Two Separate Views!
Human beings generally come equipped with two eyes and one head. If you have any doubts about your equipment or your ability to see 3D, check out Why Some People Have Trouble Seeing 3D before continuing.

Unlike horses, humans have two eyes located side-by-side in front of their heads. Thanks to the close side-by-side positioning, each eye takes a view of the same area from a slightly different angle. The two eye views have plenty in common, but each eye picks up visual information the other doesn't. Have you ever compared the different views of your right and left eye? The Eye Hop Game lets you do just that.

Two Eyes = Three Dimensions (3D)
Each eye captures its own view and the two separate images are sent on to the brain for processing. When the two images arrive simultaneously at the back of the brain, they are united into one picture. The brain combines the two images by matching up the similarities and adding in the small differences. The small differences between the two images add up to a big difference in the final picture! The combined image is more than the sum of its parts. It is a three-dimensional stereo picture.

The word "stereo" comes from the Greek word "stereos" which means firm or solid. With stereo vision you see an object as solid in three spatial dimensions: width, height and depth. Or, in a, y and z. It is the added perception of the depth dimension that makes stereo vision so rich and special.

Stereo Vision Has Many Advantages
Stereo vision—or stereoscopic vision—probably evolved as a means of survival. With stereo vision, we can see WHERE objects are in relation to our own bodies with much greater precision—especially when those objects are moving toward or away from us in the depth dimension. We can see a little bit around solid objects without moving our heads and we can even perceive and measure "empty" space with our eyes and brains.