A CAD-Interactive Software Package for 
the Synthesis of Planar Disk Cams

by

Stephen Richard Payne

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

Mechanical Engineering

Approved:

C.F. Reinholtz, Chairman
O. K. Eyada
T. D. L. Walker

February 14, 1994

Blacksburg, VA
c. P

LD
5659
4855
1199

P296
c. 2.
A CAD-Interactive Software Package for the Synthesis of Planar Disk Cams

by

Stephen Richard Payne

Committee Chairman: Dr. Charles F. Reinholdt
Mechanical Engineering

(ABSTRACT)

The objective of this thesis is to provide an interactive system for the graphical design and analysis of planar disk cams.

The software written for this thesis, the Cam Design System, operates as an add-on application that runs within the CADKEY® graphical environment and inherits all the CADKEY® software functions. It allows the user to specify different follower and motion curve parameters to synthesize a cam, and will analyze the cam for pressure angle flaws. It will also produce output in a numerical data format, as CAD drawings, as a detailed design report, or as NC data.

This thesis first discusses what software is already available to perform similar synthesis and analysis, and the extent of documentation of cam design and the use of conjugate
geometry theory in literature. Next, it describes how the C-code integrates with the CADKEY® graphical system. The next sections describe all the design theory used in the software, including all the conjugate geometry theory and full descriptions of each of the motion curve characteristics used in the software. The subsequent section describes the software itself and how it is structured, and provides a few examples of designing a cam. The last section sums up the work of this thesis, and provides some recommendations for future work in this area.
Acknowledgments

First and foremost I would like to graciously thank Dr. Charles Reinholtz for his guidance, support and friendship throughout my graduate studies. I have been associated with him for about 4 years now, from ASME to Robotics class to undergraduate research, and it has been quite a pleasure working with and knowing him.

I would also very much like to extend my appreciation to the other members of my advisory committee, Dr. Osama Eyada and Professor Thomas Walker. Dr. Eyada's knowledge of the use of computers in manufacturing and program interfacing with C in CAD environments has helped my learning of the manufacturing field and my programming abilities a great deal. Professor Walker's more than thorough knowledge of and experience with CADKEY® and its user interface has aided my understanding of it, and I would like to extend a special thanks to him for venturing outside his standard domain to serve on this committee.

Thanks also goes to Dr. Sanjay Dhande, who helped to get this project off the ground in its initial development stages, and for developing the conjugate geometry methods in the first place. And thanks to Steve Shooter for all the time he spent testing the software and helping to extend my knowledge of kinematics.
Another thanks goes to Cadkey, Inc., for providing the Robotics & Mechanisms Group with copies of the CADKEY® v6.0 Software and the CADKEY® Software Development Kit v6.0, and for their initial interest in this project.

A big thanks goes to friends, of which there seem to be many. Thanks to the gang in the Robotics & Mechanisms Group (especially Ravi and Steve) and the CAD Lab for making my time in the office amusing. Thanks to my numerous roommates over the years (particularly Rally, Greg, and of course Ben, among others) for making time fly. Thanks to the friends who kept making weekend visits to pass the time (and to seemingly slow my progress here!), and to those who have waited this long for me to finish my studies. And thanks to all the people I have come to know during my 6½ year stay at Virginia Tech - you have made my time in Blacksburg (and beyond) quite memorable. Unforgettable really.

And finally I impart a big sincere thanks to my family for all the love and support they have given me throughout my years at Virginia Tech, before and after (I hope)! Thanks mom, dad, and Lesley. And a special thanks to my grandfather, whose perpetual reminders of the importance of an education never left my head, whether I wanted it to or not.
Table of Contents

1. Introduction and Motivation ................................................................. 1

2. Literature Review .................................................................................. 3
   2.1 Software Review .............................................................................. 3
   2.2 Mechanism and Cam Theory Literature .......................................... 10

3. Background of Software Development .................................................. 14
   3.1 Methodology and Approach .............................................................. 14
   3.2 The CDE Advantage ....................................................................... 15
   3.3 Use in Other Graphical Environments ............................................. 16

4. Cam Design Theory .................................................................................. 18
   4.1 Basic Cam Principles ....................................................................... 19
       4.1.1 Cam Types .............................................................................. 19
       4.1.2 Cam Nomenclature .................................................................. 21
   4.2 Follower Types ................................................................................ 21
   4.3 Envelope Theory .............................................................................. 23
   4.4 Conjugate Geometry Theory .............................................................. 26
       4.4.1 Translating Flat-Faced Follower Case ....................................... 29
       4.4.2 Oscillating Flat-Faced Follower Case ....................................... 32
       4.4.3 Translating Roller Follower Case ............................................ 35
4.4.4 Oscillating Roller Follower Case .................................................. 38
4.5 Implications of Conjugate Geometry Results ................................. 42

5. Cam Motion Specifications ................................................................. 45

5.1 Basic Functions ............................................................................. 47
  5.1.1 Ramp Function .......................................................................... 48
  5.1.2 Trapezoidal Acceleration Function .......................................... 49

5.2 Trigonometric Functions ............................................................... 54
  5.2.1 Simple Harmonic Motion Function .......................................... 55
  5.2.2 Cycloidal Displacement Function ........................................... 57
  5.2.3 Half Cycloidal Function ............................................................ 59

5.3 Modified Functions ....................................................................... 63
  5.3.1 Modified Sine Function ............................................................. 64
  5.3.2 Modified Trapezoidal Function ............................................... 68

5.4 Polynomial Functions ................................................................... 72
  5.4.1 3-4-5 Polynomial Function ..................................................... 74
  5.4.2 4-5-6-7 Polynomial Function ................................................ 76

5.5 Cam Design Considerations ......................................................... 78

5.6 Pressure Angle Considerations ..................................................... 81
  5.6.1 Translating Flat-Faced Follower .............................................. 83
  5.6.2 Oscillating Flat-Faced Follower ............................................... 84
  5.6.3 Translating Roller Follower ..................................................... 85
  5.6.4 Oscillating Roller Follower ..................................................... 86

6. Program Description and Examples .................................................. 87

Table of Contents
6.1 Overall Description of Software ........................................... 87
  6.1.1 Software Development ................................................. 87
  6.1.2 Running the Software .................................................. 89
  6.1.3 Program Requirements ................................................. 91
6.2 Specific Structure of Code ................................................. 94
6.3 Menu Structure and User Information .................................... 97
6.4 Specific Examples .................................................................. 106
  6.4.1 Translating Flat-Faced Follower Example ........................... 106
  6.4.2 Translating Roller Follower Example ................................. 108

7. Conclusions and Recommendations ........................................ 115
  7.1 Summary ........................................................................... 115
  7.2 Future Work ...................................................................... 116

8. References ............................................................................. 118

Appendix A: Source Code Listing - CAM.H .................................. 124
Appendix A: Source Code Listing - GLOBAL.H ......................... 127
Appendix A: Source Code Listing - OTHER.DEF ....................... 128
Appendix A: Source Code Listing - MAKEFILE .......................... 130
Appendix A: Source Code Listing - CAM_LIB.LST ...................... 132
Appendix A: Source Code Listing - CAM.TRE ............................. 133
Appendix A: Source Code Listing - CALC.C ................................. 134
Appendix A: Source Code Listing - INTER.C ............................ 150
Appendix A: Source Code Listing - MISC.C ............................... 157
<table>
<thead>
<tr>
<th>Appendix A: Source Code Listing - OUTPUT.C</th>
<th>169</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A: Source Code Listing - PROFIL.C</td>
<td>202</td>
</tr>
<tr>
<td>Appendix A: Source Code Listing - RENDER.C</td>
<td>227</td>
</tr>
<tr>
<td>Appendix A: Source Code Listing - SPECS.C</td>
<td>247</td>
</tr>
<tr>
<td>Appendix A: Source Code Listing - UTILS.C</td>
<td>267</td>
</tr>
<tr>
<td>Vita</td>
<td>277</td>
</tr>
</tbody>
</table>
List of Figures

Figure 4.1: Disk Cam and Roller Follower ................................................................. 20
Figure 4.2: Cam Nomenclature .................................................................................. 22
Figure 4.3: Different Types of Cam and Follower Arrangements.............................. 24
Figure 4.4: Cam Profile Defined by an Envelope of Follower Positions .................... 25
Figure 4.5: Translating Flat-Faced Follower ............................................................ 28
Figure 4.6: Oscillating Flat-Faced Follower ............................................................... 33
Figure 4.7: Translating Roller Follower .................................................................... 36
Figure 4.8: Oscillating Roller Follower .................................................................... 39
Figure 4.9: Bending Moment in a Translating Flat-Faced Follower............................ 43
Figure 5.1: Motion Characteristics for the Ramp Function ........................................ 49
Figure 5.2: Motion Characteristics for the Trapezoidal Acceleration Function .......... 55
Figure 5.3: Motion Characteristics for the Simple Harmonic Motion Function .......... 57
Figure 5.4: Motion Characteristics for the Cycloidal Displacement Function .......... 60
Figure 5.5: Motion Characteristics for the Half Cycloidal Function.......................... 63
Figure 5.6: Motion Characteristics for the Modified Sine Function.......................... 67
Figure 5.7: Motion Characteristics for the Modified Trapezoidal Function .............. 73
Figure 5.8: Motion Characteristics for the 3-4-5 Polynomial Function ...................... 75
Figure 5.9: Motion Characteristics for the 4-5-6-7 Polynomial Function .................... 77
Figure 5.10: Simple Harmonic Motion-Modified Trapezoidal RDFS Motion Program ... 80
Figure 6.1: Top-Level Menu for the Cam Design System ........................................ 90
Figure 6.2: Flow Diagram of Typical Cam Design System Run ................................ 98
Figure 6.3: Cam Parameters Menu ................................................................. 99
Figure 6.4: Motion Segments Menu ................................................................. 101
Figure 6.5: Motion Profile Menu ................................................................. 102
Figure 6.6: Cam Profile Specifications Menu ............................................... 104
Figure 6.7: Cam Profile for Translating Flat-Faced Follower Example .......... 109
Figure 6.8: Motion Characteristics for Translating Flat-Faced Follower Example .... 110
Figure 6.9: Cam Profile for Translating Roller Follower Example .................. 112
Figure 6.10: Motion Characteristics for Translating Roller Follower Example ...... 113
1. Introduction and Motivation

Today’s society is fast-moving and demanding, and it has put pressure on industry to perform and produce results quickly. In the design arena, anything that will increase the speed and accuracy of the design process or that helps to improve the performance of current designs is readily adopted.

The area of mechanism design affects most high-technology industries. The exact design of this machinery is usually tedious, however, and requires an accurate idea of what the machine inputs and outputs are to be. The area of cam design in particular is a time-consuming and error-prone one, especially when a detailed design is essential. Though cams are generally more expensive than linkages, they are much easier to lay out to give a specific output function. The current method of manual cam design involves viewing the cam from different frames of reference, and requires the user to draw the follower around the cam at successive incremental positions. Once these positions are drawn in the user must draw in 'by eye' the cam profile curve tangent to these follower positions. These methods, no matter how carefully performed, are more inaccurate, more prone to error and are quite time-consuming. They also fail to provide the user with the ability to alter some facets of the design quickly, or to machine the cam directly from the design drawing or blueprint.

This thesis discusses a software package called the Cam Design System which automates the detailed cam design process within a CAD (Computer-Aided-Design) graphical
environment. It is written totally in the C programming language, and is compiled with the MetaWare® High C/C++™ v3.1 compiler (MetaWare®, 1993). It is designed to interface with the popular PC-based CAD software package CADKEY® v6.0, and runs totally within this graphical environment while giving the user complete access to CAD-related functions and complete control of the cam design process (Cadkey, 1993b). Using this software the whole cam design process takes much less time than would the manual design. The user also has better control of the design and is better able to make changes in the design. From a manufacturing point of view, the cam will now be much easier to produce since each part of its geometry has been mathematically defined in the CAD database and instructions can be easily downloaded to a NC machine. This software gives the designer the opportunity to draw cam design into the overall design process by integrating the CAD system as a central tool of the designer. In addition, it is an easy to use PC-based software package intended for use by designers. And though this software is intended in part as a learning tool, it also functions to a large degree as a powerful design tool for a wide range of engineering applications.
2. Literature Review

2.1 Software Review

The current design of cams and most other related mechanisms is tedious at best. Formerly, the use of graphical construction was the major method of mechanism synthesis. Even more than ten and twenty years ago, Erdman (1985), Mittelstadt (1985), and Freudenstein (1973) have noted the need for some kind of computerized or automated mechanical design and analysis system with interactive graphics. Also, Timmins (1992) discusses specific ideologies of software development, types of software, and project management to a great detail.

Fortunately, many design and analysis algorithms have since been automated, and the computerized design of mechanisms has improved over the years. Prior to the creation of the Cam Design System software, some existing software packages were investigated for their features and limitations. A brief history and review of each of these follows.

One of the first packages to utilize kinematic synthesis for mechanisms in conjunction with interactive graphics was "KINSYN" (Rubei and Kaufman, 1977; Kaufman, 1978; Erdman, 1993). This package was developed originally in the late 1960's and included later updates over the next ten or more years to include more advanced linkage synthesis capabilities. Some other linkage synthesis software packages developed later include "RECSYN" (Waldron and Song, 1981), "SYNTRA" (Barker and Tso, 1988),
"LINCAGES-4" (Erdman and Gustafson, 1977; Erdman and Riley, 1981), "MECSYN" (Sivertsen and Myklebust, 1980; Myklebust et al., 1985), and "LINCAGES-6" (Chase et al., 1981). "LINCAGES-4," for instance, provides for the synthesis of four-bar linkages (Erdman and Gustafson, 1977; Erdman and Riley, 1981). It performs body guidance synthesis for three- or four-precision point positions and then presents the entire set of solutions. An interactive graphical environment allows the user to drag the ground link and moving pivots of the linkage to any position on the design curves and instantly see the reconstructed mechanism. Also, synthesis and analysis windows allow for the viewing of information necessary for designing an optimal mechanism that satisfies angle, size, pivot location, velocity, path, interferences and other design constraints. "LINCAGES-6" is a related package started in the late 1970's that extends the "LINCAGES-4" package to the synthesis of six-bar mechanisms (Chase et al., 1981). This package also allows the elimination of undesirable solutions according to bad transmission angle characteristics or branch, order and Grashof defects. Demetriou (1987) has also developed software in Pascal, "CYPRUS," for the three- and four-position synthesis for path generation and body guidance of four-bar and six-bar linkages. Leinonen and Paarni (1990) have documented their development of "AUCASYN," a program which handles the synthesis of four-bar mechanisms, provides for workspace definitions for robots and machines, and handles type and number synthesis for a number of mechanisms. Salem and Manoochehri (1990) discuss an AutoCAD®-based design system that is programmed using the AutoLISP® macro-type language interpreter within the AutoCAD® software. This software performs kinematic and dynamic analysis, and aids in the optimal design and synthesis of planar mechanisms.
"Lear_Links v2.4" is a four-bar linkage synthesis software package that includes stand-alone programs for three- and four-precision point synthesis (Lear Com, 1993). It analyzes all kinematic properties and allows the user to animate the mechanism. It also provides analysis graphs and maps of solutions to determine the feasibility of the four-bar linkage.

"Inertia/InMotion" provides for the dynamic, kinematic and static analyses of complete mechanical systems and their assemblies (Modern Computer Aided Engineering, 1989). It provides a variety of links, joints, constraints and loading options. In addition, the entire system can be animated, and the results of the kinematic analyses of position, velocity and acceleration can be plotted.

"Analytix" is a sketch-based mechanism design and analysis tool that combines integrated static, kinematic, dynamic and tolerance analyses (Saltire Software, 1993a). It operates in a Microsoft® Windows™ environment and provides for animation and various other tabular and graphical outputs. Its specifications include numerous geometry types, dimensions, constraints, and several different viewing capabilities. It also provides measurement reports, and calculation and viewing tools.

"Micro-Mech" performs primarily planar mechanism analysis (Dynacomp, 1992). This software deals with planar mechanisms containing revolute and sliding joints. It also performs kinematic, dynamic, time response, and force analyses.

"Kinematix" allows the user to perform optimization on linkage designs chosen from an on-line atlas of design and analysis configurations (Saltire Software, 1993b). It operates
in a Microsoft® Windows™ environment and allows the user to analyze the motion of a wide array of coupler curves. It also provides for velocity and acceleration measurements, as well as animation and other graphics capabilities.

"DE/MEC v2.0" associates CAD geometry with a mechanism, so that it can use properly shaped and accurate geometric components for the application (Desktop Engineering International, 1991). Also, the package allows the user to optimize the design to user-defined constraints, some of which may include a maximum and/or minimum link length, the position and orientation of the links, and a timing requirement. It also provides for the animation of the mechanism.

Algor's "Dynapak" and "Kinepak" packages provide a system for kinematic and rigid-body dynamic analysis (Algor, 1990). These systems serve to simulate and predict the motion of several mechanisms, and provide a small amount of other analysis capabilities.

Aside from the abundance of general mechanism design software, there is a good deal of software aimed specifically at cam design. For instance, Gandhi (1985) has developed a package in the FORTRAN programming language, "ANICAM," that serves as a CAD/CAM interface to design and model disk cams in three dimensions. This package provides for the selection and display of several different motion curve types for the cam profile. However, this software serves as a module of the computer-aided-design package CADAM® (Dassault Systemes Corporation, 1992), and is therefore limited in its portability to other systems. Cleghorn and Podhorodeski (1988) present a software package also written in FORTRAN, "CAMPRF," that synthesizes disk cam profiles using analytical techniques, produces renderings on a DOS® graphics screen, and runs only on
IBM® PCs or compatibles. These two programs are similar in their analytical techniques used to develop the cam profile, and they both produce an accurate rendering of the cam profile on a computer display.

Some specific texts on kinematics and cam design are accompanied with software routines on disk to supplement the text. Norton's (1992) "Design of Machinery" software actually contains several different menu-driven, interactive programs for the design and analysis of mechanisms and machines. Kinematic analysis and synthesis of four-bar, five-bar and six-bar linkages is offered, along with dynamic analysis capabilities for four-bar linkages. In addition, some basic cam and follower systems can be designed and analyzed. Dynamic analysis of the slider-crank linkage found in internal combustion engines is also included. The program actually uses envelopes of surfaces to define the cam profile; however, the user-interface and the overall menu system is somewhat awkward. Jensen's (1987) "Cam Design and Manufacture" software performs both synthesis and analysis and is written in the BASIC programming language. However, this software is limited only to translating and oscillating roller followers and has a poor user interface. Mabie and Reinholtz's (1987) "Kinematic Toolkit Release 1.0" is also written in BASIC and performs a variety of synthesis and analysis functions for gears, cams and linkages. It is quite useful in the synthesis of four-bar linkages and has some application to translating and oscillating cam-and-follower systems; however, it has limited use of graphics and limited application for full-fledged cam design problems.

Aside from these commercially available and privately developed software packages, there is also a wide array of privately developed and shareware-type programs available that run the gamut of mechanism design and analysis. Most of these programs tend to be
written with a narrow objective or with a very specialized application in mind. They have been written primarily in the BASIC and FORTRAN programming languages, and even include some Pascal and C-code. For instance, Novak (1992) presents a small set of FORTRAN code that allows the user to specify the parameters for the cycloidal motion of an oscillating or translating elliptical follower and disk cam in contact. This program will also produce the data for the point of contact between the follower and the cam, and for the lift profile of the cam. Carpenter (1992) uses C-code to design cams with a user-customized trapezoidal acceleration profile throughout. "CAMSYN," by Tjung and Coe (1992), is coded in C and uses graPHIGS for its graphics library. "PHIGS," or the Programmer’s Hierarchical Interactive Graphics System, is an ISO standard collection of powerful graphical interface routines used to display and interact with three-dimensional images (Gaskins, 1992). "CAMSYN" is a menu-driven computer-aided cam design program that provides for either the harmonic or cycloidal motion of a simple radial roller follower. The program solves for the displacement, velocity and acceleration of the cam, and will also animate the cam and follower in motion.

Besides simple stand-alone programs such as these, there are a few commercially available add-on products that operate in conjunction with AutoCAD®. For instance, the "Power Transmission" add-on product streamlines the design of cams and gears within the AutoCAD® software (Softdesk, 1993a). It allows the user to select follower type, motion curve types, and other specific parameters. after which time the cam and follower path are instantly created. Several analysis options are also made available to aid the evaluation of the cam design, including follower visualization and an automatic displacement diagram. The "NC Link" product allows the user to translate the part geometry in AutoCAD® to a format compatible with several different CAM (Computer-
Aided-Manufacturing) applications, and does not require the user to modify the original design geometry (Softdesk, 1993b). The "DADS v4.5" software performs kinematic and dynamic analysis of mechanical systems and allows the user to build computer models of real-world systems with its extensive library of mechanical elements (Autodesk, 1989). The "MSC/pal 2 v3.5" software package identifies stress and vibration problems with mechanical systems in general (Autodesk, 1989). The "FOURBAR v1.5" program performs synthesis and analysis for four-bar linkages (Autodesk, 1989). This is an interactive program that allows the user to easily edit link lengths, angular increments, and points on the coupler link. Both "MasterCAM v3.01" and "SmartCAM v4.5" are programs that run in conjunction with AutoCAD®, where each creates CNC code for mechanisms and drawings in general (Autodesk, 1989). These packages both provide for many different tool capabilities, surface operations and NC machine types. In short, most of these AutoCAD®-related products serve to help in general drafting and designing tasks. They also have some application in mechanism analysis and in creating NC instructions to produce a part.

As it stands, most of the software commercially available seems to be applicable primarily to the development of linkage and/or gear systems, and not so much for the design of cams. These packages also tend to devote much to the analysis portion of the design problem as opposed to the synthesis task. Many of these software packages are too narrowly focused and are therefore of limited use, perhaps concentrating only on one particular type of mechanism arrangement, like a three-dimensional gear train for instance. Some also include mechanism analysis as their only function. On the other hand, some packages are far too broad in scope to be of value. The packages that strive to
synthesize and analyze all general mechanisms from scratch sometimes seem to be
difficult to operate, are on the whole more expensive, and are in limited use.
In conclusion, no software package offers the ultimate mechanical design tool for cam
and follower systems. Many of the packages are limited to only linkage design. Also,
roughly half of these packages do not offer any synthesis capabilities, and instead are
devoted only to mechanism analysis functions.

2.2 Mechanism and Cam Theory Literature

There is a great abundance of literature available pertaining to general mechanism design
and analysis. Topics in kinematics literature range from type synthesis of four-bar and
six-bar linkages, to gear design and stress analysis, to cams and far beyond. Specifically,
Mabie and Reinholtz (1987) have documented many of the current methods and theories
of kinematics and overall machine design in their text, as have Barton (1993), Norton
Similarly, Paul (1979), Soni (1974), Martin (1982), Hardison (1979), Tao (1967),
Kimbrell (1992), Hinkle (1960), and Gans (1991) have also meticulously documented
much of this basic theory. All of these publications serve as general resources for
mechanism design and analysis, and most have some mention of general cam design
theory, including nomenclature, different cam and follower types, and several different
motion curve types. Most of these also include mention of some analysis techniques, cam
sizing requirements, and cam profile characteristics. Several even go beyond this and
discuss other not-so-common cam and follower arrangements, and force and dynamic
considerations. Erdman and Sandor (1984), for instance, have published two volumes concerning many extremely detailed aspects of advanced mechanism synthesis and analysis, and also include topics on mechanism dynamics and spatial mechanisms.

In the area of cam theory itself, Rothbart (1956) was one of the first to publish a definitive source regarding cam design and analysis. This text includes basic cam curves and discusses size and profile determination, dynamics of high speed cams, force analysis, and cam surface materials and stresses. Both Molian (1968) and Neklutin (1969) have published a general text on cam theory and consider many different design details in defining the cam contour. For instance, they cover several useful examples of different cam profiles, include detailed analyses of the specification of cam motions, and take into account forces and stresses on the follower and cam, as well as the torque experienced by the camshaft. Tesar and Matthew (1976) have authored a text which details not only basic cam theory, but also includes one- and two-degree-of-freedom synthesis concepts and discusses the thorough analysis of modeled cam systems. Chakraborty and Dhande (1977) talk about general considerations in cam synthesis and analysis, undercutting and mechanical error. More importantly, they also take cam theory into two- and three-dimensional cases using purely generic analytical techniques to solve for specific cases of cam and follower systems. Chen (1982) also authored a definitive cam theory text and thoroughly covers profile synthesis, contact stresses and wear, force analysis, and cam system dynamics. Both Jensen (1987) and Norton (1992) have given excellent descriptions of the cam design process. These texts entail everything from cam and follower systems, motion programs, and cam profile determination to pressure angle and radius of curvature considerations, cam analysis with respect to force and stresses, and manufacturing and practical design considerations. In addition, they discuss the dynamic forces, velocities and accelerations involved in cams.
Chironis (1991) has put out an excellent descriptive reference text on a number of different types of mechanisms. Chen (1977) discusses the dynamics and kinematics of cam-and-follower systems and where the application of these theories most need additional development. Rothbart (1985) has put together a general reference book on mechanical design, gears, and other mechanisms, and mentions cam dynamics and cam profile accuracy in addition to basic cam theory. Jensen (1991) also provides an in-depth reference, replete with information pertaining to many complex and unusual mechanisms. Angeles and López-Cajún (1991) discuss cam theory in general, but concentrate more on the optimization of cam mechanisms with regard to the size of the cam and the stress endured by the follower. They also discuss the use of custom-made motion curves composed of several spline functions. Also, Erdman (1993) has put together an excellent resource detailing the history and milestones of the development of kinematic theory over the past four decades.

The theory of conjugate geometry has not yet been too widely applied to the design of cam mechanisms. The theory of conjugate geometry and the definition of a cam surface by its point of contact with a follower essentially originated from the theory of envelopes and conjugate surface characteristics. Boltyanskii (1964) discusses the geometric concept and solution methods of envelopes and investigates some problems in kinematic applications involving envelopes. Faux and Pratt (1983) investigate the applications and theory of mathematical and geometrical representation techniques developed for computational geometry. They also give the mathematical definition of an envelope. These theories have become increasingly significant for use in manufacturing applications. Chakraborty and Dhande (1977) discuss many generalized models of two- and three-dimensional cam systems. Their work makes it possible to determine all
important cam parameters such as the cam profile and pressure angle in a closed-form solution. Voruganti (1990) also discusses the application of conjugate geometry in symbolic and mathematical form for application in design and manufacturing functions. Tsay and Hwang (1992) discuss the particular application of envelope theory in determining camoid profiles with translating followers. Wilson and Sadler (1993) also discuss the general application of envelope theory as a method of analytical cam design, and detail some of this analysis for several different types of followers.

These theories serve as the forerunner to the modern conjugate geometry theory, where a purely analytical closed-form model of a specific cam and follower system can be solved each and every time. This solution of conjugate geometry for the point of contact between the cam and a follower provides the full cam profile for a complete rotation of the cam.
3. Background of Software Development

3.1 Methodology and Approach

The current graphical method of cam design is subject to small cumulative errors and is a complicated and painstaking process. With the rapid availability and affordability of high-speed, powerful personal computers, the creation of a robust, user-friendly software for cam design appeared to be a logical next step. A computerized design system eliminates much of the effort and imprecision of manual cam design. Furthermore, much more detail and control can be achieved with the use of user-designated follower motions. In addition, small changes can be made more easily to existing designs while in a graphical design environment.

The Cam Design System has been designed for both educational and industry use. It has been created to be applied as an interactive learning tool along with traditional teaching methods in a university environment. Ideally, a student would be able to design a cam manually. Then he or she would use the Cam Design System to design that same cam, but much faster, more accurately, and with many more detailed specifications. For instance, these other details and controls might include different motion curve types for a rise or fall motion of the follower, or more detailed specifications for follower velocities and accelerations at certain points during the cam rotation.
3.2 The CDE Advantage

In order for a full cam design system such as this to be as effective as possible, an interactive graphical environment is not only superior but essential. Most graphical environments for current mechanism design software are provided simply through the DOS® environment. This however requires much complex coding for the programmer, and requires a great deal of brainstorming and programming just for the user-interface portion of the software. Therefore, the Cam Design System was written, compiled, and executed as a CDE (CADKEY® Dynamic Extensions™) mechanism running under the CADKEY® computer-aided design software (Cadkey, 1993b). The CDE mechanism is a more versatile, powerful, and easy to use extension of the CADL (CADKEY® Advanced Design Language™), which itself serves as a macro programming language internal to the CADKEY® package (Cadkey, 1993a). In other systems some similar mechanisms are known as Dynamic Link Libraries (DLL), Run-Time Linking (RTL), and Shared Objects (Cadkey, 1993b). However, the scope of the CDE mechanism encompasses more than this in that it can be executed across applications within the CADKEY® environment. In other words, applications developed as CDE modules are executed as extensions of the CADKEY® software.

For a programmer to write software for use in any graphical environment there are certain commands specific to each compiler and running environment to initialize the graphics screen and to execute these graphics commands. For the purpose of coding, the CDE mechanism acts fundamentally as another of these graphical environments. Moreover, the amount of coding required in producing the user-interface is diminished greatly; many commands are taken care of through special function calls to the CADKEY® database.
Some of these built-in user-interface options accessible to the user include a top-level menu system and dialog boxes for ease of data entry and general interaction. From a user's point of view, the CDE mechanism also provides all the functionality and elegance of the stand-alone CADKEY® software. Some of these enhanced commands include autoscaling and zooming in on pieces of the drawing, switching views of the screen, rotating the drawing in the current viewport, and saving or printing any piece of the cam drawing at any point during the program execution.

Within the CADKEY® framework, the CDE mechanism takes care of establishing all necessary connections and function calls while it is running. And once the CDE is started, it executes functions as if they were a part of the original CADKEY® code. The CDE comes as a big advantage to applications developers in that they may build large applications without the loss of performance or speed relative to the CADKEY® software itself (Cadkey, 1993b). In other words, the application will run at the same speed as the CADKEY® software and can be unknowingly integrated into its environment. From the point of view of the user, the CDE mechanism provides a greater number of options and will increase overall design productivity and efficiency.

3.3 Use in Other Graphical Environments

It should be noted that the CDE mechanism and CADKEY® were chosen for convenience, support and overall availability. CADKEY® is a popular and easy-to-use PC-based computer-aided design software package used by many universities and
industries nationwide. AutoCAD® also possesses a similar mechanism called ADS (AutoCAD® Development System™) used for porting C-code into its graphical environment (Autodesk, 1990a). This mechanism allows one to program external AutoCAD® applications in C that, when loaded into the AutoCAD® environment, appear identical to functions written in AutoLISP®. AutoLISP® is a macro-type programming language and is essentially an implementation of the LISP programming language implanted within the AutoCAD® software package (Autodesk, 1990b). This mechanism is akin to the CADKEY® CADL system. To use this Cam Design System software in the AutoCAD® environment would theoretically require only simple changes to the format of the graphical function calls to match the required parameters in each of the C-code header files and changes to the specific user-interface function calls for things such as dialog boxes and menu routines. In fact, most of this code translation would be devoted to the user-interface characteristics of the program. Aside from the CADKEY® and AutoCAD® software packages, this software can be integrated into the graphical environment of any CAD package that possesses the capabilities for porting C-code and executing it as an integrated application.
4. Cam Design Theory

There are many things that need to be considered in the design of a cam and follower system. The parameters for the design of any cam are specific to the application for which it is intended. For each particular application or problem the required inputs and outputs must be determined and then utilized to design the cam profile.

By far the most common problem in cam synthesis is to be given the desired motion of the follower and to create the cam profile necessary to give the follower this required motion. Designing a cam from a desired motion profile is a synthesis problem that can be solved every time (Mabie and Reinholtz, 1987). Another synthesis problem may occur when it is necessary to produce a cam according to a specific force or velocity function. Yet another distinct problem is one of cam analysis, where a cam shape is assumed and is then analyzed for what exact motion characteristics (displacement, velocity, acceleration and jerk) this cam profile will provide. Specific cam sizing and manufacturing requirements such as undercutting and cusp problems are also considered in cam analysis. The Cam Design System addresses only the first of these problems. The user specifies the necessary follower motion using the motion segments described in the following chapter of this thesis, and the cam contour is created for these specifications.
4.1 Basic Cam Principles

4.1.1 Cam Types

Since each cam design is specific to a certain application, the actual type of cam must first be chosen. The type of cam needed is dependent once again on the specific configuration of the system, what type of follower motion is required, and what type of force and torque is driving the cam.

Disk cams turn about a shaft perpendicular to the plane of the cam. The follower usually contacts the cam on the outside perimeter of the cam surface, and is usually driven by the cam on its upward motion. On the return motion the follower is most often driven by a return spring or by the force of gravity. Disk cams are in widespread use in industry; a typical application is in the automobile valve train. In a valve train, several disk cams are keyed to a camshaft that passes through the center of each separate cam. These cams in turn drive the valves of the internal combustion engine. A typical disk cam and roller follower are shown in Figure 4.1.

Although all disk cams must have some thickness, profile synthesis is usually treated only in two dimensions. The third dimension of a cam, depth, is considered only in the load bearing portion of the design.
Other common cam arrangements are cylindrical, spherical and contour cams. The cylindrical or barrel cam is in essence a cylinder with a cut groove for the follower to move along this path (Tesar and Matthew, 1976). The spherical cam resembles a ball-and-socket joint to some extent (Chakraborty and Dhande, 1977). Contour cams allow members to roll upon each other without sliding, and most often employ gears in mesh (Mabie and Reinholtz, 1987). Other cams that are still not as common are globoidal and conical cams, which are both three-dimensional cams, and radial and double-end cams. This thesis will only discuss cams of the planar disk type.
4.1.2 Cam Nomenclature

Figure 4.2 shows the basic nomenclature used to describe a typical cam mechanism. The base circle is the smallest circle located at the center of cam rotation that is still internally tangent to the cam surface. The trace point is a point on the follower that would correspond with the point of a knife-edge follower, and serves as a reference point for the follower displacement. For instance, the trace point of a roller follower is located at the center of the roller. The pitch curve is the path of the trace point around the cam contour. The stroke is the greatest distance through which the follower moves. The pressure angle is the angle between the direction of motion of the follower and the common normal to the cam and follower surface junction (Mabie and Reinholtz, 1987; Chen, 1982).

4.2 Follower Types

Once a specific application has been selected the type of cam and follower must be considered along with the desired motion. The most common disk cam followers are of the roller or flat-faced type. A roller follower interacts with the surface of the cam with primarily rolling contact, and although it can be more expensive, it is more frequently used in machinery. The flat-faced follower engages in both rolling and sliding contact, and is usually able to operate in a smaller workspace than its roller follower counterpart (Norton, 1992). Along with these two types of cam-follower interface contact types, the follower may move in either a translational or oscillatory motion. A translating follower is guided by supports alongside the follower stem and is frequently kept in contact with the cam by
Figure 4.2: Cam Nomenclature
a straight return spring. An oscillating follower has one end at a fixed pivot and has a constant force usually provided by a torsional spring at this pivot location.

These four main cam and follower arrangements are the only ones that are used in the Cam Design System, and are shown in Figure 4.3.

A knife-edge follower is a type of follower used in theory only because of the high contact stresses it would exhibit on the cam surface (Mabile and Reinholtz, 1987). Other less common types of followers are the conical, elliptical, spherical (essentially a three-dimensional roller) and cylindrical followers. These are all three-dimensional followers and will not be examined in this thesis.

### 4.3 Envelope Theory

The theory of envelopes states that a circle translating through a plane has tangents at any point in time. The series of these tangents taken together form bounding curves called envelopes. The envelope formed by these tangents form a conjugate curve pair, as shown in Figure 4.4. One curve or surface is the conjugate of the other curve or surface under constrained motion, and each is in contact with the envelope at all times (Boltyanskii, 1964; Voruganti, 1990). This condition of conjugacy depends on the shape of each of the surfaces and upon the relative motion between them.
Figure 4.3: Different Types of Cam and Follower Arrangements
Figure 4.4: Cam Profile Defined by an Envelope of Follower Positions
The theory of envelopes applies readily to the case of cam synthesis. The motion profile of the follower is the known curve in cam synthesis. The cam profile is the conjugate curve of the follower profile. For instance, the motion profile of the roller follower is created as an envelope of the cam surface by holding the follower fixed and moving the cam through its full rotation of positions. This can also be seen in Figure 4.4. This theory is used as the forerunner to the conjugate geometry theory in order to define the cam profile.

4.4 Conjugate Geometry Theory

The conjugate geometry theory is the cornerstone of the cam synthesis methods that are used in the Cam Design System software. The conjugate geometry theory is based on the contact between two surfaces, which are in this case the cam surface and the follower. Assuming that the contacting surfaces between the two bodies are both smooth and continuous, several fundamental laws governing this contact can be stated (Reinholtz, 1991):

- The line tangent to each surface must in fact be the same line at the contact point. In other words, a common tangent must exist.
- A common normal also exists which is perpendicular to the common tangent.
- Assuming no separation or impingement of the surfaces, i.e., rigid-body contact, no relative motion between the bodies can occur along the common normal. Therefore, relative motion occurs only along the line of common tangency.
• Assuming frictionless surfaces, force can only be transmitted along the common normal.

Once the follower arrangement and type of motion of the follower is fixed, the concepts above are used to synthesize the cam. In fact, the notion that relative motion occurs only along the common tangent is sufficient to synthesize any cam-and-follower system.

The method of inversion is used to view the cam attached to a fixed reference coordinate system. This allows one to define the cam profile as the inner envelope of follower positions that move around it. More specifically, the vector location of the contact point between the cam and the follower is given as \( \hat{P} \). This is shown in the translating flat-faced follower example arrangement in Figure 4.5. The normal vector is perpendicular to the common tangent line between the cam and the follower and is given as a unit vector \( \hat{N} \). Note that the common normal is the vector passing through the center of the follower (for a roller follower) or perpendicular to the follower face (for a flat-faced follower). In fact, it rarely falls along the radial line from the center of rotation of the cam. Knowing these vectors enables one to form a closed-loop vector equation between the cam and the follower. For each specific case of cam and follower, this one vector equation will yield a specific unknown crucial to the solution of the conjugate geometry problem. This will result in one final closed-form expression for the cam profile as a function of the rotation of the cam and other known parameters.

For the translating flat-faced follower and planar disk cam shown in Figure 4.5, for instance, this unknown is the overhung length of the flat-faced follower arm measured
Figure 4.5: Translating Flat-Faced Follower
from its stem to the point of contact with the cam. This value changes with each degree of rotation of the cam, but has now been solved for in terms of known values using the conjugate geometry theory. With the contact point known at all positions of rotation of the cam, the cam contour can be easily determined.

The beauty of conjugate geometry theory is that it can be readily implemented into an automated algorithm such as that contained in the Cam Design System. This is especially useful and important in the areas of design and manufacturing. With known cases such as the four cases of followers here already pre-programmed, the user must only specify the type of motion of the follower. Once the specific cam parameters have been identified, calculations may commence and the cam profile can be synthesized. Although these methods have been applied here only to planar disk cams, this method can also be applied to other types of cams such as those mentioned in Section 4.1.1 of this thesis. For example, this method can be applied to spatial cams by using common tangent planes instead of tangent lines.

4.4.1 Translating Flat-Faced Follower Case

Referring back to Figure 4.5, $\vec{R}$ is the radial vector from the center of the cam to the nearest point on the follower face and is along the direction of the follower stem. $\vec{I}$ is the vector location from the end of vector $\vec{R}$ to the point of contact with the cam along the direction of the face of the flat-faced follower. This vector represents the overhung length of the follower, or eccentricity. $\theta$ is the angular rotation of the cam, and $\psi$ is the angle
measured counterclockwise between the follower stem and the follower face. β is the oblique angle of the follower stem measured counterclockwise. In all cases, the oblique angle is measured to the vertical position of the follower stem, or

\[ \beta = \psi - \frac{\pi}{2} \]  

(4.1)

It is also seen that the magnitude of the radial vector \( \tilde{R} \) is the adjusted base circle radius plus the follower displacement, or

\[ R = \frac{c}{\sin(\pi - \beta)} + s(\theta) \]  

(4.2)

The vector location for the point of contact is found by

\[ \vec{P} = Re^{i\theta} + le^{i(\theta + \psi)} \]  

(4.3)

and the vector normal to the point of contact is

\[ \vec{N} = e^{i\left(\theta + \psi + \frac{\pi}{2}\right)} \]  

(4.4)

It should be kept in mind that vector \( \vec{N} \) is a unit vector in a direction perpendicular to the common tangent.

The intermediate result for the velocity of the point of contact with respect to the rotation of the cam, or the derivative of Eq. 4.3 with respect to \( \theta \), is

\[
\frac{d\vec{P}}{d\theta} = \frac{dR}{d\theta} e^{i\theta} + R \left( i \frac{d\theta}{d\theta} e^{i\theta} + \frac{dl}{d\theta} e^{i(\theta + \psi)} + \left[ i \frac{d\theta}{d\theta} + \frac{d\psi}{d\theta} \right] e^{i(\theta + \psi)} \right) \]  

(4.5)

\[ = R'e^{i\theta} + Rie^{i\theta} + l'e^{i(\theta + \psi)} + lie^{i(\theta + \psi)} \]
This vector $\frac{d\vec{P}}{d\theta}$ is always along the direction of the common tangent between the cam and the follower. So taking the dot product of Eq. 4.4 and Eq. 4.5 and setting the result equal to zero yields the condition of normality, or

$$\hat{N} \cdot \frac{d\vec{P}}{d\theta} = 0$$

$$= R' \cos\left(\theta + \psi + \frac{\pi}{2} - \theta\right) + R \cos\left(\theta + \psi + \frac{\pi}{2} - \theta - \frac{\pi}{2}\right) + l' \cos\left(\theta + \psi + \frac{\pi}{2} - \theta - \psi - \frac{\pi}{2}\right)$$

$$+ l \cos\left(\theta + \psi + \frac{\pi}{2} - \theta - \psi - \frac{\pi}{2}\right)$$

$$= R' \cos\left(\psi + \frac{\pi}{2}\right) + R \cos \psi + l' \cos\left(\frac{\pi}{2}\right) + l \cos(0)$$

$$= -R' \sin \psi + R \cos \psi + l$$

(4.6)

Solving this equation for the unknown yields the final conjugate geometry result for the overhung length of the flat-faced follower,

$$l = R' \sin \psi - R \cos \psi$$

(4.7)

In the case of a translating flat-faced follower, Eq. 4.7 is used in the synthesis to solve for the variable length of the follower face. Once this length is known the vector location of the contact point defined by the vector $\vec{P}$ between the cam and the follower can be found for any angle of rotation of the cam.
4.4.2 Oscillating Flat-Faced Follower Case

As shown in Figure 4.6, \( \tilde{C} \) is the vector from the center of the cam to the pivot location of the oscillating flat-faced follower. \( \tilde{I} \) is the vector location from the pivot point of the follower arm to the point of contact with the cam along the direction of the face of the flat-faced follower. This vector represents the variable length of the follower arm. \( \theta \) is the angular rotation of the cam. \( \phi \) represents the angle of oscillation of the follower arm and is the angle measured counterclockwise from the direction of \( \tilde{C} \) to the direction of \( \tilde{I} \).

The vector location for the point of contact is found by

\[
\tilde{P} = Ce^{i\theta} + le^{i(\theta + \phi)}
\]  

and the vector normal to the point of contact is

\[
\hat{N} = e^{i\left(\theta + \phi + \frac{\pi}{2}\right)}
\]  

The intermediate result for the velocity of the point of contact with respect to the rotation of the cam, or the derivative of Eq. 4.8 with respect to \( \theta \), is

\[
\frac{d\tilde{P}}{d\theta} = \frac{dC}{d\theta} e^{i\theta} + C\left(i\frac{d\theta}{d\theta} e^{i\theta} + \frac{dl}{d\theta} e^{i(\theta + \phi)} + il\left(\frac{d\theta}{d\theta} + \frac{d\phi}{d\theta}\right) e^{i(\theta + \phi)}\right)
\]  

\[
= Cie^{i\theta} + l'e^{i(\theta + \phi)} + li(1 + \phi')e^{i(\theta + \phi)}
\]  

Taking the dot product of Eq. 4.9 and Eq. 4.10 and setting the result equal to zero yields the condition of normality, or
Figure 4.6: Oscillating Flat-Faced Follower
\[ \hat{N} \cdot \frac{d\vec{P}}{d\theta} = 0 \]

\[ = C \cos\left(\theta + \phi + \frac{\pi}{2} - \theta - \frac{\pi}{2}\right) + l' \cos\left(\theta + \phi + \frac{\pi}{2} - \theta - \phi - \frac{\pi}{2}\right) \]

\[ + l(1 + \phi') \cos\left(\theta + \phi + \frac{\pi}{2} - \theta - \phi - \frac{\pi}{2}\right) \]

\[ = C \cos \phi + l' \cos\left(\frac{\pi}{2}\right) + l(1 + \phi') \cos(0) \]

\[ = C \cos \phi + l(1 + \phi') \]

(4.11)

Solving this equation for the unknown yields the final conjugate geometry result for the variable length of the arm of the flat-faced follower,

\[ l = \frac{-C \cos \phi}{(1 + \phi')} \]

(4.12)

In the case of an oscillating flat-faced follower, Eq. 4.12 is used in the synthesis to solve for the variable length of the follower arm. Once this length is known the vector location of the contact point defined by the vector \( \vec{P} \) between the cam and the follower can be found for any angle of rotation of the cam.
4.4.3 Translating Roller Follower Case

As shown in Figure 4.7, $\tilde{R}$ is the radial vector in the direction of the follower stem from the center of the cam to a point that forms a right angle with a line drawn to the center of the roller follower. $\tilde{d}$ is the vector location from the end of vector $\tilde{R}$ to the center of the roller follower and is normal to the direction of the radial vector $\tilde{R}$. This vector represents the offset distance of the follower stem from its alignment with the cam center location, and its magnitude is $d$, measured to the line of action of the follower stem. $\tilde{r}$ is the vector from the center of the roller follower to the contact point, and is always in the direction of the common normal. The magnitude of this vector is $r$, and is equal to the radius of the roller follower. $\theta$ is the angular rotation of the cam, and $\psi$ is the angle measured counterclockwise between the follower stem and the radius vector $\tilde{r}$.

It is also seen that the magnitude of the radial vector $\tilde{R}$ is the base circle radius plus the follower displacement plus the radius of the follower, minus a fraction of the offset distance, or

$$R = c + s(\theta) + r - \sqrt{2(c + s(\theta) + r)^2 \left[1 - \cos \left(\sin^{-1}\left(\frac{d}{c + s(\theta) + r}\right)\right)\right]} - d^2 \quad (4.13)$$

The vector location for the point of contact is found by

$$\vec{P} = Re^{i\theta} + de^{i\left(\frac{\theta + \psi}{2}\right)} + re^{i(\theta + \psi)} \quad (4.14)$$

and the vector normal to the point of contact is

$$\hat{N} = e^{i(\theta + \psi)} \quad (4.15)$$

The intermediate result for the velocity of the point of contact with respect to the rotation of the cam, or the derivative of Eq. 4.14 with respect to $\theta$, is
Figure 4.7: Translating Roller Follower
\[
\frac{d\tilde{P}}{d\theta} = \frac{dR}{d\theta} e^{i\theta} + R \left( i \frac{d}{d\theta} \right) e^{i\theta} + d \left( \frac{\pi}{2} \right) e^{i\left(\theta + \frac{\pi}{2}\right)} + d \left( \frac{\pi}{2} \right) e^{i\left(\theta + \frac{\pi}{2}\right)} + r \left( \frac{d\theta}{d\theta} + \frac{d\psi}{d\theta} \right) e^{i(\theta + \psi)}
\]

\[
= R e^{i\theta} + Rie^{i\theta} + die^{i\left(\theta + \frac{\pi}{2}\right)} + ri(1 + \psi')e^{i(\theta + \psi)}
\]

Taking the dot product of Eq. 4.15 and Eq. 4.16 and setting the result equal to zero yields the condition of normality, or

\[
\hat{N} \cdot \frac{d\tilde{P}}{d\theta} = 0
\]

\[
= R' \cos(\theta + \psi - \theta) + R \cos\left(\theta + \psi - \theta - \frac{\pi}{2}\right) + d \cos(\theta + \psi - \theta - \pi)
\]

\[
+ r(1 + \psi') \cos\left(\theta + \psi - \theta - \psi - \frac{\pi}{2}\right)
\]

\[
= R' \cos \psi + R \cos\left(\psi - \frac{\pi}{2}\right) + d \cos(\psi - \pi) + r(1 + \psi') \cos\left(\frac{\pi}{2}\right)
\]

\[
= R' \cos \psi + R \sin \psi - d \cos \psi
\]

\[
= R' \cos \psi + R \sin \psi
\]

\[
= (R' - d) \cos \psi + R \tan \psi
\]

\[
= (R' - d) + R \tan \psi
\]

Solving this equation for the unknown yields the final conjugate geometry result for the angle of rotation of the roller on the roller follower,

*Cam Design Theory*
\[
\tan \psi = \frac{d - R'}{R} \quad (4.18)
\]
or
\[
\psi = \tan^{-1} \left( \frac{d - R'}{R} \right) \quad (4.19)
\]

In the case of a translating roller follower, Eq. 4.19 is used in the synthesis to solve for the variable angle of rotation of the roller on the end of the follower. Once this angle is known the vector location of the contact point defined by the vector \( \vec{P} \) between the cam and the follower can be found for any angle of rotation of the cam.

### 4.4.4 Oscillating Roller Follower Case

As shown in Figure 4.8, \( \vec{C} \) is the vector from the center of the cam to the pivot location of the oscillating flat-faced follower. \( \vec{l} \) is the vector location from the pivot point of the follower arm to the center of the roller follower along the direction of the arm of the follower. \( \vec{r} \) is the vector from the center of the roller follower to the contact point, and is always in the direction of the common normal. The magnitude of this vector is \( r \), and is equal to the radius of the roller follower. \( \theta \) is the angular rotation of the cam. \( \phi \) represents the angle of oscillation of the follower arm and is the angle measured counterclockwise from the direction of \( \vec{C} \) to the direction of \( \vec{l} \). \( \psi \) is the angle of rotation of the roller follower and is measured counterclockwise between the follower arm and the radius vector \( \vec{r} \).
Figure 4.8: Oscillating Roller Follower
The vector location for the point of contact is found by
\[ \vec{P} = Ce^{i\theta} + le^{i(\theta + \phi)} + re^{i(\theta + \phi + \psi)} \]  
(4.20)

and the vector normal to the point of contact is
\[ \vec{N} = e^{i(\theta + \phi + \psi)} \]  
(4.21)

The intermediate result for the velocity of the point of contact with respect to the rotation of the cam, or the derivative of Eq. 4.20 with respect to \( \theta \), is
\[
\frac{d\vec{P}}{d\theta} = \frac{dC}{d\theta} e^{i\theta} + C\left( i \frac{d\theta}{d\theta} \right) e^{i\theta} + \frac{dl}{d\theta} e^{i(\theta + \phi)} + l\left[ i \left( \frac{d\theta}{d\theta} + \frac{d\phi}{d\theta} \right) \right] e^{i(\theta + \phi)} + \frac{dr}{d\theta} e^{i(\theta + \phi + \psi)} \\
+ r\left[ i \left( \frac{d\theta}{d\theta} + \frac{d\phi}{d\theta} + \frac{d\psi}{d\theta} \right) \right] e^{i(\theta + \phi + \psi)} \]  
(4.22)

\[ = Cie^{i\theta} + li(1 + \phi')e^{i(\theta + \phi)} + ri(1 + \phi' + \psi')e^{i(\theta + \phi + \psi)} \]

Taking the dot product of Eq. 4.21 and Eq. 4.22 and setting the result equal to zero yields the condition of normality, or

*Cam Design Theory* 40
\[ \hat{N} \cdot \frac{d\hat{p}}{d\theta} = 0 \]

\[ = C \cos \left( \theta + \phi + \psi - \theta - \frac{\pi}{2} \right) + l(1 + \phi') \cos \left( \theta + \phi + \psi - \theta - \phi - \frac{\pi}{2} \right) + r(1 + \phi' + \psi') \cos \left( \theta + \phi + \psi - \theta - \phi - \psi - \frac{\pi}{2} \right) \]

\[ = C \cos \left( \phi + \psi - \frac{\pi}{2} \right) + l(1 + \phi') \cos \left( \psi - \frac{\pi}{2} \right) + r(1 + \phi' + \psi') \cos \left( -\frac{\pi}{2} \right) \]

\[ = C \sin (\phi + \psi) + l(1 + \phi') \sin \psi \]

\[ = C(\sin \phi \cos \psi + \sin \psi \cos \phi) + l(1 + \phi') \sin \psi \]

\[ = C \sin \phi \cos \psi + C \sin \psi \cos \phi + l(1 + \phi') \sin \psi \]

\[ = C \sin \phi \cot \psi + C \cos \phi + l(1 + \phi') \]

Solving this equation for the unknown yields the final conjugate geometry result for the angle of rotation of the roller on the roller follower,

\[ \cot \psi = \frac{-\left[ C \cos \phi + l(1 + \phi') \right]}{C \sin \phi} \]

(4.24)

or

\[ \psi = \tan^{-1} \left\{ \frac{C \sin \phi}{-l[C \cos \phi + l(1 + \phi')] \} \right\} \]

(4.25)

In the case of an oscillating roller follower, Eq. 4.25 is used in the synthesis to solve for the variable angle of rotation of the roller on the end of the follower. Once this angle is
known the vector location of the contact point defined by the vector $\vec{P}$ between the cam and the follower can be found for any angle of rotation of the cam.

4.5 Implications of Conjugate Geometry Results

In specifying the output motion of the follower, one is concerned with selecting motion curves to increase the smoothness at rise and fall intervals of follower motion, minimizing accelerations and forces, and increasing the overall continuity of the graphs. Aside from these concerns, other mechanical problems in the cam-and-follower system such as excessive vibration and bending moment can be readily taken into account through the use of the conjugate geometry results.

As an example, consider the translating flat-faced follower configuration shown in Figure 4.9. Referring also to the conjugate geometry derivation for the translating flat-faced follower shown in Eq. 4.7 and Figure 4.5, the final conjugate geometry result for this cam and follower system is

$$l = R' \sin \psi - R \cos \psi$$

(4.26)

Since this particular follower is not oblique, the angle $\psi$ of the follower stem with respect to the angle of the flat-face is a right angle, and this equation reduces simply to

$$l = R' = \frac{dR}{d\theta}$$

(4.27)
The bending moment in the follower stem is

\[ M = Fl \]  \hspace{1cm} (4.28)

where \( F \) is the force in the follower stem and \( l \) is the moment arm.

In this case, Eq. 4.27 indicates that the moment arm is always equal to the angular velocity of the cam, \( R' \). Note that \( R' \) is expressed in units of length per radian, giving consistent dimensions. In this way, the maximum bending moment is directly proportional to the maximum velocity encountered by the cam. Since this result comes directly from the conjugate geometry analysis, the place where the cam encounters a maximum value of \( l \) will also be the place where it has an extreme (maximum or minimum) velocity. Therefore, this extreme moment arm length can be combined with the maximum (or minimum) force in Eq. 4.28 so that the maximum (or minimum) bending moment can be
found. Hence, it is useful to analyze the conjugate geometry results in this and other cases in order to attempt to minimize the bending moment encountered by the follower stem.

Aside from bending moment characteristics, problems with mechanical vibration of a system can be predicted and minimized in much the same manner. Vibration is induced by impact and by cyclic loading, i.e., the harmonic characteristics of a system. There is also a desire to minimize these harmonic characteristics of the system, since this will also tend to minimize the amount of mechanical vibration in the system.

Furthermore, in any follower system there is always a desire to reduce the force experienced by the follower arm or stem. The force $F$ is denoted by

$$ F = ma $$

(4.29)

So in this sense minimizing the acceleration endured by the cam will minimize the force on the follower. Minimizing the force on the follower will also help to reduce the bending moment, as seen in Eq. 4.28. This is another facet of the design that needs to be taken into account when selecting motion curve types for the follower motion, some of which have lower peak accelerations than others.
5. Cam Motion Specifications

In creating a cam profile, one of the most important aspects of its design is precisely how the follower is to move throughout the cam rotation. When a rise or fall of the follower is specified, the follower must move a specific distance in a specific time. The exact motion of the follower, i.e., its displacement, velocity, acceleration and other motion characteristics in higher derivatives is critical to the optimal performance of the system.

A number of follower motion curves have been developed to produce cams for a variety of applications. Even with all of these motion curves, it is difficult to design a cam that meets these criteria exactly and is optimal in all respects. Usually the cam follower is restricted to impart certain displacements at specified cam angles. Once the displacement criteria is met, other parameters such as velocity, acceleration and force are taken into account and optimized in the final cam design. In reality, any high speed cam must be continuous through its displacement, velocity and acceleration in order for it to be a feasible design. It is primarily these other design criteria that make many of the characteristics of the different motion curves preferred for specific applications over others.

There are infinitely many motion curves that can be chosen for the follower. For instance, there are simple motions like the constant-velocity/ramp function and the constant acceleration function. There are also many sinusoidal-type curves that result in a smooth, continuous motion throughout all derivatives. There are also many possibilities of

Cam Motion Specifications 45
combining curves to get the best characteristics of each separate curve. In addition, there are many different polynomial functions that can be tailored to suit the specific boundary conditions needed for a particular cam design. Two such polynomial functions have been included in this thesis.

Nine different functions are explained below that are available in the Cam Design System. Other popular functions such as the constant acceleration function, the 3-4 polynomial function, the polydyne function, the double harmonic function, and some cubic and parabolic functions to name a few, have not yet been included in the Cam Design System. These could be readily implemented, however. The Cam Design System also allows the user to designate a custom motion program, i.e., to use any combination of rise, fall and dwell motions of the follower. Three more common motion profiles, Rise-Fall, Rise-Fall-Dwell, and Rise-Dwell-Fall-Dwell, have been included for convenience. Many other custom motions, e.g., a double-dwell cam design, could be easily input by the user segment by segment.

Motion equations have been included for the aforementioned nine curve types. The motion equations for the fall (or return) of a follower can easily be derived from the rise equations. As a generic example, suppose the displacement function for the rise function of a follower is

\[ s(\theta) = f(\theta, h, \beta) \]  \hspace{1cm} (5.1)

The displacement equation for the fall portion of the follower motion will then be

\[ s(\theta) = h - f(\theta, h, \beta) \]  \hspace{1cm} (5.2)
The fall equations for velocity, acceleration and jerk can be found by successively differentiating the fall displacement equation. For the purposes of this thesis, all fall equations have been derived from the rise equations using the relationships shown in Eq. 5.1 and Eq. 5.2. Both sets of rise and fall motion equations have been included here for convenience.

For all the cam motion curves described below, \( h \) represents the stroke length for that specific interval. An interval is a portion of the cam rotation where the follower motion is classified as either a rise, fall or dwell. The angle \( \beta \) represents the total cam angle during a specific interval, or the period of that interval. The angle \( \theta \) represents the total angle of rotation of the cam. The ratio \( \frac{\theta}{\beta} \) is a normalized dimensionless ratio of angles that ranges in value from 0 to 1. S, v, a and j represent displacement, velocity, acceleration and jerk, respectively, and are each a function of the camshaft angle \( \theta \), as well as the stroke length \( h \) and the cam angle \( \beta \). Velocity is the first derivative of displacement, acceleration is its second derivative, and the jerk function is its third derivative, or the rate of change of acceleration.

### 5.1 Basic Functions

These functions presented in this section are in moderate use in industry. These functions presented here are useful in theory. However, they might bring about some problems when applied in a real-life cam-and-follower system. Despite this, these functions serve as a foundation to other more complex and more useful functions presented in later sections.
5.1.1 Ramp Function

The ramp function, or the constant-velocity function, provides a constant velocity for a portion of the cam rise or fall, thus giving a zero acceleration and zero jerk.

For a follower during the rise segment of the motion profile, the following equations apply (Jensen, 1987):

\[ s(\theta) = h \left( \frac{\theta}{\beta} \right) \]  \hspace{1cm} (5.3)

\[ v(\theta) = \frac{h}{\beta} \]  \hspace{1cm} (5.4)

\[ a(\theta) = 0 \]  \hspace{1cm} (5.5)

\[ j(\theta) = 0 \]  \hspace{1cm} (5.6)

For a follower during the fall segment of the motion profile, the following equations apply:

\[ s(\theta) = h \left[ 1 - \left( \frac{\theta}{\beta} \right) \right] \]  \hspace{1cm} (5.7)

\[ v(\theta) = -\frac{h}{\beta} \]  \hspace{1cm} (5.8)

\[ a(\theta) = 0 \]  \hspace{1cm} (5.9)

\[ j(\theta) = 0 \]  \hspace{1cm} (5.10)

All motion characteristics for the ramp function are shown in Figure 5.1.
Figure 5.1: Motion Characteristics for the Ramp Function

5.1.2 Trapezoidal Acceleration Function

The trapezoidal acceleration function is essentially a square wave with the corners 'knocked off,' and is in practice of little use. The jerk function for this motion curve turns out to be a square wave, which is very discontinuous at the intervals of follower motion. Jerk functions such as these tend to bring about excessive vibrations in the follower train.
(Norton, 1992). This curve is also good for high-speed applications, but might be difficult to machine (Rothbart, 1985).

For a follower during the rise segment of the motion profile, the following equations apply (Molian, 1968):

For $0 \leq \theta < \frac{1}{8} \beta$:

$$s(\theta) = \frac{64}{9} h \left( \frac{\theta}{\beta} \right)^3$$  \hspace{1cm} (5.11)

$$v(\theta) = \frac{64}{3} h \left( \frac{\theta}{\beta} \right)^2$$  \hspace{1cm} (5.12)

$$a(\theta) = \frac{128}{3} h \left( \frac{\theta}{\beta^2} \right)$$  \hspace{1cm} (5.13)

$$j(\theta) = \frac{128}{3} \frac{h}{\beta^3}$$  \hspace{1cm} (5.14)

For $\frac{1}{8} \beta \leq \theta < \frac{3}{8} \beta$:

$$s(\theta) = h \left[ \frac{8}{3} \left( \frac{\theta}{\beta} \right)^2 - \frac{1}{3} \left( \frac{\theta}{\beta} \right) + \frac{1}{72} \right]$$  \hspace{1cm} (5.15)

$$v(\theta) = \frac{h}{\beta} \left[ \frac{16}{3} \left( \frac{\theta}{\beta} \right)^2 - \frac{1}{3} \right]$$  \hspace{1cm} (5.16)

$$a(\theta) = \frac{16}{3} \frac{h}{\beta^2}$$  \hspace{1cm} (5.17)

$$j(\theta) = 0$$  \hspace{1cm} (5.18)

For $\frac{3}{8} \beta \leq \theta < \frac{5}{8} \beta$:
\[ s(\theta) = h \left[ - \frac{64}{9} \left( \frac{\theta}{\beta} \right)^3 + \frac{32}{3} \left( \frac{\theta}{\beta} \right)^2 - \frac{10}{3} \left( \frac{\theta}{\beta} \right) + \frac{7}{18} \right] \]  

(5.19)

\[ v(\theta) = \frac{h}{\beta} \left[ - \frac{64}{3} \left( \frac{\theta}{\beta} \right)^2 + \frac{64}{3} \left( \frac{\theta}{\beta} \right) - \frac{10}{3} \right] \]  

(5.20)

\[ a(\theta) = \frac{h}{\beta^2} \left[ - \frac{128}{3} \left( \frac{\theta}{\beta} \right) + \frac{64}{3} \right] \]  

(5.21)

\[ j(\theta) = - \frac{128}{3} \frac{h}{\beta^3} \]  

(5.22)

For \( \left( \frac{5}{8} \beta \leq \theta < \frac{7}{8} \beta \right) \):

\[ s(\theta) = h \left[ - \frac{8}{3} \left( \frac{\theta}{\beta} \right)^2 + 5 \left( \frac{\theta}{\beta} \right) - \frac{97}{72} \right] \]  

(5.23)

\[ v(\theta) = \frac{h}{\beta} \left[ - \frac{16}{3} \left( \frac{\theta}{\beta} \right) + 5 \right] \]  

(5.24)

\[ a(\theta) = - \frac{16}{3} \frac{h}{\beta^2} \]  

(5.25)

\[ j(\theta) = 0 \]  

(5.26)

For \( \left( \frac{7}{8} \beta \leq \theta \leq \beta \right) \):

\[ s(\theta) = h \left[ \frac{64}{9} \left( \frac{\theta}{\beta} \right)^3 - \frac{64}{3} \left( \frac{\theta}{\beta} \right)^2 + \frac{64}{3} \left( \frac{\theta}{\beta} \right) - \frac{55}{9} \right] \]  

(5.27)

\[ v(\theta) = \frac{h}{\beta} \left[ \frac{64}{3} \left( \frac{\theta}{\beta} \right)^2 - \frac{128}{3} \left( \frac{\theta}{\beta} \right) + \frac{64}{3} \right] \]  

(5.28)
\[ a(\theta) = \frac{h}{\beta^2} \left[ \frac{128}{3} \left( \frac{\theta}{\beta} \right) - \frac{128}{3} \right] \]  \hspace{1cm} (5.29)

\[ j(\theta) = \frac{128}{3} \frac{h}{\beta^3} \]  \hspace{1cm} (5.30)

For a follower during the fall segment of the motion profile, the following equations apply:

For \( 0 \leq \theta < \frac{1}{8} \beta \):

\[ s(\theta) = h \left[ 1 - \frac{64}{\beta^2} \left( \frac{\theta}{\beta} \right)^3 \right] \]  \hspace{1cm} (5.31)

\[ v(\theta) = -\frac{64}{3} \frac{h}{\beta^2} \left( \frac{\theta}{\beta} \right)^2 \]  \hspace{1cm} (5.32)

\[ a(\theta) = -\frac{128}{3} \frac{h}{\beta^2} \left( \frac{\theta}{\beta} \right) \]  \hspace{1cm} (5.33)

\[ j(\theta) = -\frac{128}{3} \frac{h}{\beta^3} \]  \hspace{1cm} (5.34)

For \( \frac{1}{8} \beta \leq \theta < \frac{3}{8} \beta \):

\[ s(\theta) = h \left[ -\frac{8}{3} \left( \frac{\theta}{\beta} \right)^2 + \frac{1}{3} \left( \frac{\theta}{\beta} \right) + \frac{71}{72} \right] \]  \hspace{1cm} (5.35)

\[ v(\theta) = \frac{h}{\beta} \left[ -\frac{16}{3} \left( \frac{\theta}{\beta} \right) + \frac{1}{3} \right] \]  \hspace{1cm} (5.36)

\[ a(\theta) = -\frac{16}{3} \frac{h}{\beta^2} \]  \hspace{1cm} (5.37)

\[ j(\theta) = 0 \]  \hspace{1cm} (5.38)
For \( \left( \frac{3}{8} \beta \leq \theta < \frac{5}{8} \beta \right) \):

\[
s(\theta) = h \left[ \frac{64}{9} \left( \frac{\theta}{\beta} \right)^3 - \frac{32}{3} \left( \frac{\theta}{\beta} \right)^2 + \frac{10}{3} \left( \frac{\theta}{\beta} \right) + \frac{11}{18} \right]
\]

(5.39)

\[
\nu(\theta) = \frac{h}{\beta} \left[ \frac{64}{3} \left( \frac{\theta}{\beta} \right)^2 - \frac{64}{3} \left( \frac{\theta}{\beta} \right) + \frac{10}{3} \right]
\]

(5.40)

\[
a(\theta) = \frac{h}{\beta^2} \left[ \frac{128}{3} \left( \frac{\theta}{\beta} \right) - \frac{64}{3} \right]
\]

(5.41)

\[
j(\theta) = \frac{128}{3} \frac{h}{\beta^3}
\]

(5.42)

For \( \left( \frac{5}{8} \beta \leq \theta < \frac{7}{8} \beta \right) \):

\[
s(\theta) = h \left[ \frac{8}{3} \left( \frac{\theta}{\beta} \right)^2 - 5 \left( \frac{\theta}{\beta} \right) + \frac{169}{72} \right]
\]

(5.43)

\[
\nu(\theta) = \frac{h}{\beta} \left[ \frac{16}{3} \left( \frac{\theta}{\beta} \right) - 5 \right]
\]

(5.44)

\[
a(\theta) = \frac{16}{3} \frac{h}{\beta^2}
\]

(5.45)

\[
j(\theta) = 0
\]

(5.46)

For \( \left( \frac{7}{8} \beta \leq \theta \leq \beta \right) \):

\[
s(\theta) = h \left[ -\frac{64}{9} \left( \frac{\theta}{\beta} \right)^3 + \frac{64}{3} \left( \frac{\theta}{\beta} \right)^2 - \frac{64}{3} \left( \frac{\theta}{\beta} \right) + \frac{64}{9} \right]
\]

(5.47)
\[
\nu(\theta) = \frac{h}{\beta} \left[ -\frac{64}{3} \left( \frac{\theta}{\beta} \right)^2 + \frac{128}{3} \left( \frac{\theta}{\beta} \right) - \frac{64}{3} \right] \\
\]

\[
a(\theta) = \frac{h}{\beta^2} \left[ -\frac{128}{3} \left( \frac{\theta}{\beta} \right) + \frac{128}{3} \right] \\
\]

\[
j(\theta) = -\frac{128}{3} \frac{h}{\beta^3} \\
\]

Motion characteristics for the trapezoidal acceleration function are shown in Figure 5.2.

5.2 Trigonometric Functions

These functions are the most versatile of all the motion curves. They are composed of a series of sine and cosine functions, which makes these trigonometric functions continuous throughout all derivatives. They often tend to have low values of peak acceleration. These facets of the trigonometric functions alone make the design of cams with these functions widespread.
Figure 5.2: Motion Characteristics for the Trapezoidal Acceleration Function

5.2.1 Simple Harmonic Motion Function

This curve is the most basic of all the trigonometric functions. The simple harmonic motion function provides a smooth, continuous motion in all derivatives. It also provides a low value of peak acceleration and a relatively low pressure angle.
For a follower during the rise segment of the motion profile, the following equations apply (Norton, 1992):

\[ s(\theta) = \frac{h}{2} \left[ 1 - \cos \left( \frac{\pi \theta}{\beta} \right) \right] \quad (5.51) \]

\[ v(\theta) = \frac{\pi h}{\beta} \sin \left( \frac{\pi \theta}{\beta} \right) \quad (5.52) \]

\[ a(\theta) = \frac{\pi^2 h}{\beta^2} \cos \left( \frac{\pi \theta}{\beta} \right) \quad (5.53) \]

\[ j(\theta) = -\frac{\pi^3 h}{\beta^3} \sin \left( \frac{\pi \theta}{\beta} \right) \quad (5.54) \]

For a follower during the fall segment of the motion profile, the following equations apply:

\[ s(\theta) = \frac{h}{2} \left[ 1 + \cos \left( \frac{\pi \theta}{\beta} \right) \right] \quad (5.55) \]

\[ v(\theta) = -\frac{\pi h}{\beta} \sin \left( \frac{\pi \theta}{\beta} \right) \quad (5.56) \]

\[ a(\theta) = -\frac{\pi^2 h}{\beta^2} \cos \left( \frac{\pi \theta}{\beta} \right) \quad (5.57) \]

\[ j(\theta) = \frac{\pi^3 h}{\beta^3} \sin \left( \frac{\pi \theta}{\beta} \right) \quad (5.58) \]

All motion characteristics for the simple harmonic motion function are shown in Figure 5.3.
5.2.2 Cycloidal Displacement Function

The cycloidal motions are formed from the path made by a point on the circumference of a circle formed when rolled along a straight line. These curves are similar in shape to the simple harmonic motion curves in that they are both made from sine and cosine functions. In this case however, both the acceleration and jerk functions complete a full cycle. Also,
the acceleration curve is constrained to zero and both ends. Because of this the function can be coupled to a dwell at each end of the interval. This function is also excellent for use in high-speed applications (Rothbart, 1985). In addition, two cycloidal functions should not be used consecutively, since the pressure angle is relatively high and the accelerations returns to zero unnecessarily (Mabie and Reinholtz, 1987).

It should be noted that this function is a full cycloidal displacement function, as opposed to the half cycloidal function discussed in the following section. In reality these functions only differ slightly in their equations, but their characteristics are quite similar.

For a follower during the rise segment of the motion profile, the following equations apply (Norton, 1992):

\[ s(\theta) = h \left[ \frac{\theta}{\beta} - \frac{1}{2\pi} \sin \left( 2\pi \frac{\theta}{\beta} \right) \right] \]  \hspace{1cm} (5.59)

\[ v(\theta) = \frac{h}{\beta} \left[ 1 - \cos \left( 2\pi \frac{\theta}{\beta} \right) \right] \]  \hspace{1cm} (5.60)

\[ a(\theta) = \frac{2\pi h}{\beta^2} \sin \left( 2\pi \frac{\theta}{\beta} \right) \]  \hspace{1cm} (5.61)

\[ j(\theta) = \frac{4\pi^2 h}{\beta^3} \cos \left( 2\pi \frac{\theta}{\beta} \right) \]  \hspace{1cm} (5.62)

For a follower during the fall segment of the motion profile, the following equations apply:

\[ s(\theta) = h \left[ 1 - \left( \frac{\theta}{\beta} \right) + \frac{1}{2\pi} \sin \left( 2\pi \frac{\theta}{\beta} \right) \right] \]  \hspace{1cm} (5.63)
\[ v(\theta) = -\frac{h}{\beta} \left[ 1 - \cos \left( \frac{2\pi \theta}{\beta} \right) \right] \]  \hspace{1cm} (5.64)

\[ a(\theta) = -\frac{2\pi h}{\beta^2} \sin \left( \frac{2\pi \theta}{\beta} \right) \]  \hspace{1cm} (5.65)

\[ j(\theta) = -\frac{4\pi^2 h}{\beta^3} \cos \left( \frac{2\pi \theta}{\beta} \right) \]  \hspace{1cm} (5.66)

All motion characteristics for the cycloidal displacement function are shown in Figure 5.4.

### 5.2.3 Half Cycloidal Function

The half cycloidal function is very closely related to the cycloidal displacement function. In a nutshell, the half cycloidal function gives the user more control in piecing together curves. This function acts upon only half the period compared to that of the cycloidal displacement function described above in Section 5.2.2. Because of this, the velocity curve for the half cycloidal function is not constrained to zero at both ends of the interval, but rather only at one end. This aspect of the function makes it ideal for use either before or after a ramp function for instance.

The Cam Design System has implemented the half cycloidal function in essentially two parts. All of the following equations have been programmed into the design system. Once a half cycloidal function is chosen, the program will chose the proper rise or fall portion based on the value of the beginning velocity. If the beginning velocity for the interval is
zero, then Eq. 5.67 through Eq. 5.70 will be used for a rise portion of the follower motion. This is commonly called a C-1 cycloidal curve (Mabie and Reinholtz, 1987). If the beginning velocity for the interval in non-zero, then Eq. 5.71 through 5.74 will be used for the rise portion. This is commonly called a C-2 curve. Though each of these curves have different motion characteristics, their functions differ only slightly.
For a follower during the rise segment of the motion profile, the following equations apply
(Mabie and Reinholtz, 1987):

For a zero starting velocity (C-1 curve),

\[
s(\theta) = h \left[ \left( \frac{\theta}{\beta} \right) - \frac{1}{\pi} \sin \left( \frac{\pi \theta}{\beta} \right) \right] \tag{5.67}
\]

\[
v(\theta) = \frac{h}{\beta} \left[ 1 - \cos \left( \frac{\pi \theta}{\beta} \right) \right] \tag{5.68}
\]

\[
a(\theta) = \frac{\pi h}{\beta^2} \sin \left( \frac{\pi \theta}{\beta} \right) \tag{5.69}
\]

\[
j(\theta) = \frac{\pi^2 h}{\beta^3} \cos \left( \frac{\pi \theta}{\beta} \right) \tag{5.70}
\]

For a non-zero starting velocity (C-2 curve),

\[
s(\theta) = h \left[ \left( \frac{\theta}{\beta} \right) + \frac{1}{\pi} \sin \left( \frac{\pi \theta}{\beta} \right) \right] \tag{5.71}
\]

\[
v(\theta) = \frac{h}{\beta} \left[ 1 + \cos \left( \frac{\pi \theta}{\beta} \right) \right] \tag{5.72}
\]

\[
a(\theta) = -\frac{\pi h}{\beta^2} \sin \left( \frac{\pi \theta}{\beta} \right) \tag{5.73}
\]

\[
j(\theta) = -\frac{\pi^2 h}{\beta^3} \cos \left( \frac{\pi \theta}{\beta} \right) \tag{5.74}
\]

For a follower during the fall segment of the motion profile, the following equations apply:
For a zero starting velocity (C-3 curve),

\[ s(\theta) = h \left[ 1 - \left( \frac{\theta}{\beta} \right) + \frac{1}{\pi} \sin \left( \frac{\pi \theta}{\beta} \right) \right] \]  
(5.75)

\[ v(\theta) = -\frac{h}{\beta} \left[ 1 - \cos \left( \frac{\pi \theta}{\beta} \right) \right] \]  
(5.76)

\[ a(\theta) = -\frac{\pi h}{\beta^2} \sin \left( \frac{\pi \theta}{\beta} \right) \]  
(5.77)

\[ j(\theta) = -\frac{\pi^2 h}{\beta^3} \cos \left( \frac{\pi \theta}{\beta} \right) \]  
(5.78)

For a non-zero starting velocity (C-4 curve),

\[ s(\theta) = h \left[ 1 - \left( \frac{\theta}{\beta} \right) - \frac{1}{\pi} \sin \left( \frac{\pi \theta}{\beta} \right) \right] \]  
(5.79)

\[ v(\theta) = -\frac{h}{\beta} \left[ 1 + \cos \left( \frac{\pi \theta}{\beta} \right) \right] \]  
(5.80)

\[ a(\theta) = \frac{\pi h}{\beta^2} \sin \left( \frac{\pi \theta}{\beta} \right) \]  
(5.81)

\[ j(\theta) = \frac{\pi^2 h}{\beta^3} \cos \left( \frac{\pi \theta}{\beta} \right) \]  
(5.82)

Motion characteristics for a half cycloidal function with a zero starting velocity are shown in Figure 5.5.
Figure 5.5: Motion Characteristics for the Half Cycloidal Function

5.3 Modified Functions

The previous five curves are best suited for the conventional rise-dwell-fall sequence. The functions presented in this section however are made for use typically in high-speed applications. But situations regularly arise when a high-speed and high-precision cam is
required, and the aforementioned curves exhibit only a few adequate dynamic characteristics (Erdman and Sandor, 1984). The following curves better serve to provide the user with these dynamic characteristics at high speeds.

### 5.3.1 Modified Sine Function

The modified sine function combines two sinusoidal curves of different frequency to get a smoother curve with a moderately lower peak acceleration value. It also provides for a lower peak velocity and an overall smoother jerk curve.

For a follower during the rise segment of the motion profile, the following equations apply (Norton, 1992):

For \( 0 \leq \theta < \frac{1}{8} \beta \):

\[
\begin{align*}
 s(\theta) &= h \left[ 0.43990085 \left( \frac{\theta}{\beta} \right) - 0.0350062 \sin \left( 4\pi \frac{\theta}{\beta} \right) \right] \quad (5.83) \\
 v(\theta) &= 0.43990085 \frac{h}{\beta} \left[ 1 - \cos \left( 4\pi \frac{\theta}{\beta} \right) \right] \quad (5.84) \\
 a(\theta) &= 5.5279571 \frac{h}{\beta^2} \sin \left( 4\pi \frac{\theta}{\beta} \right) \quad (5.85) \\
 j(\theta) &= 69.4663577 \frac{h}{\beta^3} \cos \left( 4\pi \frac{\theta}{\beta} \right) \quad (5.86)
\end{align*}
\]
For \( \frac{1}{8} \beta \leq \theta < \frac{7}{8} \beta \):

\[
s(\theta) = h \left\{ 0.28004957 + 0.43990085 \left( \frac{\theta}{\beta} \right) - 0.31505577 \cos \left[ \left( \frac{4}{3} \frac{\pi}{\beta} \right) - \frac{\pi}{6} \right] \right\} \tag{5.87}
\]

\[
v(\theta) = 0.43990085 \frac{h}{\beta} \left\{ 1 + 3 \sin \left[ \left( \frac{4}{3} \frac{\pi}{\beta} \right) - \frac{\pi}{6} \right] \right\} \tag{5.88}
\]

\[
a(\theta) = 5.5279571 \frac{h}{\beta^2} \cos \left[ \left( \frac{4}{3} \frac{\pi}{\beta} \right) - \frac{\pi}{6} \right] \tag{5.89}
\]

\[
j(\theta) = -23.1553 \frac{h}{\beta^3} \sin \left[ \left( \frac{4}{3} \frac{\pi}{\beta} \right) - \frac{\pi}{6} \right] \tag{5.90}
\]

For \( \frac{7}{8} \beta \leq \theta \leq \beta \):

\[
s(\theta) = h \left\{ 0.56009915 + 0.43990085 \left( \frac{\theta}{\beta} \right) - 0.0350062 \sin \left[ 2\pi \left( \frac{2}{\beta} - 1 \right) \right] \right\} \tag{5.91}
\]

\[
v(\theta) = 0.43990085 \frac{h}{\beta} \left\{ 1 - \cos \left[ 2\pi \left( \frac{2}{\beta} - 1 \right) \right] \right\} \tag{5.92}
\]

\[
a(\theta) = 5.5279571 \frac{h}{\beta^2} \sin \left[ 2\pi \left( \frac{2}{\beta} - 1 \right) \right] \tag{5.93}
\]

\[
j(\theta) = 69.4663577 \frac{h}{\beta^3} \cos \left[ 2\pi \left( \frac{2}{\beta} - 1 \right) \right] \tag{5.94}
\]

For a follower during the fall segment of the motion profile, the following equations apply:

For \( 0 \leq \theta < \frac{1}{8} \beta \):
\[ s(\theta) = h \left[ 1 - 0.43990085 \left( \frac{\theta}{\beta} \right) + 0.0350062 \sin \left( 4\pi \frac{\theta}{\beta} \right) \right] \]  
(5.95)

\[ v(\theta) = -0.43990085 \frac{h}{\beta} \left[ 1 - \cos \left( \frac{4\pi \theta}{\beta} \right) \right] \]  
(5.96)

\[ a(\theta) = -5.5279571 \frac{h}{\beta^2} \sin \left( \frac{4\pi \theta}{\beta} \right) \]  
(5.97)

\[ j(\theta) = -69.4663577 \frac{h}{\beta^3} \cos \left( \frac{4\pi \theta}{\beta} \right) \]  
(5.98)

For \( \left( \frac{1}{8} \beta \leq \theta < \frac{7}{8} \beta \right) \):

\[ s(\theta) = h \left[ 0.71995043 - 0.43990085 \left( \frac{\theta}{\beta} \right) + 0.31505577 \cos \left( \frac{4\pi \theta}{3 \beta} - \frac{\pi}{6} \right) \right] \]  
(5.99)

\[ v(\theta) = -0.43990085 \frac{h}{\beta} \left[ 1 + 3 \sin \left( \frac{4\pi \theta}{3 \beta} - \frac{\pi}{6} \right) \right] \]  
(5.100)

\[ a(\theta) = -5.5279571 \frac{h}{\beta^2} \cos \left( \frac{4\pi \theta}{3 \beta} - \frac{\pi}{6} \right) \]  
(5.101)

\[ j(\theta) = 23.1553 \frac{h}{\beta^3} \sin \left( \frac{4\pi \theta}{3 \beta} - \frac{\pi}{6} \right) \]  
(5.102)

For \( \left( \frac{7}{8} \beta \leq \theta \leq \beta \right) \):

\[ s(\theta) = h \left[ 0.43990085 - 0.43990085 \left( \frac{\theta}{\beta} \right) + 0.0350062 \sin \left( 2\pi \left( \frac{2\theta}{\beta} - 1 \right) \right) \right] \]  
(5.103)

\[ v(\theta) = -0.43990085 \frac{h}{\beta} \left[ 1 - \cos \left( 2\pi \left( \frac{2\theta}{\beta} - 1 \right) \right) \right] \]  
(5.104)
\[ a(\theta) = -5.5279571 \frac{h}{\beta^2} \sin \left[ 2\pi \left( \frac{\theta}{\beta} - 1 \right) \right] \] (5.105)

\[ j(\theta) = -69.4663577 \frac{h}{\beta^3} \cos \left[ 2\pi \left( \frac{\theta}{\beta} - 1 \right) \right] \] (5.106)

All motion characteristics for the modified sine function are shown in Figure 5.6.

**Figure 5.6**: Motion Characteristics for the Modified Sine Function
5.3.2 Modified Trapezoidal Function

The modified trapezoidal function was designed to minimize extreme accelerations. It combines a pieces of a sinusoidal acceleration curve with a trapezoidal acceleration function to get a smoother curve at the corners of each interval. This makes for smoother transitions at the beginning and the end of intervals throughout all derivatives, and provides for a relatively low peak acceleration value. In fact, its excellent vibration characteristics and easier fabrication qualities tend to make this function just as effective as the cycloidal function (Rothbart, 1985).

For a follower during the rise segment of the motion profile, the following equations apply (Norton, 1992):

For \(0 \leq \theta < \frac{1}{8} \beta\):

\[
s(\theta) = h \left[ 0.38898448 \left( \frac{\theta}{\beta} \right) - 0.0309544 \sin \left( 4\pi \frac{\theta}{\beta} \right) \right] \quad (5.107)
\]

\[
\nu(\theta) = 0.38898448 \frac{h}{\beta} \left[ 1 - \cos \left( 4\pi \frac{\theta}{\beta} \right) \right] \quad (5.108)
\]

\[
a(\theta) = 4.888124 \frac{h}{\beta^2} \sin \left( 4\pi \frac{\theta}{\beta} \right) \quad (5.109)
\]

\[
j(\theta) = 61.425769 \frac{h}{\beta^3} \cos \left( 4\pi \frac{\theta}{\beta} \right) \quad (5.110)
\]

For \(\frac{1}{8} \beta \leq \theta < \frac{3}{8} \beta\):

Cam Motion Specifications

68
\[ s(\theta) = h \left[ 2.44406184 \left( \frac{\theta}{\beta} \right)^2 - 0.22203097 \left( \frac{\theta}{\beta} \right) + 0.00723407 \right] \]  
(5.111)

\[ v(\theta) = \frac{h}{\beta} \left[ 4.888124 \left( \frac{\theta}{\beta} \right) - 0.22203097 \right] \]  
(5.112)

\[ a(\theta) = 4.888124 \frac{h}{\beta^3} \]  
(5.113)

\[ j(\theta) = 0 \]  
(5.114)

For \( \left( \frac{3}{8} \beta \leq \theta \leq \frac{5}{8} \beta \right) \):

\[ s(\theta) = h \left[ 1.6110154 \left( \frac{\theta}{\beta} \right) - 0.0309544 \sin \left( 4\pi \frac{\theta}{\beta} - \pi \right) - 0.3055077 \right] \]  
(5.115)

\[ v(\theta) = \frac{h}{\beta} \left[ 1.6110154 - 0.38898448 \cos \left( 4\pi \frac{\theta}{\beta} - \pi \right) \right] \]  
(5.116)

\[ a(\theta) = 4.888124 \frac{h}{\beta^3} \sin \left( 4\pi \frac{\theta}{\beta} - \pi \right) \]  
(5.117)

\[ j(\theta) = 61.425769 \frac{h}{\beta^3} \cos \left( 4\pi \frac{\theta}{\beta} - \pi \right) \]  
(5.118)

For \( \left( \frac{5}{8} \beta \leq \theta \leq \frac{7}{8} \beta \right) \):

\[ s(\theta) = h \left[ -2.44406184 \left( \frac{\theta}{\beta} \right)^2 + 4.6660917 \left( \frac{\theta}{\beta} \right) - 1.2292648 \right] \]  
(5.119)

\[ v(\theta) = \frac{h}{\beta} \left[ -4.888124 \left( \frac{\theta}{\beta} \right) + 4.6660917 \right] \]  
(5.120)

\[ a(\theta) = -4.888124 \frac{h}{\beta^3} \]  
(5.121)

*Cam Motion Specifications*  
69
\[ j(\theta) = 0 \quad (5.122) \]

For \( \left( \frac{7}{8} \beta \leq \theta \leq \beta \right) \):

\[ s(\theta) = h \left[ 0.6110154 + 0.38898448 \left( \frac{\theta}{\beta} \right) + 0.0309544 \sin \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \right] \quad (5.123) \]

\[ v(\theta) = 0.38898448 \frac{h}{\beta} \left[ 1 + \cos \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \right] \quad (5.124) \]

\[ a(\theta) = -4.888124 \frac{h}{\beta^2} \sin \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \quad (5.125) \]

\[ j(\theta) = -61.425769 \frac{h}{\beta^3} \cos \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \quad (5.126) \]

For a follower during the fall segment of the motion profile, the following equations apply:

For \( \left( 0 \leq \theta < \frac{1}{8} \beta \right) \):

\[ s(\theta) = h \left[ 1 - 0.38898448 \left( \frac{\theta}{\beta} \right) + 0.0309544 \sin \left( 4\pi \frac{\theta}{\beta} \right) \right] \quad (5.127) \]

\[ v(\theta) = -0.38898448 \frac{h}{\beta} \left[ 1 - \cos \left( 4\pi \frac{\theta}{\beta} \right) \right] \quad (5.128) \]

\[ a(\theta) = -4.888124 \frac{h}{\beta^2} \sin \left( 4\pi \frac{\theta}{\beta} \right) \quad (5.129) \]

\[ j(\theta) = -61.425769 \frac{h}{\beta^3} \cos \left( 4\pi \frac{\theta}{\beta} \right) \quad (5.130) \]

For \( \left( \frac{1}{8} \beta \leq \theta < \frac{3}{8} \beta \right) \):
\[
s(\theta) = h \left[ -2.44406184 \left( \frac{\theta}{\beta} \right)^2 + 0.22203097 \left( \frac{\theta}{\beta} \right) + 0.99276593 \right] \quad (5.131)
\]

\[
v(\theta) = -\frac{h}{\beta} \left[ 4.888124 \left( \frac{\theta}{\beta} \right) - 0.22203097 \right] \quad (5.132)
\]

\[
a(\theta) = -4.888124 \frac{h}{\beta^2} \quad (5.133)
\]

\[
j(\theta) = 0 \quad (5.134)
\]

For \( \left( \frac{3}{8} \beta \leq \theta < \frac{5}{8} \beta \right) \):

\[
s(\theta) = h \left[ 1.3055077 - 1.6110154 \left( \frac{\theta}{\beta} \right) + 0.0309544 \sin \left( 4 \pi \frac{\theta}{\beta} - \pi \right) \right] \quad (5.135)
\]

\[
v(\theta) = -\frac{h}{\beta} \left[ 1.6110154 - 0.38898448 \cos \left( 4 \pi \frac{\theta}{\beta} - \pi \right) \right] \quad (5.136)
\]

\[
a(\theta) = -4.888124 \frac{h}{\beta^2} \sin \left( 4 \pi \frac{\theta}{\beta} - \pi \right) \quad (5.137)
\]

\[
j(\theta) = -61.425769 \frac{h}{\beta^3} \cos \left( 4 \pi \frac{\theta}{\beta} - \pi \right) \quad (5.138)
\]

For \( \left( \frac{5}{8} \beta \leq \theta < \frac{7}{8} \beta \right) \):

\[
s(\theta) = h \left[ 2.44406184 \left( \frac{\theta}{\beta} \right)^2 - 4.6660917 \left( \frac{\theta}{\beta} \right) + 2.2292648 \right] \quad (5.139)
\]

\[
v(\theta) = -\frac{h}{\beta} \left[ -4.888124 \left( \frac{\theta}{\beta} \right) + 4.6660917 \right] \quad (5.140)
\]

\[
a(\theta) = 4.888124 \frac{h}{\beta^2} \quad (5.141)
\]
\[ j(\theta) = 0 \] (5.142)

For \( \left( \frac{7}{8} \beta \leq \theta \leq \beta \right) \):

\[ s(\theta) = h \left[ 0.38898448 - 0.38898448 \left( \frac{\theta}{\beta} \right) - 0.0309544 \sin \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \right] \] (5.143)

\[ v(\theta) = -0.38898448 \frac{h}{\beta} \left[ 1 + \cos \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \right] \] (5.144)

\[ a(\theta) = 4.888124 \frac{h}{\beta^2} \sin \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \] (5.145)

\[ j(\theta) = 61.425769 \frac{h}{\beta^3} \cos \left( 4\pi \frac{\theta}{\beta} - 3\pi \right) \] (5.146)

All motion characteristics for the modified trapezoidal function are shown in Figure 5.7.

### 5.4 Polynomial Functions

The class of polynomial functions are the most versatile of all the motion functions. This framework provides for an infinite number of functions that can be designed to suit almost any application.

The general form for a polynomial function is

\[ s(\theta) = C_0 + C_1 \left( \frac{\theta}{\beta} \right) + C_2 \left( \frac{\theta}{\beta} \right)^2 + C_3 \left( \frac{\theta}{\beta} \right)^3 + \cdots + C_n \left( \frac{\theta}{\beta} \right)^n \] (5.147)
The constants $C_1, C_2, \ldots, C_n$ in Eq. 5.147 are unknown and it is these that are solved for in the development of the specific motion function. A polynomial cam design problem is uniquely defined by applying certain boundary conditions to the displacement, velocity, acceleration and jerk functions. Once these boundary conditions are set, the constants $C_1, C_2, \ldots, C_n$ can be solved for, and the polynomial function in Eq. 5.147 is defined (Norton, 1992). Two such examples of polynomial functions are described below.
5.4.1 3-4-5 Polynomial Function

The 3-4-5 polynomial function provides for a continuous cam profile throughout all derivatives. It is very similar in shape to the cycloidal function, but also has a relatively low value of peak velocity and peak acceleration.

For a follower during the rise segment of the motion profile, the following equations apply (Wilson and Sadler, 1993):

\[ s(\theta) = h \left[ 10 \left(\frac{\theta}{\beta}\right)^3 - 15 \left(\frac{\theta}{\beta}\right)^4 + 6 \left(\frac{\theta}{\beta}\right)^5 \right] \] (5.148)

\[ v(\theta) = \frac{h}{\beta} \left[ 30 \left(\frac{\theta}{\beta}\right)^2 - 180 \left(\frac{\theta}{\beta}\right)^3 + 120 \left(\frac{\theta}{\beta}\right)^4 \right] \] (5.149)

\[ a(\theta) = \frac{h}{\beta^2} \left[ 60 \left(\frac{\theta}{\beta}\right)^2 - 360 \left(\frac{\theta}{\beta}\right)^3 + 360 \left(\frac{\theta}{\beta}\right)^4 \right] \] (5.150)

\[ j(\theta) = \frac{h}{\beta^3} \left[ 60 - 360 \left(\frac{\theta}{\beta}\right) + 360 \left(\frac{\theta}{\beta}\right)^2 \right] \] (5.151)

For a follower during the fall segment of the motion profile, the following equations apply:

\[ s(\theta) = h \left[ 1 - 10 \left(\frac{\theta}{\beta}\right)^3 + 15 \left(\frac{\theta}{\beta}\right)^4 - 6 \left(\frac{\theta}{\beta}\right)^5 \right] \] (5.152)

\[ v(\theta) = \frac{h}{\beta} \left[ -30 \left(\frac{\theta}{\beta}\right)^2 + 60 \left(\frac{\theta}{\beta}\right)^3 - 30 \left(\frac{\theta}{\beta}\right)^4 \right] \] (5.153)
\[ a(\theta) = \frac{h}{\beta^2} \left[ -60 \left( \frac{\theta}{\beta} \right) + 180 \left( \frac{\theta}{\beta} \right)^2 - 120 \left( \frac{\theta}{\beta} \right)^3 \right] \]  

(5.154)

\[ j(\theta) = \frac{h}{\beta^3} \left[ -60 + 360 \left( \frac{\theta}{\beta} \right) - 360 \left( \frac{\theta}{\beta} \right)^2 \right] \]  

(5.155)

All motion characteristics for the 3-4-5 polynomial function are shown in Figure 5.8.

**Figure 5.8:** Motion Characteristics for the 3-4-5 Polynomial Function
5.4.2 4-5-6-7 Polynomial Function

The 4-5-6-7 polynomial function also provides for a continuous cam profile throughout all derivatives. It acts very similar to the 3-4-5 polynomial function, except that it constrains the jerk function to be zero at both ends of the interval.

For a follower during the rise segment of the motion profile, the following equations apply (Jensen, 1987):

\[
s(\theta) = h \left[ 35 \left( \frac{\theta}{\beta} \right)^4 - 84 \left( \frac{\theta}{\beta} \right)^5 + 70 \left( \frac{\theta}{\beta} \right)^6 - 20 \left( \frac{\theta}{\beta} \right)^7 \right] \tag{5.156}
\]

\[
v(\theta) = \frac{h}{\beta} \left[ 140 \left( \frac{\theta}{\beta} \right)^3 - 420 \left( \frac{\theta}{\beta} \right)^4 + 420 \left( \frac{\theta}{\beta} \right)^5 - 140 \left( \frac{\theta}{\beta} \right)^6 \right] \tag{5.157}
\]

\[
a(\theta) = \frac{h}{\beta^2} \left[ 420 \left( \frac{\theta}{\beta} \right)^2 - 1680 \left( \frac{\theta}{\beta} \right)^3 + 2100 \left( \frac{\theta}{\beta} \right)^4 - 840 \left( \frac{\theta}{\beta} \right)^5 \right] \tag{5.158}
\]

\[
j(\theta) = \frac{h}{\beta^3} \left[ 840 \left( \frac{\theta}{\beta} \right)^2 - 5040 \left( \frac{\theta}{\beta} \right)^3 + 8400 \left( \frac{\theta}{\beta} \right)^4 - 4200 \left( \frac{\theta}{\beta} \right)^5 \right] \tag{5.159}
\]

For a follower during the fall segment of the motion profile, the following equations apply:

\[
s(\theta) = h \left[ 1 - 35 \left( \frac{\theta}{\beta} \right)^4 + 84 \left( \frac{\theta}{\beta} \right)^5 - 70 \left( \frac{\theta}{\beta} \right)^6 + 20 \left( \frac{\theta}{\beta} \right)^7 \right] \tag{5.160}
\]

\[
v(\theta) = \frac{h}{\beta} \left[ -140 \left( \frac{\theta}{\beta} \right)^3 + 420 \left( \frac{\theta}{\beta} \right)^4 - 420 \left( \frac{\theta}{\beta} \right)^5 + 140 \left( \frac{\theta}{\beta} \right)^6 \right] \tag{5.161}
\]
\[ a(\theta) = \frac{h}{\beta^2} \left[ -420 \left( \frac{\theta}{\beta} \right)^2 + 1680 \left( \frac{\theta}{\beta} \right)^3 - 2100 \left( \frac{\theta}{\beta} \right)^4 + 840 \left( \frac{\theta}{\beta} \right)^5 \right] \]  

(5.162)

\[ j(\theta) = \frac{h}{\beta^3} \left[ -840 \left( \frac{\theta}{\beta} \right) + 5040 \left( \frac{\theta}{\beta} \right)^2 - 8400 \left( \frac{\theta}{\beta} \right)^3 + 4200 \left( \frac{\theta}{\beta} \right)^4 \right] \]  

(5.163)

All motion characteristics for the 4-5-6-7 polynomial function are shown in Figure 5.9.

**Figure 5.9:** Motion Characteristics for the 4-5-6-7 Polynomial Function
5.5 Cam Design Considerations

The first task of the designer is to select the mathematical functions that define the motion of the follower. In putting a cam together from a series of these motion curves, there are a few principles that must be kept in mind. First and foremost, the displacement function must be continuous throughout the full rotation of the cam. If a discontinuity is allowed in the displacement function, the follower will essentially experience an 'impact,' i.e., undergo a finite displacement in zero time. This is an unacceptable characteristic of any machine design system. Next, the characteristics of higher derivatives should be considered in the design. If the velocity function is discontinuous at a few points, for example, then the acceleration function will be infinite at those points. Referring back to Eq. 4.29, it is seen that these infinite spikes of acceleration will theoretically produce an infinite force. Although infinite force is not a realistic occurrence, the dynamic forces will nonetheless be quite large at these locations and will create high stresses and cause rapid wear of the cam and follower surfaces (Norton, 1992). Finally, discontinuities in the acceleration function bring about infinite spikes in the jerk function. Though these spikes in the jerk function are less detrimental to the system than would be discontinuities in the first or second derivatives of displacement, this facet of the system still usually makes it an unacceptable design.

As a simple design example, consider a cam design composed of a basic rise-dwell-fall-dwell arrangement. Suppose a simple harmonic motion function is desired for the rise portion, and a modified trapezoidal function for the fall portion of the follower. Referring to Figures 5.3 and 5.7, respectively, it can be seen that there will be no discontinuities in the displacement function of this cam; the harmonic function rises to the value $h$, dwells at
this value, and then falls back to zero with modified trapezoidal motion. The velocity function also retains continuity throughout; it rises and falls sinusoidally back to zero, dwells at zero, then falls and rises back to zero. However, the acceleration function presents a small problem. At the beginning of the rise interval the acceleration function jumps to a finite value instantaneously. And at the end of this interval, it also falls back to zero acceleration in an infinitely small time interval. Aside from this interval, the rest of the acceleration function is continuous throughout. This same problem with discontinuities is encountered in the fall portion of the jerk function. Overall, this would be a bad design mainly because of the discontinuities in the acceleration function, as well as the jerk function. The motion characteristics of this example arrangement and the infinite spikes in the jerk function are illustrated in Figure 5.10.

There are some characteristics inherent to some curves that make them preferable over others. There is often a need to design a cam and follower system to meet specific force, velocity or acceleration requirements. For example, in any cam design there is usually a desire to reduce the bending moment, as discussed in Section 4.5 of this thesis. Putting all other concerns of continuity aside for the moment, consider the choice between the cycloidal displacement function and the modified trapezoidal function. Referring to Figure 5.4 the maximum acceleration of the cycloidal displacement function is seen to be

$$a_{\text{max}} = \frac{2\pi h}{\beta^2}$$ (5.164)

And from Figure 5.7 the peak acceleration of the modified trapezoidal function is

$$a_{\text{max}} = 4.8812 \frac{h}{\beta^2}$$ (5.165)
But by combining Eq. 4.28 and Eq. 4.29, it can also be seen that

\[ M = Fl = (ma)l \]  

(5.166)
So in the case of trying to minimize the maximum bending moment, the modified trapezoidal function would be chosen because of its lower peak acceleration characteristics.

In reality, designing a cam to fit only a displacement criterion is a simple task. It is the considerations of the higher derivatives that make a cam either feasible or inappropriate for certain applications. In fact, the fundamental law of cam design is such that the design of any cam is acceptable if and only if all motion functions are continuous through the displacement, velocity and acceleration functions (Norton, 1992). In addition, the jerk function should also be finite across the entire interval.

5.6 Pressure Angle Considerations

Pressure angle is an important consideration in the design of a cam-and-follower system. The pressure angle, $\alpha$, of a system is simply defined as the angle between the direction of motion of the follower and the direction of the force imparted on the follower by the contacting cam surface.

For most applications, a pressure angle less than 30° will result in acceptable operation of the system. Rothbart (1956) was successful in using pressure angles up to 47½° for light loads with accurate, low-friction bearings. Despite this anomaly, a pressure angle greater than 30° might result in the binding of translating followers, the increased effects of wear on the cam or follower contact surface, as well as other problems in the mechanical
operation of the cam-and-follower system. For convention, the pressure angle is usually
designated to be in the first quadrant, i.e., $0 \leq \alpha \leq \frac{\pi}{2}$.

With the use of conjugate geometry, the value of the pressure angle, $\alpha$, can be solved for explicitly for each cam-and-follower case. Following the cam synthesis, the maximum pressure angle can be determined for the cam as it goes through a full revolution.

In synthesizing a cam profile from several different motion curve segment types, pressure angle also plays an important factor in the choice of these curve types. Section 5.5 of this thesis discusses the selection of motion curves to improve the smoothness at rise and fall intervals and increase the overall continuity of the graphs. Pressure angle considerations can much in the same way be taken into account to help to further reduce forces and minimize bending in the follower. If the angular position of the cam is known where the maximum pressure angle occurs, then this is the critical point at which some design changes might need to be made. For instance, one way the pressure angle can be reduced is by increasing the base circle radius of the cam (Mabie and Reinholtz, 1987). Even so, one can now see the importance of taking pressure angle characteristics into account, in addition to the continuity considerations in the design of a cam.
5.6.1 Translating Flat-Faced Follower

Referring back to Figure 4.5, the pressure angle for a translating flat-faced follower and disk cam system can be found directly from the synthesis equations by taking the dot product of the common normal and the follower velocity vector equations. The equations for the normal vector $\hat{N}$ and the relative velocity vector $\frac{d\bar{P}}{d\theta}$ are given by Eq. 4.4 and Eq. 4.5, respectively. The relative velocity vector will be considered to have unit magnitude since one is only interested in the direction of the vectors. Since the dot product of two vectors is equal to their magnitudes multiplied by the cosine of the angle between them, the pressure angle can be found from the following relationship:

$$\cos \alpha = \left[ e^{i(\theta + \psi + \frac{\pi}{2})} \cdot e^{i\theta} \right]$$  \hspace{1cm} (5.167)

Solving this equation yields the value for the pressure angle, $\alpha$,

$$\alpha = \psi + \frac{\pi}{2}$$

$$= \left[ \left( \psi + \frac{\pi}{2} \right) - \pi \right] \hspace{1cm} \left( 0 \leq \alpha \leq \frac{\pi}{2} \right)$$

$$= \left| \psi - \frac{\pi}{2} \right|$$

$$= \beta$$  \hspace{1cm} (5.168)

The pressure angle for any translating flat-faced follower is always the value of the oblique angle of the flat-faced follower, $\beta$. Obviously then, when the face is normal to the line of action of the follower stem, the pressure will be constant at $\alpha = 0^\circ$. 

Cam Motion Specifications
5.6.2 Oscillating Flat-Faced Follower

Referring back to Figure 4.6, the pressure angle for an oscillating flat-faced follower and disk cam system can also be found directly from the synthesis equations by taking the dot product of the common normal and the follower velocity vector equations. The equations for the normal vector $\hat{N}$ and the relative velocity vector $\frac{d\bar{P}}{d\theta}$ are given by Eq. 4.9 and Eq. 4.10, respectively. So the pressure angle can be found from the following relationship:

$$\cos\alpha = \left[ e^{i\theta + \frac{\pi}{2}} \cdot e^{i\theta + \frac{\pi}{2}} \right]$$

(5.169)

Solving this equation yields the value for the pressure angle, $\alpha$,

$$\alpha = \pi$$

$$= 0^\circ \quad \left(0 \leq \alpha \leq \frac{\pi}{2}\right)$$

(5.170)

The pressure angle for an oscillating flat-faced follower is always constant, and is equal to zero degrees in this case. This is because the direction of motion of the follower and direction of the transmission of force are both along the same line throughout the complete rotation of the cam. It should be noted that this is only the case when the follower arm is in the form of a single vector in the direction from its pivot point to the point of contact on the cam, as depicted in Figure 4.6. If this follower arm is manufactured with a kink or bend in it, the conjugate geometry analysis equations would need to be derived again for this special case. The pressure angle would not be equal to
zero in this special case, but it would indeed remain constant for the full rotation of the cam.

5.6.3 Translating Roller Follower

Referring back to Figure 4.7, the pressure angle for a translating roller follower and disk cam system can again be found directly from the synthesis equations by taking the dot product of the common normal and the follower velocity vector equations. The equations for the normal vector $\hat{N}$ and the relative velocity vector $\frac{d\hat{P}}{d\theta}$ are given by Eq. 4.15 and Eq. 4.16, respectively. So the pressure angle can be found from following relationship:

$$\cos \alpha = \begin{bmatrix} e^{i(\theta + \psi)}, e^{i\theta} \end{bmatrix}$$

(5.171)

Solving this equation yields the value for the pressure angle, $\alpha$,

$$\alpha = \psi$$

$$= |\psi - \pi| \quad 0 \leq \alpha \leq \frac{\pi}{2}$$

(5.172)

From this result it is seen that the pressure angle for any translating roller follower is always equal to the angle of rotation of the roller on the follower.
5.6.4 Oscillating Roller Follower

Referring back to Figure 4.8, the pressure angle for an oscillating roller follower and disk cam system can be found directly from the synthesis equations by taking the dot product of the common normal and the follower velocity vector equations. The equations for the normal vector $\hat{N}$ and the relative velocity vector $\frac{d\vec{P}}{d\theta}$ are given by Eq. 4.21 and Eq. 4.22, respectively. So the pressure angle can be found from following relationship:

$$\cos \alpha = \left[ e^{i(\theta + \psi)} \cdot e^{i\left(\theta + \phi - \frac{\pi}{2}\right)} \right]$$

(5.173)

Solving this equation yields the value for the pressure angle, $\alpha$,

$$\alpha = \psi + \frac{\pi}{2}$$

$$= \left[ \left(\psi + \frac{\pi}{2}\right) - \pi \right] \quad \left( 0 \leq \alpha \leq \frac{\pi}{2} \right)$$

(5.174)

$$= \left| \psi - \frac{\pi}{2} \right|$$

From this result it is seen that the pressure angle for any oscillating roller follower is always equal to ninety degrees plus the angle of rotation of the roller on the follower.
6. Program Description and Examples

6.1 Overall Description of Software

The Cam Design System is a user-interactive software system for the synthesis and kinematic analysis of planar disk cams that runs as an add-on application within the CADKEY® graphical environment. As an interactive application in CADKEY®, the user has the advantage of accessing functions for the Cam Design System as well as the regular CADKEY® graphical functions. By integrating the design system and the CAD package, the user will have a great deal of flexibility in using the Cam Design System.

6.1.1 Software Development

The Cam Design System is programmed in the C language, first invented and implemented by Dennis Ritchie on a UNIX-based DEC machine (Schildt, 1990). Today C has become a very popular, useful and portable block-structured programming language. Of particular interest to modern programmers is the ability of the C language to handle a wide array of applications involving complex structures and graphics.

The Cam Design System is compiled using the MetaWare® High C/C++™ v3.1 globally optimizing compiler (MetaWare®, 1993). This is a commercially available 32-bit C
language compiler that was chosen because of its use by Cadkey, Inc. in the original
design of the CADKEY® software, and because of its use in documentation with many
example CDE files. The GNU™ C freeware package is another 32-bit C and C++
language compiler that can be used in the development of this software (Free Software
Foundation, 1992). It should also be noted that since the CADKEY® software has been
developed as a 32-bit application, CDEs and all other extensions to it must also be 32-bit.
Therefore, it is necessary to use a 32-bit C compiler in the design of CDE applications.
The Phar Lap™ 386|DOS-Extender v4.0 (Phar Lap™ Software, 1991) or equivalent
product is also required when creating the final Cam Design System CDE file. This
software serves to link the object files together to create the actual CDE file, contains
definitions pertinent to the operation of the CDE application, and helps to handle the
computer memory requirements during the program compilation.

In addition, the user must utilize the header files and function prototypes contained in the
CADKEY® Software Development Kit v6.0 package (Cadkey, 1993b). This package
contains all the essential documentation and header files to create the necessary data
structures and encapsulate all the information about the C-code functions. The CDE
generator contained in this package serves to then decipher the functions in the CDE
application by using these data structures.
6.1.2 Running the Software

To activate the Cam Design System, the user simply selects the 'Cam Design System' title from the Applications Menu in CADKEY® (option F9) and presses the OK button. Once the Cam Design System is activated, a menu appears at the top of the CADKEY® screen. This menu is the top-level menu to the user, and allows the user to select from a category of functions, namely Synthesis, Analysis, Output, or Exit. This top-level menu for the Cam Design System and the overall appearance of the CADKEY® design environment is shown in Figure 6.1. Once the user clicks the left mouse button on one of these functions, a small window expands below the menu selection to show which options are available in the program.

Under the Synthesis option, the user may select: (1) Synthesize Cam, to begin the synthesis of a new cam; or (2) Modify Cam, to edit certain parameters of an already defined cam profile. The synthesis category is where the user may specify different follower types, motion segments and curve types for the follower motion, and where the final cam profile is rendered on the screen.

Under the Analysis option, the user may select: (1) View Graphs, to specifically view the SVAJ (displacement, velocity, acceleration and jerk) graphs for a certain cam profile already defined; or (2) Kinematic Analysis, to examine an already defined cam for pressure angle characteristics.

Under the Output option, the user may select: (1) Start New Cam, to purge the current cam data from the RAM and begin the definition of another; (2) Engineering Drawing,
Figure 6.1: Top-Level Menu for the Cam Design System
produce a fully-detailed drawing of the cam profile, complete with border and title block; (3) **Report**, to produce a full report on an already defined cam, including its parameters and motion curve characteristics; (4) **Save SVAJ Data**, to save the full SVAJ data list to an ASCII text file; (5) **Save Cam Data**, to save the cam parameter data to an ASCII text file; (6) **Save Drawing**, to save the drawing to a CADKEY® part (.PRT) file and/or pattern (.PTN) file; or (7) **Manufacturing (CL Data)**, to produce an ASCII text file containing the cutter location data (CL Data) necessary to manufacture an already defined cam profile.

The **Exit** option closes the Cam Design System software application from within CADKEY®, clears all RAM related to the Cam Design System, and allows the user to return to normal CADKEY® use. It does, however, leave the current cam design or whatever else may be located in the current viewport on the screen after program exit.

Overall, the Cam Design System provides enough functionality and versatility for the user to build and analyze a multitude of cams. The top-level pull-down menu is a popular, easy-to-use interface design in most current software. Also, the use of dialog boxes for interaction with the user makes the software user-friendly and convenient.

### 6.1.3 Program Requirements

First and foremost, the Cam Design System requires the CADKEY® v6.0 software running on an IBM® PC or compatible. It is recommended that the system that will be
running the Cam Design System at least meet the minimum requirements necessary to run the CADKEY® v6.0 software. In other words, the system should be of at least 386DX CPU power, and 33 MHz speed. VGA graphics capabilities are also a necessity. It is also suggested that in order to be safe the Cam Design System be run with no less than 400K of free RAM, in addition to that required by the CADKEY® software. Also, there must be at least 300K free on the hard disk after the program has been loaded.

The Cam Design System uses the external file CAMPATH.CAM to indicate the path where the Cam Design System software is located. This text file contains two lines: the first is the default drive, such as 'C:', and the second contains the default path where the Cam Design System files are located, such as '\CAM\'. Note that each of the backslashes must be included in the path specification (beginning and end). This file also may be modified by the user according to where the Cam Design System is being installed. It is critical that this file be copied into the 'CADKEY6' directory (or in whichever directory the CADKEY® v6.0 software is located) before the application is executed. The software will not operate if this procedure is not performed.

The program uses the external file CAM.TRE as its defining guideline for the user-interactive pull-down menu tree structure located at the top of the CADKEY® drawing screen. This serves as CADKEY®'s general framework on all function calls and as a guideline for the general flow and operation of the software. This file must be located in the Cam Design System default directory (as specified in the file CAMPATH.CAM), and should not be modified by the user under any circumstances. It is therefore a good idea for the user to make these 2 files, CAM.TRE and CAMPATH.CAM, hidden and/or read-only (by using the DOS® command 'ATTRIB.EXE +HR CAM.TRE', for instance).
The file **RDMENU.CDE** is a CDE file that contains all the menu commands necessary to properly execute the Cam Design System and its top-level pull-down menu in the CADKEY® environment. This CDE file must be opened before the execution of the CAM.CDE application. This is most easily facilitated by setting this (and the CAM.CDE file, for that matter) as both automatically loaded CDE files in the CADKEY® default configuration settings.

The file **CAM.CDE** is the actual CDE file for the Cam Design System. This 155K file is loaded as an application into CADKEY® after the file RDMENU.CDE has been loaded and begins immediately thereafter.

Other files may be created once the Cam Design System is executed, such as ASCII text files containing the SVAJ data, and drawings and reports. This will always take place within the default path, as designated by the user in the file CAMPATH.CAM. All of the files created by, loaded into, or used by the Cam Design System will have a default extension of '.CAM' unless otherwise designated by the user.

Certain structures and variables are created internally to the program as they are needed by the Cam Design System. As mentioned above, the program frees all RAM formerly taken by the data structures upon exit. The Cam Design System may also be executed and halted as many times as desired within each CADKEY® session, thus enabling the user to produce and manipulate several different cam profiles within the same session.
6.2 Specific Structure of Code

The Cam Design System consists of a CDE-file compiled from a group of C-code which contains a total of 26 different functions in 8 separate C files. The content of each of these files is as follows:

- **CALC.C**: Contains all code to calculate all displacement, velocity, acceleration and jerk data (SVAJ data) for each of the 8 different curve types described in Sections 5.1-5.4 of this thesis. All these formulae are evaluated at each 0.50° increment of the overall cam angle in order to produce the cam profile.

- **INTER.C**: Contains all code to set up the user interface for the software, and initializes all variables and allocates all memory necessary for the program.

- **MISC.C**: Contains all code necessary for kinematic analysis and for viewing the follower motion curves once a cam profile has been designed.

- **OUTPUT.C**: Contains all code necessary for output, including any data files, drawings, reports, or NC data.

- **PROFIL.C**: Contains all code for the entering of data necessary for the synthesis of the cam profile, such as motion profile, curve type and stroke length.

- **RENDER.C**: Contains all code for the final rendering of the cam, including the specific conjugate geometry solutions for each follower type and data for rendering different hub and follower types.

- **SPECS.C**: Contains all code necessary for entering cam parameters related to profile synthesis, such as follower type and angular velocity of the cam, base circle radius, offset values, and hub type.
• UTILS.C: Contains code for a variety of functions for manipulating structures, 
installing the Cam Design System into the CADKEY® Applications menu, and 
defining the icons used to visually describe the different follower types.

In addition, there are 2 header files, CAM.H and GLOBAL.H, which contain the 
prototypes for all functions and global structures, and other global variables, respectively. 
An additional definition file, OTHER.DEF, contains the overall definitions and 
declarations of all the functions to be used in the compilation of the final CDE file 
(CAM.CDE) for the Cam Design System. This is the file used to actually generate the 
overall instructions, interfacing routines, and global constants for the final CDE file. The 
final CAM.CDE file is created by way of the software supplied in the CADKEY® 
Software Development Kit v6.0 (Cadkey, 1993b) and the MetaWare® High C/C++™ 
v3.1 compiler (MetaWare®, 1993). All of this source code has been very heavily 
commented and is contained in Appendix A of this thesis.

An attempt has been made to keep the number of global variables to a minimum. The 
Cam Design System uses four global doubly-linked lists. The first of these is used to 
keep track of the data for the cam profile motion segments (used in profile synthesis). 
The second is dedicated to the SVAJ data produced when calculating the cam data (also 
used in profile synthesis). The last two are used for storing the pressure angle values 
throughout the complete rotation of the cam and for storing the NC data for the cam 
profile, respectively. The program also uses two global data structures in order to have 
full access to them from any point of execution in the software. One structure is used to 
store the cam parameters, such as follower type, angular velocity, and maximum lift 
value, and another is used to store specific cam parameters pertinent to rendering the final
cam profile. Also included as global variables are such things as a generic prompt
variable, a generic dialog box and menu variable, flag variables to indicate whether or not
a cam has already been defined for instance, a global viewing-scale variable, and two
variables for numerical conversion factors.

As a CDE application, the Cam Design System executes independently from within
CADKEY®. Once the program begins, the first two functions of the program contained
in the file INTER.C, Cam_Design_System (the 'main' function) and run_toolbar, serve
to internally initialize several variables and set up the CADKEY® viewport screen before
any user interaction begins. Much of this initialization dictates how the program will act
within CADKEY® and how certain objects are drawn. More specifically, all immediate
mode commands are enabled during the Cam Design System's operation, so that the
cursor-pick device dynamically changes coordinate values as it is moved, and so that
other CADKEY® functions are enabled during the program execution. All drawing is
done in the world coordinate system, with the display axes in the lower left corner of the
screen and the display grid both turned off. All drawing is done on level #5, with pen #5
(to more easily facilitate entity detection and plotting). All units are in millimeters, and
all entity attributes, e.g., note sizes, color, and line thickness, are set to the normal default
CADKEY® values. The CADKEY® screen is set to a window of 200 mm width (-100.0
mm to +100.0 mm) by 200 mm height (-100.0 mm to +100.0 mm). The drawing scale is
reset to 1.0, and the origin is located at (0.0, 0.0, 0.0). The program will also clear all
entities from the database and the CADKEY® screen each time before the Cam Design
System begins.
Most functions are accessible to the Cam Design System directly from the top-level menu: `cam_params`, and `edit_cam` for the Profile option; `view_graphs` and `kinematic` under the Analysis option; and `start_new_cam`, `eng_drawing`, `report`, `save_svaj_data`, `save_cam_data`, `drawing_save`, and `manufacture` under the Output option. In addition, there are several other functions that are called internally from within these functions that also interact with the user, but are not accessed directly by the user. For example, `enter_motion_program` and `cam_render` are two of these that allow the user to enter the motion profile and render the cam on the CADKEY® screen, respectively. Also, there are several 'utility' functions used internally by the program, such as `forward_it`, `rewind_it`, and `install_it`. These serve to advance or rewind the profile structure to a certain segment, and to install the Cam Design System into the CADKEY® Applications menu, respectively.

### 6.3 Menu Structure and User Information

The flowchart shown in Figure 6.2 displays the tasks involved in a sample execution of the Cam Design System. It shows the user creating a cam from scratch, then analyzing and verifying its characteristics, and then exporting a report, drawing and numerical data to disk before exiting.

Figure 6.3 shows the Cam Parameters menu for the Cam Design System. This is the first menu available to the user in the synthesis of a cam under the Synthesize Cam option in the Synthesis category. In this menu the user must first specify the type of

*Program Description and Examples*
Figure 6.2: Flow Diagram of Typical Cam Design System Run
Figure 6.3: Cam Parameters Menu
follower arrangement by either clicking the mouse on the radio box or on the actual icon-picture of the cam-and-follower arrangement. The user must also designate the direction of rotation of the cam. Clockwise rotation of the cam is taken in the positive sense. This is also an important parameter in the design since the direction of the rotation of the cam is part of the original assumptions made in the conjugate geometry analysis described in Section 4.4 of this thesis. For this analysis, the cam-and-follower system was inverted, i.e., the cam was held fixed as the follower indexed around it. For instance, if the cam rotates in a clockwise direction, i.e., has a positive value for angular velocity, then this direction is opposite to that assumed in the conjugate geometry analysis. This can be seen in the sense of the cam angle $\theta$ in Figure 4.5. This will result in an incorrect cam profile. The user needs only to make sure the sense of the rotation of the cam is correct. The Cam Design System will take into account the rotation in its analysis and synthesize the cam properly. The user must press the OK button to advance to the next menu.

The next menu available to the user is the Motion Segments menu, which is shown in Figure 6.4. This menu allows the user to enter a motion profile with any combination of rise, fall or dwell arrangements for the follower. Three common cases are provided, Rise-Fall, Rise-Fall-Dwell, and Rise-Dwell-Fall-Dwell. The user also has the option to create a custom profile piece by piece through the use of the buttons marked "Rise," "Fall," and "Dwell." A "Backup" key is supplied for ease of use in editing the motion profile, which is dynamically displayed at the bottom of the screen. After a maximum of six segments has been entered, the user must press the OK button to advance to the next menu.

The next menu is the Motion Profile menu, which is shown in Figure 6.5. This menu allows the user to specify the exact motion of the follower. The user may pick any of the
Figure 6.4: Motion Segments Menu
Figure 6.5: Motion Profile Menu
nine different curve types by clicking the left mouse button on the proper curve type radio button, and then selecting the button for the desired motion curve. The angle ranges are entered in the boxes shown, and the Cam Design System checks to ensure that they add up to a full 360° of the cam rotation. The stroke length for a rise or a fall are also entered here. The program also makes sure that the overall displacement of the follower is zero. Once the user has selected these parameters, and the Cam Design System has verified this, the user must press the OK button to return to advance to the next menu.

At this point in the program the Cam Design System will internally calculate all the SVAJ data for the cam profile, and will then bring up the Cam Profile Specifications menu, which is shown in Figure 6.6. This menu allows the user to specify certain parameters for the cam synthesis that are specific to the type of follower. The user must first enter the base circle radius. Depending on which type of follower is being used there are other parameters to enter as well. For a translating flat-faced follower, for example, Figure 6.6 indicates a box for an oblique angle. This and other particular values could be left at zero if so desired. For all oscillating followers, the x and y locations of the pivot point of the arm of the follower must be denoted. Once all necessary parameters have been entered, the user must select the OK button to render the cam. The cam is then rendered with each separate motion segment shown in a different color for clarity. This then returns the user to the top-level menu. It should also be noted here that this cam is in fact a three-dimensional image in the CADKEY® database. The Cam Design System has been programmed such that z values of zero exist for all entities defined in the system.

Now that the cam contour has been created the user has several options available. By selecting the Analysis category and the View Graphs option, the user may view the
Figure 6.6: Cam Profile Specifications Menu

Can Profile Specifications

Translating Flat-Faced:

Can Hub Type: 1
(1 - 2 available)

OK
Cancel

Base Circle Radius (C): 57.00
Oblique Angle (B): 15.00
displacement, velocity, acceleration and jerk functions on screen to determine if this is a valid and robust cam design. Once again each of these graphs shows the different segments of the follower motion profile in a different color for clarity. The user may also select Kinematic Analysis, which gives a graph of the pressure angle, $\alpha$, versus the cam angle, $\theta$, for the complete rotation of the cam. It will also alert the user to the maximum value of the pressure angle and at what cam angle that it occurs.

Back at the top-level menu, the user may select the Output category for another range of functions. The Start New Cam option allows the user to erase the current cam design from the screen and from memory, and to start a new design. The Engineering Drawing option allows the user to produce a drawing of the cam profile, complete with a border and a title block. This drawing is saved in a CADKEY® part file (.PRT) and/or pattern file (.PTN) format. The user may also later convert this to an IGES or DXF file for use by another CAD system. The Report function allows the user to produce a report on a cam. This report is in ASCII text format, and includes a list of the cam parameters and motion curve characteristics. The Save SVAJ Data option allows the user to save the full list of SVAJ data to an ASCII text file for use in another application. The Save Cam Data option saves only the cam parameter data to an ASCII text file. The Save Drawing option saves the drawing as-is to a CADKEY® part file and/or pattern file. Finally, the Manufacturing (CL Data) function allows the user to produce an ASCII text file containing the cutter location data (CL Data) necessary to manufacture an already defined cam profile. This cutter location data may also be imported into other NC software packages to be used in the manufacture of the part.
Once a cam profile has already been defined, the user may edit certain parameters of the cam specifications and re-generate the cam. To do this the user must select the **Modify Cam** option from the Synthesis category in the top-level menu. The user must then select which facet of the design should be changed, and may do so by selecting any of the four buttons: Follower Type, Motion Segments, Motion Profile, Cam Parameters. This will bring up the corresponding menu with the current parameters intact. The user may edit any of these parameters, and must then press the **OK** button back in the Edit menu once all modifications have been completed. This will submit the cam parameters to the Cam Design System, and a new, updated cam will be rendered.

Once all procedures have been executed to the extent desired, the user may depart the program by selecting the Exit category from the top-level menu.

### 6.4 Specific Examples

#### 6.4.1 Translating Flat-Faced Follower Example

**Problem:** A planar disk cam rotating counterclockwise at 100 rpm drives a translating oblique flat-faced follower through a total displacement of 18 mm. The follower stem is at a counterclockwise 15° angle to the flat-face, and the minimum radius of the cam is 57 mm. Use a dwell-rise-dwell-fall motion specification, with cycloidal displacement motion for the rise (80° - 195°), and a modified trapezoidal curve for the fall (309° -
360°). Determine the full cam profile with a 0° vertical hub (type 1), verify the SVAJ data, and determine the maximum pressure angle.

**Solution:** Under the **Cam Parameters** option in the main menu, select the 'Translating Flat-Faced Follower' icon and select counterclockwise for the direction of angular velocity, and hit ENTER. After this select the **OK** button to save the parameters and go to the next menu.

Next, the specific motion requirements for this disk cam need to be entered. Enter the motion segments now: Use a DRDF arrangement (dwell-rise-dwell-fall) by picking each button separately, then choose the **OK** button. Next choose the **Cycloidal Displacement** button for the rise segment, and the **Modified Trapezoidal** button for the fall segment. Then enter the angle ranges for each of the segments, which are 80, 115, 105, and 60 degrees, respectively. Enter the stroke length for segments 2 and 4 as 18 for each. This screen is shown above in Figure 6.5 in the Motion Profile menu. Once all of these values have been entered, select the **OK** button to submit these values and go to the next menu.

Next, the user must enter specific cam parameters for the final cam synthesis. Enter the radius of the base circle, c, which is 57 mm. Then, the oblique angle of the follower stem must be entered, which is +15° (counterclockwise to the follower face is defined as positive). Finally, hub type 1 should be selected. This screen is shown above in Figure 6.6. After this, the program will render and autoscale the cam profile, with the portion of the cam corresponding to each of the motion segments in a different color (for clarity).
This cam profile and follower are shown in Figure 6.7. The user is then prompted to hit ENTER to continue. The program then returns control to the user at the top-level menu.

Select the option to view the SVAJ graphs to verify that the curves are correct and if they provide for a viable cam design. These graphs are shown in Figure 6.8. The only problem with these graphs is in the jerk function, where discontinuities abound. However, the other 3 functions are smooth and continuous, so that in this respect the cam design is feasible up to this point. Next, execute a kinematic analysis to verify that the maximum pressure angle is $\alpha_{\text{max}} = \beta = 15^\circ$. This pressure angle is far below the safe working limit of 30\(^\circ\). This cam is therefore an acceptable design.

6.4.2 Translating Roller Follower Example

**Problem:** A planar disk cam rotating counterclockwise at 200 rpm drives a translating offset roller follower through a total displacement of 38 mm. The centerline of the follower is offset 13 mm to the right of and parallel to the vertical centerline of the cam. The minimum radius of the cam is 25 mm and the roller diameter is 22 mm. Use a rise-dwell-fall-dwell motion specification, with simple harmonic motion for the rise (0\(^\circ\) - 100\(^\circ\)), and a modified sine curve for the fall (200\(^\circ\) - 275\(^\circ\)). Determine the full cam profile with a 90\(^\circ\) right hub (type 2), verify the SVAJ data, and determine the maximum pressure angle. (This problem is originally taken from Mabie and Reinholtz, 1987, "Mechanisms and Dynamics of Machinery," Problem 3.4, and is modified for use in this thesis).
**Solution:** Under the **Cam Parameters** option in the main menu, select 'Translating Roller Follower' and select counterclockwise for the direction of angular velocity, and hit ENTER. After this select the **OK** button to save the parameters and return go to the next menu.

Next, the specific motion requirements for this disk cam need to be entered. Enter the motion segments now: Use a RFD arrangement (rise-dwell-fall-dwell), and choose the **SHM Curve** button for the first segment (rise), and the **Modified Sine** button for the third segment (fall). Then enter the angle ranges for each of the segments, which are 100, 100, 75, and 85 degrees, respectively. Since the total displacement of the roller follower is 38 mm, enter this as the stroke length for the rise and fall segments (segments 1 and 3). Once all of these values have been entered, select the **OK** button to submit these values and go to the next menu.

Next, the user must enter specific cam parameters for the final cam synthesis. Enter the radius of the base circle, c, which is 25 mm. Then the offset of the roller follower, d, must be entered, which is +13 mm (an offset to the right of the centerline of the follower is defined as positive). Finally, the radius of the roller, r, must be entered, which is +11 mm (radius = 22 mm, diameter / 2 = 11 mm), and hub type 2 should be selected. After this, the program will render and autoscale the cam profile. This cam contour is shown in Figure 6.9. The user is then prompted to hit ENTER or continue. The program then returns control to the user at the top-level menu.

Select the option to view the SVAJ graphs to verify that the curves are correct and if they provide for an acceptable cam design. These graphs are shown in Figure 6.10, and look
Figure 6.10: Motion Characteristics for Translating Roller Follower Example
to be what is expected. Both the displacement and velocity functions are continuous, but the acceleration and jerk functions both contain spikes at a few intervals. The discontinuities in the acceleration curve make this cam unacceptable from this standpoint. Next, execute a kinematic analysis to verify that the maximum pressure angle is \( \alpha_{\text{max}} = 22.51^\circ \). This pressure angle is just below the limit of 30\(^\circ\), and is acceptable. However, because of the discontinuities in the higher derivatives this cam-and-follower system will not be an acceptable one.
7. Conclusions and Recommendations

7.1 Summary

The Cam Design System software is intended to serve as an interactive graphical system to simplify and streamline the design and analysis process for planar disk cams. Compared to manual design, this method is much faster, easier, neater, more accurate and more user-friendly. It also employs the CAD system as the central tool of the designer in building the cam profile. The user has the option to make modifications to a cam design and see the results within minutes, as opposed to the drafting method where this process would take considerably longer. Manual cam design would possibly require blueprints and specific cutter location data, which are difficult to acquire from a non-symmetrical and complex drawing such as that of a cam profile. The nature of this software, however, is such that once a cam contour is designed it is ready-made to be manufactured from the specifications on the CADKEY® screen and from numerical cutter location data (CL data) exported to a file.

The Cam Design System also makes for an excellent implementation and adaptation of the theory of conjugate geometry. With the use of conjugate geometry, the full cam profile can be precisely defined mathematically as the point of contact between the cam and the follower throughout its full rotation. In addition, this code should be portable to other hardware systems and operating environments, requiring principally a change in the graphical-interface and user-interface portions of the C-code.
7.2 Future Work

The Cam Design System source code contained in Appendix A of this thesis is by no means optimal. Every effort was made to use dynamic structures and pointers, to limit the use of global and extraneous variables and loops, and to document the code throughout. Notwithstanding these efforts, and although the code is already quite modular and efficient in design, the Cam Design System C-code would likely benefit from the scrutiny of a more experienced programmer.

Although the Cam Design System already expedites the current cam design process, there is still much potential to add increased capabilities for the system. The code is designed such that other follower types can be easily added, once the conjugate geometry analysis for these has been performed. An automotive valve train case would make this software even more useful, for instance. Along the same line of thought, the addition of cam types such as the ones explained in Section 4.1.1, as well as providing for the application of some of the follower types mentioned in Section 4.2 would make the software more valuable to designers. The addition of some other curve types aside from the ones mentioned in Chapter 5 would also provide the designer more versatility in building a cam. In addition, allowing the user to specify certain values of velocity and acceleration or limits at certain cam speeds would also provide a greater flexibility in the final design.

The inclusion of a dynamic analysis option for the software would provide the user the opportunity to scrutinize velocity, acceleration, force and torque characteristics of the cam and follower system. Also, some kind of animation routine would be extremely beneficial by allowing the user to visualize the follower in motion as the cam rotates. In
addition, the option of displaying a full three-dimensional solid or wireframe model of the cam and follower system would especially add to the visual understanding of the user.

Also, this software offers a unique opportunity as a building block to an automated design expert system. As one example, such a system could predict when certain motion curve combinations would bring about discontinuities in the follower motion and excessive forces on the follower. Such a system could also allow the user more control over sizing specific cam components such as the hub and follower stem. It could in fact realistically merge cam design into the overall machine design task.
8. References


11. Carpenter, K., 1992, "Trapezoidal Acceleration Cam Design," Final Project in Advanced Kinematics Graduate Course, Submitted to the Department of Mechanical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA.


52. Novak, J. C., 1992, "Programming an Elliptical Follower," Final Project in Advanced Kinematics Graduate Course, Submitted to the Department of Mechanical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA.


71. Tjung, J. W., and Coe, D., 1992, "CAMSYN: A Cam Design Software Project," Final Project in Advanced Kinematics Graduate Course, Submitted to the Department of Mechanical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA.


Appendix A: Source Code Listing

File: CAM.H

/***************************************************************************/
/* Cam Design System - Overall Header File */
/***************************************************************************/
/* File: CAM.H */
/***************************************************************************/
/* Programmed by: */
/* */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* */
/* c/o Dr. Charles F. Reinholtz */
/* */
/* 106 Randolph Hall */
/* */
/* Department of Mechanical Engineering */
/* */
/* Virginia Tech */
/* */
/* Blacksburg, VA 24061-0238 */
/***************************************************************************/
/* Purpose: Sets up all the global typedefs, includes, and function */
/* prototype declarations for the Cam Design System. */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-01-94 */
/***************************************************************************/

/* necessary include files */
#include <math.h>
#include "ck_dlg.h"

/* used to define the value of pi to 45 significant digits (pre-defined */
/* in the <math.h> include file */
#define _PI

/* initialize function prototypes (with parameters passed to them) */
extern void cam_render (struct svaj **);
extern void erase_screen (int);
extern void forward_it (struct profile **, int);
extern void initialize_icons (DIALOG_BOX *);
extern void profile_specs (int, char[], double *, double *, double *,
double *, double *);
extern void rewind_curve (struct svaj **);
extern void rewind_nc (struct nc_data **);
extern void rewind_pressure (struct pressure_analysis **);
extern void rewind_profile (struct profile **);
extern void run_toolbar (DIALOG BOX *);

/* other function prototypes (all voids) */
extern void Cam_Design_System (void);
extern void calculate_cam (void);
extern void cam_params (void);
extern void drawing_save (void);
extern void edit_cam (void);
extern void eng_drawing (void);
extern void enter_motion_program (void);
extern void kinematic (void);
extern void manufacture (void);
extern void motion_segs (void);
extern void report (void);
extern void save_cam_data (void);
extern void save_sva_j_data (void);
extern void start_new_cam (void);
extern void user_warning (void);
extern void view_graphs (void);

/* initialize structure for cam data */
struct profile {
    int     seg_no;
    char   seg_type[5];
    char   curve_type[25];
    int   curve_type_no;
    double angle_range;
    double stroke_length;
    struct profile *next;
    struct profile *prev;
};

/* initialize structure for the follower type and cam velocity */
struct the_cam {
    char   follower_type[25];
    int     type_num;
    int     hub_type;
    double max_lift;
    double omega;
};

/* initialize structure for SVAJ (displacement, velocity, acceleration */
/* and jerk) data */
struct sva_j {
    float  index;
    double s;
    double v;
    double a;

Appendix A: Source Code Listing - CAM.H
double j;
int color;
struct svaj *next_one;
struct svaj *prev_one;
};

/* initialize structure for certain cam profile values that will be */
/* used later in the cam rendering and kinematic analysis */
struct kine_anal {
    double k1;
    double k2;
    double k3;
    double k4;
    double k5;
    double k6;
    double k7;
};

/* initialize structure for pressure angle checking */
struct pressure_analysis {
    double p_angle;
    double theta;
    struct pressure_analysis *next;
    struct pressure_analysis *prev;
};

/* initialize structure for NC data */
struct nc_data {
    double x;
    double y;
    double angle;
    struct nc_data *next;
    struct nc_data *prev;
};
Appendix A: Source Code Listing

File: GLOBAL.H

/****************************************************************************
/* Cam Design System - Global Variable Header File */
/****************************************************************************
/* File: GLOBAL.H */
/****************************************************************************
/* Programmed by: */
/* */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* */
/* c/o Dr. Charles F. Reinholdz */
/* */
/* 106 Randolph Hall */
/* */
/* Department of Mechanical Engineering */
/* */
/* Virginia Tech */
/* */
/* Blacksburg, VA 24061-0238 */
/****************************************************************************
/* Purpose: Sets up all the global constants for the Cam Design System. */
/* */
/* */
/* Author: Steve Payne Date of creation: 10-05-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
/****************************************************************************

/* initialize other miscellaneous global variables */
extern char prompt[65], input_drive[3], input_path[35];
extern char motion_profile[10];
extern double conv1, conv2, scale_cam;
extern int cam_defined, nc_flag, edit_flag;

/* initialize these structures as external, and assign them */
/* to a global variable */
extern struct prof;
extern struct the_cam *cam;
extern struct svaj *curve;
extern struct kine_anal *analysis;
extern struct pressure_analysis *pressure;
extern struct nc_data *nc;
Appendix A: Source Code Listing

File: OTHER.DEF

/*==================================================================*/
/* Cam Design System - All Defined Functions */
/*==================================================================*/
/* File: OTHER.DEF */
/*==================================================================*/
/* Programmed by: */
/* */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* */
/* c/o Dr. Charles F. Reisholtz */
/* */
/* 106 Randolph Hall */
/* */
/* Department of Mechanical Engineering */
/* */
/* Virginia Tech */
/* */
/* Blacksburg, VA 24061-0238 */
/*==================================================================*/
/* Purpose: Function definitions that are used to create the main.CDE */
/* header file from CADKEY's CDEGEN.EXE (from the SDK vf.0 Kit). */
/* */
/* Author: Steve Payne Date of creation: 10-01-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
/*==================================================================*/

open vcid install_it();
void Cam_Design_System();
void calculate_cam();
void cam_params();
void cam_render();
void drawing_save();
void edit_cam();
void eng_drawing();
void erase_screen();
void forward_it();
void initialize_icons();
void kinematic();
void manufacture();
void motion_segs();
void profile_specs();
void report();
void rewind_curve();
void rewind_nc();
void rewind_pressure();
void rewind_profile();
void save_cam_data();
void save_sraj_data();
void start_new_cam();
void run_toolbar();
void user_warning();
void view_graphs();
hide install_it;
hide Cam_Design_System;
hide calculate_cam;
hide cam_params;
hide cam_render;
hide drawing_save;
hide edit_cam;
hide eng_drawing;
hide erase_screen;
hide forward_it;
hide initialize_icons;
hide kinematic;
hide manufacture;
hide motion_segs;
hide profile_specs;
hide report;
hide rewind_curve;
hide rewind_nc;
hide rewind_pressure;
hide rewind_profile;
hide save_cam_data;
hide save_sraj_data;
hide start_new_cam;
hide run_toolbar;
hide user_warning;
hide view_graphs;
Appendix A: Source Code Listing

File: MAKEFILE

#*********************************************************************************************
# Cam Design System - Makefile
#*********************************************************************************************
# File: MAKEFILE.
#*********************************************************************************************
# Programmed by:
# #
# # Virginia Tech Robotics & Mechanisms Group
# # c/o Dr. Charles F. Reinholtz
# # 106 Randolph Hall
# # Department of Mechanical Engineering
# # Virginia Tech
# # Blacksburg, VA 24061-0238
#*********************************************************************************************
# Purpose: Compiles all necessary C code and header files, and creates the actual CDE file for
# the Cam Design System.
# #
# Author: Steve Payne Date of creation: 09-20-93
# Last modified by: Steve Payne Date of last modification: 02-11-94
#*********************************************************************************************

# Name of CDE file to create
PROJ = cam

# Command to invoke the MetaWare High C/C++ Compiler v3.1
CC = c:\highc\bin\hc386

# Compiler flags
FLAGS = -c -486 -O0 -Hon=387 -Hnoswap -Hnocopyr -Ic:\ckdev6\include -Ic:\highc\inc

# CADKEY CDE header file and C code generator
CDE = c:\ckdev6\bin\cdegen

# Phar Lap 386|DOS-Extender v4.0 (Linker)
LINK = c:\\users\payne\cam\386lib

# Object files to be linked to create $(PROJ).CDE
OBJS = inter.obj calc.obj misc.obj other.obj output.obj profil.obj \ render.obj specs.obj utils.obj
# Links .obj files together to create $(PROJ).CDE
/usr pais/cam/$(PROJ).CDE: $(OBJS)
  $(LINK) $(@) @cam_lib.lst

# Creates CADKEY header file and C code for CDE functions
other.C: other.def
  $(CDE) other.def

# Create these .obj files from .c files
calc.obj: calc.c cam.h global.h
inter.obj: inter.c cam.h
misc.obj: misc.c cam.h global.h
other.obj: other.c cam.h global.h
output.obj: output.c cam.h global.h
profil.obj: profil.c cam.h global.h
render.obj: render.c cam.h global.h
specs.obj: specs.c cam.h global.h
utils.obj: utils.c cam.h global.h

# Implicit rule for compiling
.c.obj:
  $(CC) $(FLAGS) <$
Appendix A: Source Code Listing

File: CAM_LIB.LST

-NOBACKUP
-NOBANNER
-TWOCASE
-REPLACE c:\users\payne\cam\calc.obj
-REPLACE c:\users\payne\cam\inter.obj
-REPLACE c:\users\payne\cam\misc.obj
-REPLACE c:\users\payne\cam\other.obj
-REPLACE c:\users\payne\cam\output.obj
-REPLACE c:\users\payne\cam\profil.obj
-REPLACE c:\users\payne\cam\render.obj
-REPLACE c:\users\payne\cam\specs.obj
-REPLACE c:\users\payne\cam\utils.obj
-REPLACE c:\ckdev6\obj\ck_cdl.obj
-REPLACE c:\ckdev6\obj\ck_dlg.obj
-REPLACE c:\ckdev6\obj\ck_rdm.obj
-REPLACE c:\ckdev6\obj\ck_sys.obj
Appendix A: Source Code Listing

File: CAM.TRE

menu,synthesis,2
Synthesize Cam,F,cam_params,freeze,checkf
Modify Cam,F,edit_cam,freeze,checkf

menu,analysis,2
View Graphs,F,view_graphs,freeze,checkf
Kinematic Analysis,F,kinematic,freeze,checkf

menu,output,7
Start New Cam,F,start_new_cam,freeze,checkf
Engineering Drawing,F,eng Drawing,freeze,checkf
Report,F,report,freeze,checkf
Save SVAJ Data,F,save_svaj_data,freeze,checkf
Save Cam Data,F,save_cam_data,freeze,checkf
Save Drawing,F,drawing_save,freeze,checkf
Manufacturing (CL Data),F,manufacture,freeze,checkf
Appendix A: Source Code Listing

File: CALC.C

/***************************************************/
/* Cam Design System - Calculations for cam profile */
/***************************************************/
/* File: CALC.C */
/***************************************************/
/* Programmed by: */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* c/o Dr. Charles F. Reinholdt */
/* 106 Randolph Hall */
/* Department of Mechanical Engineering */
/* Virginia Tech */
/* Blacksburg, VA 24061-0238 */
/***************************************************/
/* Purpose: All calculations for the cam profile sva data is contained */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-09-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
/***************************************************/

/* necessary include files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ck_cadl.h" /* for standard CADL functions */
#include "ck_dlg.h" /* for dialog boxes */
#include "global.h" /* Cam Design System global variable header file */
#include "cam.h" /* Cam Design System main header file */

/***********************************************************/
void calculate_cam (void)
/***********************************************************/
/* Purpose: Calculates the cam profile, given all the data, and produces */
/* a rendering on the screen. */
/* */
/* Author: Steve Payne Date of creation: 07-09-93 */
DIALOG_BOX *db = NULL;
double theta, beta, theta_deg, beta_deg, h, velocity, total_angle;
double ratio, c0, c1, c2, c3, c4, c5, c6, c7, increment, factor;
double start_interval_lift, lift_total, other_lift_sum, old_lift;
double beginning_velocity;
int number_segments, d, curve_index;

/* initialize all coefficient values to 0 */
c0 = c1 = c2 = c3 = c4 = c5 = c6 = c7 = 0.0;

/* convert omega to rad/sec, and into new variable */
velocity = cam->omega * conv2;

/* set the angle increment value for the SVAJ data calculations */
increment = 0.50;

/* set the beginning value for velocity to 0.0 */
beginning_velocity = 0.0;

/* set the total angle value to 0 */
total_angle = 0.0;

/* set the beginning lift value to 0 */
h = 0.0;

/* set the starting lift value to 0 */
start_interval_lift = 0.0;
other_lift_sum = 0.0;

/* set the initial total lift value to 0 */
lift_total = 0.0;

/* set the old lift value (used when there is a dwell...keeps record of */
/* the most recent lift value) to 0 */
old_lift = 0.0;

/* get number of segments */
while (prof->next != NULL) {
    /* advance the pointer */
    prof = prof->next;
}

/* number of segments = index of last segment # */
number_segments = prof->seg_no;

/* if an oscillating follower (type 2 or 4), then convert */
/* stroke length value, h, to radians - set this value, factor */
if ((cam->type_num == 2) || (cam->type_num == 4))
{
    factor = conv1;
}
else
{
    factor = 1.0;
}

/* rewind the profile structure back to the beginning */
rewind_profile (&prof);

/* go through this loop only (number_segments) times */
for (d = 1; d <= number_segments; d++)
{
    /* Start of for loop */
    /* advance the pointer to the next profile segment of the structure */
    /* (unless it's the first segment in the profile list) */
    if (d != 1)
    {
        prof = prof->next;
    }

    /* set the beginning value for velocity (to match for cycloids) */
    if (d == 1)
    {
        beginning_velocity = 0.0;
    }
    else
    {
        beginning_velocity = curve->prev_one->γ;
    }

    /* set beta equal to the current angle range */
    beta_deg = prof->angle_range;
    beta = beta_deg * conv1;

    /* determine the value of the lift for this interval */
    /* also, set the value of the starting lift for this interval (since */
    /* the displacement needs to be continuous from interval to interval) */
    if (strcmp (prof->seg_type, 'D') != 0)
    {
        /* if the first segment in the list */
        if (prof->prev == NULL)
        {
            start_interval_lift = 0.0;
            other_lift_sum = prof->stroke_length * factor;
        }
        else
        {
            if (strcmp (prof->seg_type, "R") == 0)
{
    start_interval_lift += (old_lift * factor);
    other_lift_sum += (prof->stroke_length * factor);
}
else
{
    other_lift_sum -= (prof->stroke_length * factor);
    if (other_lift_sum == 0)
    {
        start_interval_lift = 0;
    }
    else
    {
        start_interval_lift = other_lift_sum;
    }
}

/* set the value of lift for this interval */
h = prof->stroke_length * factor;

/* sets the most recent non-dwell value of the lift */
/* also get values for the total lift */
if (strcmp (prof->seg_type, "R") == 0)
{
    old_lift = h;
    lift_total += h;
}
else
if (strcmp (prof->seg_type, "F") == 0)
{
    old_lift = 0;
    lift_total -= h;
}

/* if there is a new maximum lift, then set this in the cam structure */
if (fabs (lift_total) > cam->max_lift)
{
    cam->max_lift = lift_total;
}

/* set the theta value = 0.0 */
theta_deg = 0.0;
theta = 0.0;

/* get the curve_type_no index */
curve_index = prof->curve_type_no;

do {
    /* Start of do-while loop */
    if (strcmp (prof->seg_type, "D") != 0)
{  /* start of non-dwell if loop */
/* use this value ratio as the value of theta/beta, */
/* for simplicity */
ratio = theta / beta;

switch (curve_index)
{   /* Start of curve_index switch loop */
  case 1:
    {   /* Ramp function */
        if (strcmp (prof->seg_type, "R") == 0)
        {   
            curve->s = start_interval_lift + h * ratio;
            curve->v = h / beta;
            curve->a = 0.0;
            curve->j = 0.0;
        }
    }
    else
    {   
            curve->s = start_interval_lift + h * (1.0 - ratio);
            curve->v = -h / beta;
            curve->a = 0.0;
            curve->j = 0.0;
    }
    break;
  }
  /* End of Ramp Function */
  case 2:
    {   /* SHM Curve */
        if (strcmp (prof->seg_type, "R") == 0)
        {   /* Rise segment */
            curve->s = start_interval_lift + (h / 2.0) * (1.0 - cos (pi * ratio));
            curve->v = (pi / beta) * (h / 2.0) * sin (pi * ratio);
            curve->a = (((pi * pi) / (beta * beta)) * (h / 2.0) * cos (pi * ratio));
            curve->j = -(pow (pi, 3.0) / pow (beta, 3.0)) * (h / 2.0) * sin (pi * ratio);
        }
    }
    else
    {   /* Fall segment */
            curve->s = start_interval_lift + (h / 2.0) * (1.0 + cos (pi * ratio));
            curve->v = -(pi / beta) * (h / 2.0) * sin (pi * ratio);
            curve->a = -(((pi * pi) / (beta * beta)) * (h / 2.0) * cos (pi * ratio));
            curve->j = (pow (pi, 3.0) / pow (beta, 3.0)) * (h / 2.0) * sin (pi * ratio);
    }
    break;
  }
  /* End of SHM Curve */
  case 3:
    {   /* Half Cycloidal */

Appendix A: Source Code Listing - CALC.C  138
if (strcmp (prof->seg_type, "R") == 0)
    /* Rise Segment */
    if (fabs (beginning_velocity) <= 0.1)
        /* first half of rise (C-1) */
        curve->s = start_interval_lift + h * (ratio - (1.0 / pi) * sin (pi * ratio));
        curve->v = (h / beta) * (1.0 - cos (pi * ratio));
        curve->a = (pi * h / (beta * beta)) * sin (pi * ratio);
        curve->j = (pi * pi * h / (pow (beta, 3.0))) * cos (pi * ratio);
    }

else
    /* second half of rise (C-2) */
    curve->s = start_interval_lift + h * (ratio + (1.0 / pi) * sin (pi * ratio));
    curve->v = (h / beta) * (1.0 + cos (pi * ratio));
    curve->a = -(pi * h / (beta * beta)) * sin (pi * ratio);
    curve->j = -(pi * pi * h / (pow (beta, 3.0))) * cos (pi * ratio);
}
/* End of rise segment */

else
    /* Fall Segment */
    if (fabs (beginning_velocity) <= 0.1)
        /* first half of fall (C-3) */
        curve->s = start_interval_lift + h * (1.0 - ratio + (1.0 / pi) * sin (pi * ratio));
        curve->v = -(h / beta) * (1.0 - cos (pi * ratio));
        curve->a = -(pi * h / (beta * beta)) * sin (pi * ratio);
        curve->j = -(pi * pi * h / (pow (beta, 3.0))) * cos (pi * ratio);
    }

else
    /* second half of fall (C-4) */
    curve->s = start_interval_lift + h * (1.0 - ratio - (1.0 / pi) * sin (pi * ratio));
    curve->v = -(h / beta) * (1.0 + cos (pi * ratio));
    curve->a = (pi * h / (beta * beta)) * sin (pi * ratio);
    curve->j = (pi * pi * h / (pow (beta, 3.0))) * cos (pi * ratio);
}
/* End of fall segment */
break;
}

case 4:
    /* Modified Sine */
    if (strcmp (prof->seg_type, "K") == 0)
{ /* Rise Segment */
if ((theta >= 0.0) && (theta < 0.125 * beta))
{
    curve->s = start_interval_lift + h * (0.43990085 * 
        ratio - 0.0350062 * sin (4.0 * pi * ratio));
    curve->v = 0.43990085 * (h / beta) * (1.0 - cos (4.0 * 
        pi * ratio));
    curve->a = 5.5279571 * h / (beta * beta) * sin (4.0 * 
        pi * ratio);
    curve->j = 69.4663577 * (h / pow (beta, 3.0)) * cos 
        (4.0 * pi * ratio);
}
else
if ((theta >= 0.125 * beta) && (theta < 0.875 * beta))
{
    curve->s = start_interval_lift + h * (0.28004957 + 
        0.43990085 * ratio - 0.31505577 * cos ((4.0 * pi / 
        3.0) * ratio - (pi / 6.0)));
    curve->v = 0.43990085 * (h / beta) * (1.0 + 3.0 * sin 
        ((4.0 * pi / 3.0) * ratio - (pi / 6.0)));
    curve->a = 5.5279571 * h / (beta * beta) * cos ((4.0 * 
        pi / 3.0) * ratio - (pi / 6.0));
    curve->j = -23.1553 * (h / pow (beta, 3.0)) * sin 
        ((4.0 * pi / 3.0) * ratio - (pi / 6.0));
}
else
if ((theta >= 0.875 * beta) && (theta <= beta))
{
    curve->s = start_interval_lift + h * (0.56009915 + 
        0.43990085 * ratio - 0.0350062 * sin (2.0 * pi * 
        (2.0 * ratio - 1.0)));
    curve->v = 0.43990085 * (h / beta) * (1.0 - cos (2.0 * 
        pi * (2.0 * ratio - 1.0)));
    curve->a = 5.5279571 * h / (beta * beta) * sin (2.0 * 
        pi * (2.0 * ratio - 1.0));
    curve->j = 69.4663577 * (h / pow (beta, 3.0)) * cos 
        (2.0 * pi * (2.0 * ratio - 1.0));
}
} /* End of rise segment */
else
{ /* Fall segment */
if ((theta >= 0.0) && (theta < 0.125 * beta))
{
    curve->s = start_interval_lift + h * (1.0 - 0.43990085 
        * ratio + 0.0350062 * sin (4.0 * pi * ratio));
    curve->v = -0.43990085 * (h / beta) * (1.0 - cos (4.0 
        * pi * ratio));
    curve->a = -5.5279571 * h / (beta * beta) * sin (4.0 * 
        pi * ratio);
    curve->j = -69.4663577 * (h / pow (beta, 3.0)) * cos 
        (4.0 * pi * ratio);
}
} else
    if ((theta >= 0.125 * beta) && (theta < 0.875 * beta))
    {
        curve->s = start_interval_lift + h * (0.71995043 - 
                 0.43990085 * ratio + 0.31505577 * cos ((4.0 * pi / 
                 3.0) * ratio - (pi / 6.0)));
        curve->v = -0.43990085 * (h / beta) * (1.0 + 3.0 * sin 
                 ((4.0 * pi / 3.0) * ratio - (pi / 6.0)));
        curve->a = -5.5279571 * h / (beta * beta) * cos ((4.0 
                 * pi / 3.0) * ratio - (pi / 6.0));
        curve->j = 23.1553 * (h / pow (beta, 3.0)) * sin ((4.0 
                 * pi / 3.0) * ratio - (pi / 6.0));
    }
    else
    if ((theta >= 0.875 * beta) && (theta <= beta))
    {
        curve->s = start_interval_lift + h * (0.43990085 - 
                 0.43990085 * ratio + 0.0350062 * sin (2.0 * pi * 
                 (2.0 * ratio - 1.0)));
        curve->v = -0.43990085 * (h / beta) * (1.0 - cos (2.0 
                 * pi * (2.0 * ratio - 1.0)));
        curve->a = -5.5279571 * h / (beta * beta) * sin (2.0 * 
                 pi * (2.0 * ratio - 1.0));
        curve->j = -69.4663577 * (h / pow (beta, 3.0)) * cos 
                 (2.0 * pi * (2.0 * ratio - 1.0));
    }
} /* End of fall segment */
break;
} /* End of Modified Sine */

case 5:
    { /* 3-4-5 Curve */
      if (strcmp (prof->seg_type, "R") == 0)
        { /* Rise Segment */
          c0 = c1 = c2 = 0;
          c3 = 10.0;
          c4 = -15.0;
          c5 = 6.0;
        }
      else
        { /* Fall segment */
          c0 = 1.0;
          c1 = c2 = 0;
          c3 = -10.0;
          c4 = 15.0;
          c5 = -6.0;
        }
        curve->s = start_interval_lift + h * (c0 + c1 * ratio + c2 * 
                pow (ratio, 2.0) + c3 * pow (ratio, 3.0) + c4 * pow 
                (ratio, 4.0) + c5 * pow (ratio, 5.0));
        curve->v = (h / beta) * (c1 + 2.0 * c2 * ratio + 3.0 * c3 *
pow (ratio, 2.0) + 4.0 * c4 * pow (ratio, 3.0) + 5.0 * c5
* pow (ratio, 4.0));
curve->a = (h / (beta * beta)) * (2.0 * c2 + 6.0 * c3 *
  ratio + 12.0 * c4 * pow (ratio, 2.0) + 20.0 * c5 * pow
  (ratio, 3.0));
curve->j = (h / pow (beta, 3.0)) * (6.0 * c3 + 24.0 * c4 *
  ratio + 60.0 * c5 * pow (ratio, 2.0));
break;
} /* End of 3-4-5 Curve */
case 6:
 { /* Trapezoidal Curve */
 if (strcmp (prof->seg_type, "R") == 0)
 { /* Rise Segment */
 if (((theta >= 0) && (theta < 0.125 * beta))

 { curve->s = start_interval_lift + (64.0 / 9.0) * h *
    pow (ratio, 3.0);
  curve->v = (64.0 / 3.0) * (h / beta) * pow (ratio,
    2.0);
  curve->a = (128.0 / 3.0) * (h / pow (beta, 2.0)) *
    ratio;
  curve->j = (128.0 / 3.0) * (h / pow (beta, 3.0));
 } else
 if (((theta >= 0.125 * beta) && (theta < 0.375 * beta))

 { curve->s = start_interval_lift + h * ((1.0 / 72.0) -
    ((1.0 / 3.0) * ratio) + (8.0 / 3.0) * pow (ratio,
    2.0));
  curve->v = (h / beta) * ((-1.0 / 3.0) + (16.0 / 3.0) *
    ratio);
  curve->a = (h / (beta * beta)) * (16.0 / 3.0);
  curve->j = 0.0;
 } else
 if (((theta >= 0.375 * beta) && (theta < 0.625 * beta))

 { curve->s = start_interval_lift + h * ((7.0 / 18.0) -
    (10.0 / 3.0) * ratio + (32.0 / 3.0) * pow (ratio,
    2.0) - (64.0 / 9.0) * pow (ratio, 3.0));
  curve->v = (h / beta) * ((-10.0 / 3.0) + (64.0 / 3.0)
    * ratio) - (64.0 / 3.0) * pow (ratio, 2.0));
  curve->a = (h / (beta * beta)) * ((64.0 / 3.0) -
    (((128.0 / 3.0) * ratio));
  curve->j = (-128.0 / 3.0) * (h / pow (beta, 3.0));
 } else
 if (((theta >= 0.625 * beta) && (theta < 0.875 * beta))

 { curve->s = start_interval_lift + h * (-8.0 / 3.0) *
    pow (ratio, 2.0) + 5.0 * ratio - (97.0 / 72.0));

Appendix A: Source Code Listing - CALCC
curve->v = (h / beta) * (-(16.0 / 3.0) * ratio + 5.0);
curve->a = -(16.0 / 3.0) * (h / pow(beta, 2.0));
curve->j = 0.0;
}
else

if ((theta >= 0.875 * beta) && (theta <= beta))
{
    curve->s = start_interval_lift + h * ((64.0 / 9.0) * pow(ratio, 3.0) - (64.0 / 3.0) * pow(ratio, 2.0) + (64.0 / 3.0) * ratio - (55.0 / 9.0));
curve->v = (h / beta) * ((64.0 / 3.0) * pow(ratio, 7.0) - (128.0 / 3.0) * ratio + (64.0 / 3.0));
curve->a = (h / (beta * beta)) * ((128.0 / 3.0) * ratio - (128.0 / 3.0));
curve->j = (128.0 / 3.0) * (h / pow(beta, 3.0));
}
else

    /* End of rise segment */
else

    /* Fall Segment */
if ((theta >= 0) && (theta < 0.125 * beta))
{
    curve->s = start_interval_lift + h * (1.0 - (64.0 / 9.0) * pow(ratio, 3.0));
curve->v = -(64.0 / 3.0) * (h / beta) * pow(ratio, 2.0);
curve->a = -(128.0 / 3.0) * (h / pow(beta, 2.0)) * ratio;
curve->j = -(128.0 / 3.0) * (h / pow(beta, 3.0));
}
else

    if ((theta >= 0.125 * beta) && (theta < 0.375 * beta))
    {
        curve->s = start_interval_lift + h * ((71.0 / 72.0) + (1.0 / 3.0) * ratio - (8.0 / 3.0) * pow(ratio, 2.0));
curve->v = (h / beta) * ((1.0 / 3.0) - (16.0 / 3.0) * ratio);
curve->a = -(h / (beta * beta)) * (16.0 / 3.0);
curve->j = 0.0;
    }
else

    if ((theta >= 0.375 * beta) && (theta < 0.625 * beta))
    {
        curve->s = start_interval_lift + h * ((11.0 / 18.0) + (10.0 / 3.0) * ratio - (32.0 / 3.0) * pow(ratio, 2.0) + (64.0 / 9.0) * pow(ratio, 3.0));
curve->v = (h / beta) * ((10.0 / 3.0) - (64.0 / 3.0) * ratio + (64.0 / 3.0) * pow(ratio, 2.0));
curve->a = (h / (beta * beta)) * (-64.0 / 3.0) + ((128.0 / 3.0) * ratio));
curve->j = (128.0 / 3.0) * (h / pow(beta, 3.0));
    }
else

        if ((theta >= 0.625 * beta) && (theta < 1.0))
        {
            curve->s = start_interval_lift + h * (1.0 - (64.0 / 9.0) * pow(ratio, 3.0));
curve->v = -(64.0 / 3.0) * (h / beta) * pow(ratio, 2.0);
curve->a = -(128.0 / 3.0) * (h / pow(beta, 2.0)) * ratio;
curve->j = -(128.0 / 3.0) * (h / pow(beta, 3.0));
        }
    }
else
  if ((theta >= 0.625 * beta) && (theta < 0.875 * beta))
  {
    curve->s = start_interval_lift + h * ((8.0 / 3.0) *
      pow (ratio, 2.0) - 5.0 * ratio + (169.0 / 72.0));
    curve->v = (h / beta) * ((16.0 / 3.0) * ratio - 5.0);
    curve->a = (16.0 / 3.0) * (h / pow (beta, 2.0));
    curve->j = 0.0;
  }
  else
  if ((theta >= 0.875 * beta) && (theta <= beta))
  {
    curve->s = start_interval_lift + h * (-64.0 / 9.0) *
      pow (ratio, 3.0) + (64.0 / 3.0) * pow (ratio, 2.0) -
      (64.0 / 3.0) * ratio + (64.0 / 9.0));
    curve->v = (h / beta) * (-64.0 / 3.0) * pow (ratio,
      2.0) + (128.0 / 3.0) * ratio - (64.0 / 3.0));
    curve->a = (h / (beta * beta)) * (-128.0 / 3.0) *
      ratio + (128.0 / 3.0));
    curve->j = -(128.0 / 3.0) * (h / pow (beta, 3.0));
  }
  } /* End of fall segment */
break;
} /* End of Trapezoidal Curve */

case 7:
{
  /* Cycloidal Displacement */
  if (strcmp (prof->sec_type, "R") == 0)
  {
    /* Rise Segment */
    curve->s = start_interval_lift + h * (ratio - (1.0 / (2.0 *
      pi)) * sin (2.0 * pi * ratio));
    curve->v = (h / beta) * (1.0 - cos (2.0 * pi * ratio));
    curve->a = (2.0 * pi * h) / (beta * beta) * sin (2.0 * pi *
      ratio);
    curve->j = (4.0 * pi * pi * h) / (pow (beta, 3.0)) * cos
      (2.0 * pi * ratio);
  }
else
  {
    /* Fall Segment */
    curve->s = start_interval_lift + h * (1.0 - ratio + (1.0
      / (2.0 * pi)) * sin (2.0 * pi * ratio));
    curve->v = -(h / beta) * (1.0 - cos (2.0 * pi * ratio));
    curve->a = -(2.0 * pi * h) / (beta * beta) * sin (2.0 *
      pi * ratio);
    curve->j = -(4.0 * pi * pi * h) / (pow (beta, 3.0)) * cos
      (2.0 * pi * ratio);
  }
break;
} /* End of Cycloidal Displacement */

case 8:
{
  /* Modified Trapezoidal */

Appendix A: Source Code Listing - CALC.C 144
if (strcmp (prof->seg_type, "R") == 0)
{
    /* Rise segment */
    if ((theta >= 0.0) && (theta < 0.125 * beta))
    {
        curve->s = start_interval_lift + h * (0.38898448 *
            ratio - 0.0309544 * sin (4.0 * pi * ratio));
        curve->v = 0.38898448 * h / beta * (1.0 - cos (4.0 *
            pi * ratio));
        curve->a = 4.888124 * h / (beta * beta) * sin (4.0 *
            pi * ratio);
        curve->j = 61.425769 * h / (pow (beta, 3.0)) * cos
            (4.0 * pi * ratio);
    }
    else
    if ((theta >= 0.125 * beta) && (theta < 0.375 * beta))
    {
        curve->s = start_interval_lift + h * (2.44406184 * pow
            (ratio, 2.0) - 0.22203097 * ratio + 0.00723047);
        curve->v = h / beta * (4.888124 * ratio - 0.22203097);
        curve->a = 4.888124 * h / (beta * beta);
        curve->j = 0.0;
    }
    else
    if ((theta >= 0.375 * beta) && (theta < 0.625 * beta))
    {
        curve->s = start_interval_lift + h * (1.6110154 *
            ratio - 0.0309544 * sin (4.0 * pi * ratio - pi) -
            0.0305077);
        curve->v = (h / beta) * (1.6110154 - 0.38898448 * cos
            (4.0 * pi * ratio - pi));
        curve->a = 4.888124 * h / (beta * beta) * sin (4.0 *
            pi * ratio - pi);
        curve->j = 61.425769 * h / (pow (beta, 3.0)) * cos
            (4.0 * pi * ratio - pi);
    }
    else
    if ((theta >= 0.625 * beta) && (theta < 0.875 * beta))
    {
        curve->s = start_interval_lift + h * (-2.44406184 *
            pow (ratio, 2.0) + 4.6660917 * ratio - 1.2292648);
        curve->v = (h / beta) * (-4.888124 * ratio +
            4.6660917);
        curve->a = -4.888124 * h / (beta * beta);
        curve->j = 0.0;
    }
    else
    if ((theta >= 0.875 * beta) && (theta <= beta))
    {
        curve->s = start_interval_lift + h * (0.6110154 +
            0.38898448 * ratio + 0.0309544 * sin (4.0 * pi *
            ratio - 3.0 * pi));
    }
curve->v = 0.38898448 * (h / beta) * (1.0 + cos (4.0 * 
pi * ratio - 3.0 * pi));
curve->a = -4.888124 * h / (beta * beta) * sin (4.0 * 
pi * ratio - 1.0 * pi);
curve->j = -61.425769 * h / (pow (beta, 3.0)) * cos 
(4.0 * pi * ratio - 3.0 * pi);
}
/* End cf rise segment */
else

/* Fall Segment */
if ((theta >= 0.0) && (theta < 0.125 * beta))
{
    curve->s = start_interval_lift + a * (1.0 - 0.38898448 
* ratio + 0.0309544 * sin (4.0 * pi * ratio));
curve->v = -0.38898448 * h / beta * (1.0 - cos (4.0 * 
pi * ratio));
curve->a = -4.888124 * h / (beta * beta) * sin (4.0 * 
pi * ratio);
curve->j = -61.425769 * h / (pow (beta, 3.0)) * cos 
(4.0 * pi * ratio);
}
else
if ((theta >= 0.125 * beta) && (theta < 0.375 * beta))
{
    curve->s = start_interval_lift + h * (0.99276593 - 2.44406184 * pow (ratio, 2.0) + 0.22203097 * 
ratio);
curve->v = -h / beta * (4.888124 * ratio - 
0.22203097);
curve->a = -4.888124 * h / (beta * beta);
curve->j = 0.0;
}
else
if ((theta >= 0.375 * beta) && (theta < 0.625 * beta))
{
    curve->s = start_interval_lift + h * (1.3055077 - 1.6110154 * ratio + 0.0309544 * sin (4.0 * pi * 
ratio - pi));
curve->v = -(h / beta) * (1.6110154 - 0.38898448 * cos 
(4.0 * pi * ratio - pi));
curve->a = -4.888124 * h / (beta * beta) * sin (4.0 * 
pi * ratio - pi);
curve->j = -61.425769 * h / (pow (beta, 3.0)) * cos 
(4.0 * pi * ratio - pi);
}
else
if ((theta >= 0.625 * beta) && (theta < 0.875 * beta))
{
    curve->s = start_interval_lift + h * (2.2292648 + 2.44406184 * pow (ratio, 2.0) - 4.6666917 * ratio);
curve->v = -(h / beta) * (-4.888124 * ratio +

Appendix A: Source Code Listing - CALC.C

146
4.6660917);  
    curve->a = 4.888124 * h / (beta * beta);  
    curve->j = 0.0;  
  }  
else  
  if ((theta >= 0.875 * beta) && (theta <= beta))  
  {  
    curve->s = start_interval_lift + h * (-0.38898448 -  
      0.38898448 * ratio - 0.0309544 * sin (4.0 * pi *  
      ratio - 3.0 * pi));  
    curve->v = -0.38898448 * (h / beta) * (1.0 + cos (4.0  
      * pi * ratio - 3.0 * pi));  
    curve->a = 4.888124 * h / (beta * beta) * sin (4.0 *  
      pi * ratio - 3.0 * pi);  
    curve->j = 61.425769 * h / (pow (beta, 3.0)) * cos  
      (4.0 * pi * ratio - 3.0 * pi);  
  }  
  /* End cf fall segment */  
break;  
}  
/* End of Modified Trapezoidal */  
case 9:  
  {  
    /* 4-5-6-7 Curve */  
    if (strcmp (prof->seg_type, "K") == 0)  
    {  
      /* Rise Segment */  
      c0 = c1 = c2 = c3 = 0;  
      c4 = 35.0;  
      c5 = -84.0;  
      c6 = 70.0;  
      c7 = -20.0;  
    }  
else  
    {  
      /* Fall segment */  
      c0 = 1.0;  
      c1 = c2 = c3 = 0;  
      c4 = -35.0;  
      c5 = 84.0;  
      c6 = -70.0;  
      c7 = 20.0;  
    }  
  
    curve->s = start_interval_lift + h * (c0 + c1 * ratio + c2 *  
      pow (ratio, 2.0) + c3 * pow (ratio, 3.0) + c4 * pow  
      (ratio, 4.0) + c5 * pow (ratio, 5.0) + c6 * pow (ratio,  
      6.0) + c7 * pow (ratio, 7.0));  
    curve->v = (h / beta) * (c0 + c1 * ratio + c2 *  
      pow (ratio, 2.0) + c3 * pow (ratio, 3.0) + c4 * pow  
      (ratio, 4.0) + c5 * pow (ratio, 5.0) + c6 * pow (ratio,  
      6.0) + c7 * pow (ratio, 7.0));  
    curve->a = (h / (beta * beta)) * (2.0 + c2 + 6.0 * c3 *  
      ratio + 12.0 * c4 * pow (ratio, 2.0) + 20.0 * c5 * pow  
      (ratio, 3.0) + 30.0 * c6 * pow (ratio, 4.0) + 42.0 * c7 *  
      pow (ratio, 5.0));
curve->j = (\lambda / \text{pow}(\beta, 3.0)) \times (6.0 \times c3 + 24.0 \times c4 \times 
\text{ratio} + 60.0 \times c5 \times \text{pow}(\text{ratio}, 2.0) + 120.0 \times c6 \times \text{pow}(\text{ratio}, 3.0) + 210.0 \times c7 \times \text{pow}(\text{ratio}, 4.0));
break;
}   /* End of 4-5-6-7 Curve */
}   /* End of curve_index switch loop */
} */ end of non-dwell if loop */
else
{   /* this is a dwell curve */
if (curve->prev_one == NULL)

{   /* if the first interval in the profile, then set */
   /* the s here to 0.0 for the duration of the dwell */
   curve->s = 0.0;
}
else

{   /* the dwell lift during this interval is a constant */
   /* and is equal to the lift at the end of the previous */
   /* interval */
   curve->s = curve->prev_one->s;
}
curve->v = curve->a = curve->j = 0.0;
}
/* set the index for the curve data (the theta value) */
curve->index = total_angle;

/* increment the total angle counter */
total_angle += increment;

/* increment theta value slowly (by 'increment' degrees) */
theta_deg += increment;
theta += (increment * conv1);

/* set the color to plot this segment of the cam profile in */
curve->color = (d + 1);

if (total_angle < 360.0)

{   /* start of if loop */
   /* set up for dynamic memory allocation of the structure */
   curve->next_one = (struct avaj*) malloc(sizeof(struct avaj));

   /* give a reference for the previous profile values */
   curve->next_one->prev_one = curve;

   /* advances pointer, so that we can add another profile */
   /* to the profile database structure */
   curve = curve->next_one;
}   /* end of if loop */
}   /* End of do-while loop */
while (theta_deg < beta_deg);
}   /* End of for loop */
/* set this as the last profile element in the structure */
curve->next_one = NULL;

/* rewind the cam profile data structure to the beginning */
rewind_profile (&prof);

/* rewind the svaj/curve data structure to the beginning */
rewind_curve (&curve);

/* if not in edit mode */
if (edit_flag != 1)
{
    /* go to the function to actually draw a rendering of the cam on */
    /* the CADKEY screen */
    can_render (&curve);

    /* rewind the svaj/curve data structure to the beginning */
    rewind_curve (&curve);
}

    /* end of calculate_cam */
Appendix A: Source Code Listing

File: INTER.C

/******************************************************************************
/* Cam Design System - User Interface & Menu Structure */
/******************************************************************************
/* File: INTER.C */
/******************************************************************************
/* Programmed by: */
/* */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* c/o Dr. Charles F. Reinholdt */
/* 106 Randolph Hall */
/* Department of Mechanical Engineering */
/* Virginia Tech */
/* Blacksburg, VA 24061-0238 */
/******************************************************************************
/* Purpose: Sets up the CADKEY graphics environment working variables */
/* and sets up the functions to handle the menuing system. */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-01-94 */
/******************************************************************************

/* necessary include files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ck_rdm.h" /* for pull-down menus */
#include "ck_cdl.h" /* for standard CADL functions */
#include "ck dlg.h" /* for dialog boxes */
#include "ck sys.h" /* for system commands */
#include "cam.h" /* Cam Design System main header file */

/* initialize miscellaneous global variables (defined as extern */
/* in GLOBAL.H) */
DIALOG_BOX *bar = NULL;
void *menu = NULL;
char prompt[65], input_drive[3], input_path[35], motion_profile[10];
double conv1, conv2, scale_cam;
int cam_defined, nc_flag, edit_flag;
struct profile *prof;
struct the_can *cam;
struct svaj *curve;
struct kine_anal *analysis;
struct pressure_analysis *pressure;
struct nc_data *nc;

/*****************************
void Cam_Design_System (void)
******************************/

/*******************************************************************/
// Purpose: Serves as the main() for the program. Provides for the */
// initial (pull-down menu) setup (through the use of the */
// external file 'cam.cre'), and the initialization of the main */
// cam profile structure. */
//*/
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-01-94 */
/**********************************************************************/
{
    /* start of main (Cam_Design_System) */
    int min;
    FILE *file;
    char name[50];
    CK_ENTATT *e_att;

    /* these three lines open a file (called "cmapth.cam") to read in */
    /* the correct path for the cam program */
    file = fopen("cmapth.cam", "r");
    fscanf (file, "%s", input_drive);
    fscanf (file, "%s", input_path);
    fclose (file);

    /* set up the pathname */
    strcpy (name, input_drive);
    strcat (name, input_path);

    /* set up values for conversion constants */
    conv1 = pi / 180.0;
    conv2 = 2.0 * pi / 60.0;

    /* set up for initial dynamic memory allocation of the structures */
    prof = (struct profile *) malloc (sizeof(struct profile));
    cam = (struct the_can *) malloc (sizeof(struct the_can));
    curve = (struct svaj *) malloc (sizeof(struct svaj));
    analysis = (struct kine_anal *) malloc (sizeof(struct kine_anal));
    pressure = (struct pressure_analysis *) malloc (sizeof(struct
        pressure_analysis));

    /* first profile data in list */

Appendix A: Source Code Listing - INTER.C

151
prof->prev = prof->next = NULL;

/* first curve data in list */
curve->prev_one = curve->next_one = NULL;

/* first pressure angle data in list */
pressure->prev = pressure->next = NULL;

/* set cam_defined flag to 0 (no cam fully defined yet) */
cam_defined = 0;

/* set the nc flag to 0 (to NC data to be exported) */
nc_flag = 0;

/* set the edit flag to 0 */
edit_flag = 0;

/* set the max lift value = 0 */
cam->max_lift = 0.0;

/* set the default cam scale value to 1.0 */
scale_cam = 1.0;

/* set the default cam hub_type to 1 */
cam->hub_type = 1;

/* set default motion profile as nothing */
strcpy (motion_profile, "");

/* set the default follower type to Translating Flat-Faced */
cam->type_num = 1;
strcpy (cam->follower_type, "Translating Flat-Faced");

/* set the default angular rotation of the cam to CCW (-) */
cam->omega = -1.0;

/* set the default base circle radius to 0.0 */
analysis->k1 = 0.0;

/* set the default maximum pressure angle to 0.0 */
analysis->k6 = 0.0;

/* set all other analysis structure values to 0.0 */
analysis->k2 = analysis->k3 = analysis->k4 = analysis->k5 = analysis->k7 = 0.0;

/* set CADKEY screen for world coordinates */
ck_set (CK_SET_COORD, 1);

/* set CADKEY screen to track world coordinates */
ck_set (CK_SET_CURTRACK, 1);
/* turn off CADKEY display axes (in all viewports) */
ck_set (CK_SET_DSPAXES, 0);

/* turn off the CADKEY display grid */
ck_set (CK_SET_GRID, 0);

/* set the default CADKEY note angle to 0 */
ck_set (CK_SET_NOTEANG, 0.0);

/* set the default CADKEY note fill mode to no-fill */
ck_set (CK_SET_NOTEFILL, 0);

/* set the default CADKEY note font to 1 */
ck_set (CK_SET_NOTEFONT, 1);

/* set the default CADKEY note slant to 0 */
ck_set (CK_SET_NOTESLANT, 0);

/* set the default CADKEY note mode to non-underlining */
ck_set (CK_SET_NOTEULINE, 0);

/* set the default CADKEY note height */
ck_set (CK_SET_NOTETH, 5.0);

/* set the default CADKEY note text line spacing factor */
ck_set (CK_SET_NOTELINE, 0.60);

/* set the default CADKEY note font (BOLD) */
ck_set (CK_SET_NOTEFONT, 2);

/* set the default CADKEY note text fill mode to on */
ck_set (CK_SET_NOTEFILL, 1);

/* set the default CADKEY dimension height */
ck_set (CK_SET_DIMHT, 8.0);

/* set the default CADKEY arrow style to filled (#2) */
ck_set (CK_SET_ARRSTYLE, 2);

/* set the default CADKEY arrow direction to out */
ck_set (CK_SET_ARRDIRE, 1);

/* everything drawn on level #5 in CADKEY (just for convenience, and we */
/* don't draw over top of something already on level #1, for instances) */
ck_set (CK_SET_LEVEL, 5);

/* create all parts with pen #5 (again for convenience) */
/* (used mainly for plotting) */
ck_set (CK_SET_PEN, 5);
/* set the default CADKEY color to 15 (white) */
ck_set(CK_SET_COLOR, 15);

/* set the line type to solid lines (type 1) */
ck_set(CK_SET_LINETYPE, 1);

/* set the line width to normal (width = 1) */
ck_set(CK_SET_LINEWIDTH, 1);

/* set all CADKEY units to mm */
ck_set(CK_SET_UNIT, 1);

/* set the default button border width */
dg_set_button_border(2);

/* normal CADKEY mode for recognizing data functions */
ck_mode(CK_NORMAL);

/* set the attribute entity structure to what is currently set */
ck_set(CK_SET_SYSDFLT, &t_info, &d_info, &da_info, &nt_att, &dt_att, &ae_att);

/* set the initial location of the CADKEY graphics cursor */
ck_setcur(0.0, 0.0, CK_PRIME_VP);

/* set the current window viewport dimension (in mm) */
ck_set(CK_SET_WINDOW, CK_PRIME_VP, -100.0, -100.0, 100.0, 100.0);

/* set up the viewport scale and origin */
ck_set(CK_SET_SCALE, CK_PRIME_VP, 1.0, 0.0, C.0);

/* erase the screen before the start */
erase_screen(CK_PRIME_VP);

/* develop and generate overhead modeless dialog box */
if (bar == NULL)
{
    bar = dg_modeless_dialog(0, 3, &min, 50, NULL, DG_RED, DG_BLUE,
                          DG_CANCOLOR, "run_toolbar");
}

dg_add_note(bar, 1, 0.50, 0.65,
"CAM DESIGN SYSTEM ",
DG_WHITE, DG_BLUE, 0);
dg_add_note(bar, 2, 1.50, 2.0, "Synthesis", DG_WHITE, DG_BLUE,
DG_RET_ON_SEL);
dg_add_note(bar, 3, 1.50, 18.0, "Analysis", DG_WHITE, DG_BLUE,
DG_RET_ON_SEL);
dg_add_note(bar, 4, 1.50, 35.0, "Output", DG_WHITE, DG_BLUE,
DG_RET_ON_SEL);
dg_add_note(bar, 5, 1.50, 58.0, "Exit", DG_WHITE, DG_BLUE,
DG_RET_ON_SEL);
dg_add_box (bar, 6, 0.40, 15.65, 0.95, 35.0, NULL, DG_WHITE, DG_BLUE, 0);
dg_add_box (bar, 7, 0.38, 15.55, 0.99, 35.3, NULL, DG_WHITE, DG_BLUE, 0);

    /* draw the dialog box */
    dg_draw_dialog (bar);

    /* use the default path name to get the cam tree file */
    strcat (name, "cam.tre");

    /* read in the menu tree structure from the file cam.tre */
    menu = read_menu_tree (name);
}
           /* end of main (Cam_Design_System) */

/***************************************************************************/
void run_toolbar (DIALOG_BOX *dud)
/***************************************************************************/
/* Purpose: Allows for the execution of the dg_modeless dialog box. */
/* Most of this routine has been ported from CADKEY SDK v6.0 */
/* Manual Example, pp. 9-212 - 9-213. */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 02-12-94 */
/***************************************************************************/
{
    int opt;

    opt = dg_run_dialog (dud);
    if (((sys_get_state (SYS_ST_LEAF_NODE)) ||
         (sys_get_state (SYS_ST_IMMED_NODE)))
        
    return;
    }

    sys_push_menu();
    ck_prompt (" ");

    switch (opt)
    {
        /* start of opt switch loop */
    case 2:
    {
        run_menu_tree (menu, "synthesis", bar, 2, 0.0, 3.0);
        break;
    }
    case 3:
    {
        run_menu_tree (menu, "analysis", bar, 2, 0.0, 18.0);
        break;
    }
case 4:
{
    run_menu_tree (menu, "output", bar, 2, 0.0, 35.0);
    break;
}

case 5:
{
    if (menu != NULL)
    {
        if (!sys_get_state (SYS_ST_EXITING))
        {
            free_menu_tree (menu);

            /* erase dialog box and free up memory */
            dg_erase_dialog (bar);
            dg_free_dialog (bar);

            /* clear these variables so that the program will */
            /* run properly the next time (since the CDB is */
            /* still loaded in the Applications menu) */
            menu = NULL;
            bar = NULL;
        }
    }

    /* free memory taken by structures */
    if (prof != NULL)
    {
        free (prof);
    }
    if (cam != NULL)
    {
        free (cam);
    }
    if (curve != NULL)
    {
        free (curve);
    }
    if (analysis != NULL)
    {
        free (analysis);
    }
    if (pressure != NULL)
    {
        free (pressure);
    }
    break;
}

}  /* end of opt switch loop */

}  /* end of run_toolbar */
Appendix A: Source Code Listing

File: MISC.C

/***************************************************************************/
/* Can Design System - Other Miscellaneous Functions */
/***************************************************************************/
/* File: MISC.C */
/***************************************************************************/
/* Programmed by: */
/* */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* c/o Dr. Charles F. Keinholz */
/* 106 Randolph Hall */
/* Department of Mechanical Engineering */
/* Virginia Tech */
/* Blacksburg, VA 24061-0238 */
/***************************************************************************/
/* Purpose: These are other miscellaneous functions for the Cam Design */
/* System program is contained here. */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-03-94 */
/***************************************************************************/

/* necessary include files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ck_cdl.h" /* for standard CADr, functions */
#include "ck_dlg.h" /* for dialog boxes */
#include "global.h" /* Cam Design System global variable header file */
#include "cam.h" /* Cam Design System main header file */

/***************************************************************************/
void kinematic (void)
/***************************************************************************/
/* Purpose: Performs kinematic analysis on the cam. */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-03-94 */
/***************************************************************************/
{  
    /* start of kinematic */
    DIALOG_BOX *db = NULL;
    int    choice, d;
    double pressure_min, x_factor, y_factor, dd, tick;
    double pt1x, pt1y, pt2x, pt2y;

    /* if there is no structure then return */
    if ((curve->next_one == NULL)  
        {  
        user_warning();
        return;
        }

    /* initialize dialog box choice flag */
    choice = 0;

    /* create new dialog box for analysis results */
    db = dq_init_dialog (7, 34, 3, "Analysis Results", 0, 0, 4, 0);
    sprintf (prompt, "MAX Pressure Angle, \( \alpha \): \( %.2fs \), analysis->k6);
    dg_add_note (db, 0, 0.75, 2.00, prompt, 0, 0, 0);
    sprintf (prompt, "At Cam Angle, \( \theta \): \( %.2fs \), analysis->k7);
    dg_add_note (db, 1, 2.65, 8.00, prompt, 0, 0, 0);
    dg_add_button (db, 2, 5.00, 13.00, 8, "OK", 0, 0, DG_RET_ON_SEL);

    /* draw the dialog box */
    dg_draw_dialog (db);

    do {  /* start of do-while loop */
        /* run the dialog box */
        choice = dg_run_dialog (db);
    }  /* end of do-while loop */
    while (choice != 2);

    /* erase and free memory from dialog box */
    dq_erase_dialog (db);
    dg_free_dialog (db);

    /* now graph the pressure angle vs. theta for the full cam profile */

    /* autoscale and redraw the cam before erasing it (to be sure to get */
    /* the whole cam view in the viewport) */
    ck_set (CK_SET_SCALE, CK(PRIME VP, scale_cam, 0.0, 0.0));
    ck_redraw (CK(PRIME VP));

    /* erase the screen before starting */
    erase_screen (CK (PRIME VP));

    /* find the minimum pressure angle (for proper scaling) */
    pressure_min = pressure->p_angle;

Appendix A: Source Code Listing - MISC.C
while (pressure->next != NULL)
    { /* start of while loop */
        /* advance the pointer */
        pressure = pressure->next;

        if (pressure->p_angle < pressure_min)
            { pressure_min = pressure->p_angle; }
    } /* end of while loop */

/* rewind pressure angle structure */
rewind_pressure (&pressure);

/* set up scale factors for graphing the SWAJ data */
x_factor = (183.0) / (360.0);
if (fabs (analysis->k6 - pressure_min) < 10.0)
    { y_factor = 175.0 / 10.0 / 4.0; }
else
    { y_factor = (175.0) / fabs (analysis->k6 - pressure_min) / 4.0; }

/* draw all the x and y axes, in color 1 */
ck_set (CK_SET_COLOR, 1);
ck_line (-98.0, -90.0, 0.0, -98.0, 90.0, 0.0, NULL);
ck_line (-99.0, 0.0, 0.0, 90.0, 0.0, NULL);
ck_line (-98.0, -90.0, 0.0, -100.0, -94.0, 0.0, NULL);
ck_line (-98.0, -90.0, 0.0, -56.0, -84.0, 0.0, NULL);
ck_line (-98.0, 90.0, 0.0, -100.0, 84.0, 0.0, NULL);
ck_line (-98.0, 90.0, 0.0, -96.0, 84.0, 0.0, NULL);
ck_line (90.0, 0.0, 0.0, 84.0, 2.0, 0.0, NULL);
ck_line (90.0, 0.0, 0.0, 84.0, -2.0, 0.0, NULL);
ck_note (21.0, -20.0, 0.0, "Pressure Angle vs. Theta", NULL, NULL, NULL);

/* create tick marks on the graphs */
for (d = 1; d <= 3; d++)
    { dd = d;
      tick = 90.0 * dd;
      ck_line (-98.0 + x_factor * tick, 0.0, -3.0, -98.0 + x_factor * tick,
               3.0, 0.0, NULL); }

/* autoscale and redraw the graphs */
ck_auto (CK_PRIME_vp);
ck_redraw (CK_PRIME_vp);

/* set the line width to wide (width = 3) to draw the curves */
ck_set (CK_SET_LINEWIDTH, 3);

/* draw the data in the proper color */
ck_set (CK_SET_COLOR, 4);

/* draw the pressure angle curve */
while (pressure->next != NULL)
{
    /* start cf while loop */

    /* calculate the point coordinates */
    pt1x = -98.0 + x_factor * curve->index;
    pt1y = y_factor * pressure->p_angle;

    if (pressure->next->next == NULL)
        {
            pt2x = -98.0 + x_factor * curve->next_one->index;
            pt2y = y_factor * pressure->next->p_angle;
        }
    else
        {
            pt2x = -98.0 + x_factor * curve->next_one->next_one->index;
            pt2y = y_factor * pressure->next->next->p_angle;
        }

    /* draw the line segment */
    ck_line (pt1x, pt1y, 0.0, pt2x, pt2y, 0.0, NULL);

    /* advance the pointer */
    if (pressure->next->next != NULL)
        {
            pressure = pressure->next->next;
            curve = curve->next_one->next_one;
        }
    else
        {
            pressure = pressure->next;
            curve = curve->next_one;
        }
    } /* end of while loop */

/* rewind the structures to the beginning */
rewind_curve ($curve);
rewind_pressure ($pressure);

/* reset the line width to normal (width = 1) */
ck_set (CK_SET_LINEWIDTH, 1);

/* prompt user to hit ENTER to go back to main menu */
ck_pause ("Hit ENTER to continue...");

/* autoscale and redraw the part before erasing it (to be sure to */
/** get all the graphs in the viewport */
c_k_auto (CK_PRIME_VP);
c_k_redraw (CK_PRIME_VP);

/* erase screen (before program continues) */
erase_screen (CK_PRIME_VP);

/* go to the function to actually draw a rendering of the cam on */
/* the CADKEY screen */
cam_render ($curve);

/* rewind the svaj/curve data structure to the beginning */
rewind_curve ($curve);
} /* end of kinematic */

/*******************************************************************************/
void view_graphs (void)
/*******************************************************************************/
/* Purpose: Produces a graph of the current SVAJ data in memory. */
/* */
/* Author: Steve Payne  Date of creation: 08-24-93 */
/* Last modified by: Steve Payne  Date of last modification: 03-03-94 */
/*******************************************************************************/
{
  double max_s, min_s, max_v, min_v, max_a, min_a, max_j, min_j;
  double s_factor, v_factor, a_factor, j_factor, diff_s, diff_v;
  double diff_a, diff_j, x_factor;
  double ptlx, ptly, pt2x, pt2y, dd, tick;
  int current_color, d;

  /* if there is no structure then return */
  if (curve->next_one == NULL)
    {
      user_warning();
      return;
    }

  /* rewind the svaj/curve data structure to the beginning */
  rewind_curve (&curve);

  /* set up min and max values for svaj data */
  max_s = mis_s = curve->s;
  max_v = min_v = curve->v;
  max_a = min_a = curve->a;
  max_j = min_j = curve->j;

  while (curve->next_one != NULL)
    {
      /* start of while loop */
/* test for max and min values (used for scaling graph) */
if (curve->s < min_s)
{
    min_s = curve->s;
}
if (curve->s > max_s)
{
    max_s = curve->s;
}
if (curve->v < min_v)
{
    min_v = curve->v;
}
if (curve->v > max_v)
{
    max_v = curve->v;
}
if (curve->a < min_a)
{
    min_a = curve->a;
}
if (curve->a > max_a)
{
    max_a = curve->a;
}
if (curve->j < min_j)
{
    min_j = curve->j;
}
if (curve->j > max_j)
{
    max_j = curve->j;
}

/* advance the pointer */
curve = curve->next_one;
} /* end of while loop */

/* rewind the svaj/curve data structure to the beginning */
rewind_curve (&curve);

/* set the value for the difference scales */
diff_s = fabs (max_s - min_s);
diff_v = fabs (max_v - min_v);
diff_a = fabs (max_a - min_a);
diff_j = fabs (max_j - min_j);

/* set a minimum scale value (for a constant line for instance) */
if (diff_s < 0.1)
{

Appendix A: Source Code Listing - MISC.C 162
diff_s = 25.0;
}
if (diff_v < 0.1)
{
    diff_v = 25.0;
}
if (diff_a < 0.1)
{
    diff_a = 25.0;
}
if (diff_j < 0.1)
{
    diff_j = 25.0;
}

/* set up scale factors for graphing the SVAJ data */
s_factor = (75.0) / diff_s / 2.43;
v_factor = (75.0) / diff_v / 1.10;
a_factor = (75.0) / diff_a / 1.25;
j_factor = (75.0) / diff_j / 1.22;
x_factor = (83.0) / (360.0);

/* set the current color */
current_color = curve->color;

/* autoscale and redraw the cam before erasing it (to be sure to get */
/* the whole cam view in the viewport) */
ck_set (CK_SET_SCALE, CK_PRIME_VP, scale_cam, 0.0, 0.0);
ck_redraw (CK_PRIME_VP);

/* erase the screen before starting */
erase_screen (CK_PRIME_VP);

/* set the line width to ncrmal (width = 1) (if not already set) */
ck_set (CK_SET_LINEWIDTH, 1);

/* draw all 4 of the x and y axes, in color 1 */
ck_set (CK_SET_COLOR, 1);

/* set the new note height */
ck_set (CK_SET_NOTEHT, 9.0);

/* draw axis # 1 (s curve), the arrowheads and the label */
ck_line (-98.0, 8.0, 0.0, -98.0, 92.0, 0.0, NULL);
ck_line (-98.0, 50.0, 0.0, -3.0, 50.0, 0.0, NULL);
ck_line (-98.0, 8.0, 0.0, -99.5, 16.0, 0.0, NULL);
ck_line (-98.0, 8.0, 0.0, -96.5, 16.0, 0.0, NULL);
ck_line (-98.0, 92.0, 0.0, -99.5, 84.0, 0.0, NULL);
ck_line (-98.0, 92.0, 0.0, -96.5, 84.0, 0.0, NULL);
ck_line (-3.0, 50.0, 0.0, -9.0, 52.0, 0.0, NULL);
ck_line (-3.0, 50.0, 0.0, -9.0, 48.0, 0.0, NULL);
ck_note (-105.0, 54.0, 0.0, "S", NULL, NULL, NULL);

/* draw axis #2 (v curve), the arrowheads and the label */
ck_line (-98.0, -8.0, 0.0, -98.0, -92.0, 0.0, NULL);
ck_line (-98.0, -50.0, 0.0, -3.0, -50.0, 0.0, NULL);
ck_line (-98.0, -8.0, 0.0, -99.5, -16.0, 0.0, NULL);
ck_line (-98.0, -6.0, 0.0, -96.5, -16.0, 0.0, NULL);
ck_line (-98.0, -92.0, 0.0, -99.5, -84.0, 0.0, NULL);
ck_line (-98.0, -92.0, 0.0, -56.5, -84.0, 0.0, NULL);
ck_line (-3.0, -50.0, 0.0, -9.0, -52.0, 0.0, NULL);
ck_line (-3.0, -50.0, 0.0, -9.0, -48.0, 0.0, NULL);
ck_note (-105.0, -46.0, 0.0, "V", NULL, NULL, NULL);

/* draw axis #3 (v curve), and the arrowheads and the label */
ck_line (15.0, 8.0, 0.0, 15.0, 92.0, 0.0, NULL);
ck_line (15.0, 50.0, 0.0, 110.0, 50.0, 0.0, NULL);
ck_line (15.0, 8.0, 0.0, 13.5, 16.0, 0.0, NULL);
ck_line (15.0, 8.0, 0.0, 16.5, 15.0, 0.0, NULL);
ck_line (15.0, 92.0, 0.0, 13.5, 84.0, 0.0, NULL);
ck_line (15.0, 92.0, 0.0, 16.5, 84.0, 0.0, NULL);
ck_line (110.0, 50.0, 0.0, 104.0, 52.0, 0.0, NULL);
ck_line (110.0, 50.0, 0.0, 104.0, 48.0, 0.0, NULL);
ck_note (8.0, 54.0, 0.0, "A", NULL, NULL, NULL);

/* draw axis #4 (v curve), and the arrowheads and the label */
ck_line (15.0, -8.0, 0.0, 15.0, -92.0, 0.0, NULL);
ck_line (15.0, -50.0, 0.0, 110.0, -50.0, 0.0, NULL);
ck_line (15.0, -8.0, 0.0, 13.5, -16.0, 0.0, NULL);
ck_line (15.0, -8.0, 0.0, 16.5, -16.0, 0.0, NULL);
ck_line (15.0, -92.0, 0.0, 13.5, -84.0, 0.0, NULL);
ck_line (15.0, -92.0, 0.0, 16.5, -84.0, 0.0, NULL);
ck_line (110.0, -50.0, 0.0, 104.0, -52.0, 0.0, NULL);
ck_line (110.0, -50.0, 0.0, 104.0, -48.0, 0.0, NULL);
ck_note (8.0, -46.0, 0.0, "J", NULL, NULL, NULL);

/* create tick marks on the graphs */
for (d = 1; d <= 4; d++)
{
    dd = d;
    tick = 90.0 * dd;
    ck_line (-98.0 + x_factor * tick, 51.0, 0.0, -98.0 + x_factor * tick, 49.0, 0.0, NULL);
    ck_line (-98.0 + x_factor * tick, -49.0, 0.0, -98.0 + x_factor * tick, -51.0, 0.0, NULL);
    ck_line (15.0 + x_factor * tick, 51.0, 0.0, 15.0 + x_factor * tick, 49.0, 0.0, NULL);
    ck_line (15.0 + x_factor * tick, -49.0, 0.0, 15.0 + x_factor * tick, -51.0, 0.0, NULL);
}

/* set the new note height */
ok_set (CK_SET_NOTEHT, 8.0);

/* autoscale and redraw the graphs */
ck_auto (CK_PRIME_VF);
ck_redraw (CK_PRIME_VF);

/* set the line width to wide (width = 3) to draw the curves */
ck_set (CK_SET_LINEWIDTH, 3);

/* draw the s curve */
while (curve->next_one != NULL)
{
  /* start of while loop */

  /* draw the data in the proper color */
  if (curve->color != current_color)
  {
    current_color = curve->color;
    ck_set (CK_SET_COLOR, current_color);
  }

  /* calculate the point coordinates */
  pt1x = -98.0 + x_factor * curve->index;
  pt1y = 50.0 + s_factor * curve->s;
  if (curve->next_one->next_one == NULL)
  {
    pt2x = -98.0 + x_factor * curve->next_one->index;
    pt2y = 50.0 + s_factor * curve->next_one->s;
  } 
  else
  {
    pt2x = -98.0 + x_factor * curve->next_one->next_one->index;
    pt2y = 50.0 + s_factor * curve->next_one->next_one->s;
  }

  /* draw the line segment */
  ck_line (pt1x, pt1y, 0.0, pt2x, pt2y, 0.0, NULL);

  /* advance the pointer */
  if (curve->next_one->next_one != NULL)
  {
    curve = curve->next_one->next_one;
  }
  else
  {
    curve = curve->next_one;
  }
  /* end of while loop */

  /* draw the v curve */
  while (curve->prev_one != NULL)
  {
    /* start of while loop */
/* draw the data in the proper color */
if (curve->color != current_color)
{
    current_color = curve->color;
    ck_set (CK_SET_COLOR, current_color);
}

/* calculate the point coordinates */
ptlx = -98.0 + x_factor * curve->index;
ptly = -50.0 + y_factor * curve->y;
if (curve->prev_one->prev_one == NULL)
{
    pt2x = -98.0 + x_factor * curve->prev_one->index;
    pt2y = -50.0 + y_factor * curve->prev_one->y;
}
else
{
    pt2x = -98.0 + x_factor * curve->prev_one->prev_one->index;
    pt2y = -50.0 + y_factor * curve->prev_one->prev_one->y;
}

/* draw the line segment */
ck_line (ptlx, ptly, 0.0, pt2x, pt2y, 0.0, NULL);

/* advance the pointer */
if (curve->prev_one->prev_one != NULL)
{
    curve = curve->prev_one->prev_one;
}
else
{
    curve = curve->prev_one;
}
    /* end of while loop */

/* draw the a curve */
while (curve->next_one != NULL)
{
    /* start of while loop */

    /* draw the data in the proper color */
    if (curve->color != current_color)
{
        current_color = curve->color;
        ck_set (CK_SET_COLOR, current_color);
    }

    /* calculate the point coordinates */
    ptlx = 15.0 + x_factor * curve->index;
    ptly = 50.0 + a_factor * curve->a;
    if (curve->next_one->next_one == NULL)
{  
    pt2x = 15.0 + x_factor * curve->next_one->index;
    pt2y = 50.0 + a_factor * curve->next_one->a;
}  
else  
{  
    pt2x = 15.0 + x_factor * curve->next_one->next_one->index;
    pt2y = 50.0 + a_factor * curve->next_one->next_one->a;
}  
/* draw the line segment */  
ck_line (pt1x, ptly, 0.0, pt2x, pt2y, 0.0, NULL);

/* advance the pointer */  
if (curve->next_one->next_one != NULL)  
{  
    curve = curve->next_one->next_one;
}  
else  
{  
    curve = curve->next_one;
}  
}  /* end of while loop */  

/* draw the j curve */  
while (curve->prev_one != NULL)  
{  
    /* start of while loop */

    /* draw the data in the proper color */  
    if (curve->color != current_color)  
    {  
        current_color = curve->color;
        ck_set (CK_SET_COLOR, current_color);
    }

    /* calculate the point coordinates */  
    pt1x = 15.0 + x_factor * curve->index;
    ptly = -50.0 + j_factor * curve->j;
    if (curve->prev_one->prev_one == NULL)  
    {  
        pt2x = 15.0 + x_factor * curve->prev_one->index;
        pt2y = -50.0 + j_factor * curve->prev_one->j;
    }  
else  
{  
    pt2x = 15.0 + x_factor * curve->prev_one->prev_one->index;
    pt2y = -50.0 + j_factor * curve->prev_one->prev_one->j;
}  
/* draw the line segment */  
ck_line (pt1x, ptly, 0.0, pt2x, pt2y, 0.0, NULL);
/ * advance the pointer */
    if (curve->prev_one->prev_one != NULL)
    {
        curve = curve->prev_one->prev_one;
    }
    else
    {
        curve = curve->prev_one;
    }
} /* end of while loop */

/* reset the line width to normal (width = 1) */
ck_set (CK_SET_LINEWIDTH, 1);

/* prompt user to hit ENTER to go back to main menu */
ck_pause ("Hit ENTER to continue...");

/* autoscale and redraw the part before erasing it (to be sure to */
/* get all the graphs in the viewport) */
ck_autos (CK_PRIME_VP);
ck_redraw (CK_PRIME_VP);

/* erase screen (before program continues) */
erase_screen (CK_PRIME_VP);

/* go to the function to actually draw a rendering of the cam on */
/* the CADEX screen */
cam_render (&curve);

/* rewind the svaj/curve data structure to the beginning */
rewind_curve (&curve);

/* reset the color to normal (1) */
ck_set (CK_SET_COLOR, 1);
} /* end of view_graphs */
/*********************************************************************************/ /* Cam Design System - Output Functions */ /* *********************************************************************************/ /* File: OUTPUT.C */ /* *********************************************************************************/ /* Programmed by: */ /* */ /* * Virginia Tech Robotics & Mechanisms Group */ /* * c/o Dr. Charles F. Reinholtz */ /* * 106 Randolph Hall */ /* * Department of Mechanical Engineering */ /* * Virginia Tech */ /* * Blacksburg, VA 24061-0238 */ /* *********************************************************************************/ /* Purpose: All functions that perform some kind of output, e.g., saving */ /* the drawing, outputting the cam data, producing a report or a */ /* detailed engineering drawing, exporting manufacturing (CL) */ /* data, starting a new cam design, or outputting the SVAJ data. */ /* */ /* Author: Steve Payne Date of creation: 07-06-93 */ /* Last modified by: Steve Payne Date of last modification: 03-01-94 */ /* *********************************************************************************/ /* necessary include files */ #include <stdio.h> #include <stdlib.h> #include <string.h> #include "ck_cdl.h" /* for standard CADL functions */ #include "ck_dig.h" /* for dialog boxes */ #include "ck_sys.h" /* for system commands */ #include "global.h" /* Cam Design System global variable header file */ #include "cam.h" /* Cam Design System main header file */ /* *********************************************************************************/ void start_new_cam (void) /* *********************************************************************************/ /* Purpose: Clears all current data in memory, and enables the user to */ /* start the design of a new cam. */ /* *********************************************************************************/
/*
 * Author: Steve Payne  Date of creation: 12-25-93 */
/* Last modified by: Steve Payne  Date of last modification: 03-01-94 */
/***************************************************************************/

/* start of start_new_cam */

DIALOG_BOX *db = NULL;
int  choice;

/* initialize dialog box choice flags */
choice = 0;

/* free memory taken by structures */
if (prof != NULL)
  {
    free (prof);
  }
if (cam != NULL)
  {
    free (cam);
  }
if (curve != NULL)
  {
    free (curve);
  }
if (analysis != NULL)
  {
    free (analysis);
  }
if (pressure != NULL)
  {
    free (pressure);
  }

/* set up for initial dynamic memory allocation of the structures */
prof = (struct profile *) malloc (sizeof(struct profile));
cam = (struct the_cam *) malloc (sizeof(struct the_cam));
curve = (struct svaj *) malloc (sizeof(struct svaj));
analysis = (struct kine_analysis *) malloc (sizeof(struct kine_analysis));
pressure = (struct pressure_analysis *) malloc (sizeof(struct pressure_analysis));

/* first profile data in list */
prof->prev = prof->next = NULL;

/* first curve data in list */
curve->prev_one = curve->next_one = NULL;

/* first pressure angle data in list */
pressure->prev = pressure->next = NULL;
/* set cam_defined flag to 0 (no cam fully defined yet) */
cam_defined = 0;

/* set the nc_flag to 0 (to NC data to be exported) */
nc_flag = 0;

/* set the edit_flag to 0 */
sedit_flag = 0;

/* set the max lift value to 0 */
cam->max_lift = 0.0;

/* set the default cam_hub_type to 1 */
cam->hub_type = 1;

/* set default motion profile as nothing */
strcpy (motion_profile, "");

/* set the default follower type to Translating Flat-Faced */
cam->type_num = 1;
strcpy (cam->follower_type, "Translating Flat-Faced");

/* set the default angular rotation of the cam to CCW (-) */
cam->omega = -1.0;

/* set the default base circle radius to 0.0 */
analysis->k1 = 0.0;

/* set the default maximum pressure angle to 0.0 */
analysis->k6 = 0.0;

/* set all other analysis structure values to 0.0 */
analysis->k2 = analysis->k3 = analysis->k4 = 0.0;
analysis->k5 = analysis->k7 = 0.0;

/* erase the screen before the start */
erase_screen (CK_PRIME_vp);

/* everything drawn on level #5 in CADKEY (just for convenience, and we */
/* don't draw over top of something already on level #1, for instance) */
ck_set (CK_SET_LEVEL, 5);

/* set the default CADKEY color to 15 (white) */
ck_set (CK_SET_COLOR, 15);

/* set the line type to solid lines (type 1) */
ck_set (CK_SET_LINETYPE, 1);

/* set the line width to normal (width = 1) */
ck_set (CK_SET_LINEWIDTH, 1);
/* set all CADKEY units to mm */
ck_set (CK_SET_UNIT, 1);

/* set the initial location of the CADKEY graphics cursor */
ck_setcur (0.0, 0.0, CK_PRIME VP);

/* set the current window viewport dimension (in mm) */
ck_set (CK_SET_WINDOW, CK_PRIME VP, -100.0, -100.0, 100.0, 100.0);

/* set up the viewport scale and origin */
ck_set (CK_SET_SCALE, CK_PRIME VP, 1.0, 0.0, 0.0);

/* if from the edit function */
if (edit_flag == 1)
{
    /* make new dialog box to indicate file has been saved */
db = dg_init_dialog (6, 22, 2, 'Status', 0, 0, 4, 0);
dg_add_note (db, 0, 1.50, 4.0, "Memory Cleared", 0, 0, 0);
dg_add_button (db, 1, 4.00, 6.85, 8, "OK", 0, 0, DG_RET_ON_SEL);

    /* draw the dialog box */
dg_draw_dialog (db);

    do {     /* start of do-while loop */
    /* run the dialog box */
        choice = dg_run_dialog (db);
    } /* end of do-while loop */
    while (choice != 1);

    /* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);
}

    /* end of start_new_cam */


******************************************************************************
void eng_drawing (void)
******************************************************************************
/* Purpose: Produces a full detailed engineering drawing of the cam. */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
******************************************************************************
{
    /* start of eng_drawing */
    DIALOG_BOX *db = NULL;
    int choice, flag1, flag2, flag3, flag4, cancel_flag;
    double scale_val, x_center, y_center;
    char title[27], author[27], date[20], scale[12];
/ * initialize dialog box choice flag and other variables * /
  choice = scale_val = x_center = y_center = 0;
  flag1 = flag2 = flag3 = flag4 = cancel_flag = 0;

  /* if there is no structure then return */
  if (curve->next_one == NULL)
    {
      user_warning();
      return;
    }

  /* autoscale the cam profile */
  ck_auto (CK_PRIME_VP);

  /* automatically get the scale and center of the new viewport */
  ck_inquire (CK_SET_SCALE, CK_PRIME_VP, &scale_val, &x_center, &y_center);

  /* set default values of string to null */
  strcpy (title, "");
  strcpy (author, "");
  strcpy (date, "");
  strcpy (scale, "");

  /* initialize dialog box */
  db = dg_init_dialog (10, 47, 11, "Engineering Drawing", 0, 0, 4, 0);
  dg_add_note (db, 0, 0.75, 2.00, "Drawing Title:", 0, 0, 0);
  dg_add_note (db, 1, 2.25, 9.00, "Author:", 0, 0, 0);
  dg_add_note (db, 2, 3.75, 11.00, "Date:", 0, 0, 0);
  dg_add_note (db, 3, 5.25, 10.05, "Scale:", 0, 0, 0);
  dg_add_text_string (db, 4, 0.75, 18.25, 26, "%s", title, 0, 0, DG_RET_ON_SEL);
  dg_add_text_string (db, 5, 2.25, 18.25, 26, "%s", author, 0, 0, DG_RET_ON_SEL);
  dg_add_text_string (db, 6, 3.75, 18.25, 19, "%s", date, 0, 0, DG_RET_ON_SEL);
  dg_add_text_string (db, 7, 5.25, 18.25, 11, "%s", scale, 0, 0, DG_RET_ON_SEL);
  dg_add_box (db, 8, 7.00, 0.50, 0.05, 44.90, NULL, 0, 0, 0);
  dg_add_button (db, 9, 7.85, 5.00, 8, "OK", 0, 0, DG_RET_ON_SEL);
  dg_add_button (db, 10, 7.85, 33.00, 8, "Cancel", 0, 0, DG_RET_ON_SEL);

  /* draw the dialog box */
  dg_draw_dialog (db);

  while (choice != 10)
    {
      /* start of while loop */
      /* run the dialog box */
      choice = dg_run_dialog (db);

      switch (choice)
{    /* start of switch loop */
    case 4:
        {
            strcpy (title, dg_get_text_string (db, 4));
            flag1 = 1;
            break;
        }
    case 5:
        {
            strcpy (author, dg_get_text_string (db, 5));
            flag2 = 1;
            break;
        }
    case 6:
        {
            strcpy (date, dg_get_text_string (db, 6));
            flag3 = 1;
            break;
        }
    case 7:
        {
            strcpy (scale, dg_get_text_string (db, 7));
            flag4 = 1;
            break;
        }
    case 9:
        {
            /* start of case #9 */
            if (flag1 == 0)
            {
                sprintf (prompt,
                        "You haven't set the title...ENTER to continue...");
                ck_pause (prompt);
                sprintf (prompt,
                        "");
                ck_prompt (prompt);
                choice = 8;
                dg_move_focus (db, 8);
            }
            else
            if (flag2 == 0)
            {
                sprintf (prompt,
                        "You haven't set the author...ENTER to continue...");
                ck_pause (prompt);
                sprintf (prompt,
                        "");
                ck_prompt (prompt);
                choice = 8;
                dg_move_focus (db, 8);
            }
            else

if (flag3 == 0)
{
    printf(prompt,
            "You haven't set the date...ENTER to continue...");
    ck_pause(prompt);
    printf(prompt,
            "");
    ck_prompt(prompt);
    choice = 8;
    dg_move_focus(db, 8);
}
else
if (flag4 == 0)
{
    printf(prompt,
            "You haven't set the scale...ENTER to continue...");
    ck_pause(prompt);
    printf(prompt,
            "");
    ck_prompt(prompt);
    choice = 8;
    dg_move_focus(db, 8);
}
else
{ /* start of else loop */
    ck_set(CK_SET_COLOR, 1);

    ck_line(-170.0, 200.0, 0.0, 195.0, 200.0, 0.0, NULL);
    ck_line(195.0, 200.0, 0.0, 195.0, -150.0, 0.0, NULL);
    ck_line(195.0, -150.0, 0.0, -170.0, -150.0, 0.0, NULL);
    ck_line(-170.0, -150.0, 0.0, -170.0, 200.0, 0.0, NULL);

    /* set the default note height according to the */
    /* base circle radius */
    ck_set(CK_SET_NOTEHT, analysis->k1 / 10.0);

    /* draw in the title block */
    strcpy(promise, "Title: ");
    strcat(promise, title);
    ck_note(analysis->k1 * 1.1, -analysis->k1 * 1.20, 0.0, prompt,
            NULL, NULL, NULL);
    strcpy(promise, "Author: ");
    strcat(promise, author);
    ck_note(analysis->k1 * 1.1, -analysis->k1 * 1.35, 0.0,
            prompt, NULL, NULL, NULL);
    strcpy(promise, "Date: ");
    strcat(promise, date);
    ck_note(analysis->k1 * 1.1, -analysis->k1 * 1.50, 0.0,
            prompt, NULL, NULL, NULL);
    strcpy(promise, "Scale: ");
    strcat(promise, scale);
ck_note (analysis->k1 * 1.1, -analysis->k1 * 1.65, 0.0, 
prompt, NULL, NULL, NULL);

    /* set the flag to indicate the drawing should be saved */
    cancel_flag = 1;

    choice = 10;
    break;
}     /* end of else loop */
}       /* end of case #9 */
}       /* end of switch loop */
}       /* end of while loop */

    /* erase and free memory from dialog box */
    dg_erase_dialog (db);
    dg_free_dialog (db);

    /* autoscale and redraw the cam profile */
    if (cancel_flag == 1)
    {
        ck_auto (CK_PRIME_VP);
        ck_redraw (CK_PRIME_VP);
    }

    /* go to the drawing_save function to actually save the drawing */
    if (cancel_flag == 1)
    {
        drawing_save();
    }

    /* autoscale the cam profile */
    if (cancel_flag == 1)
    {
        ck_set (CK_SET_SCALE, CK_PRIME_VP, 0.1, 0.0, 0.0);
        erase_screen (CK_PRIME_VP);
    }

    /* delete the entities and redraw the cam and follower */
    if (cancel_flag == 1)
    {
        /* rewind the svaj/curve data structure to the beginning */
        rewind_curve (&curve);

        /* get the nc data from the cam_render function */
        cam_render (&curve);

        /* rewind the svaj/curve data structure to the beginning */
        rewind_curve (&curve);
    }

}            /* end of eng_drawing */
void report (void)

/* Purpose: Exports a full report on the cam. */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */

FILE *file;
DIALOG_BOX *db = NULL;
char drive[3], path[35], filename[9], name[50], segment_name[7];
double position, angle_start;
int pcs, choice, flag, d;
int flag1, flag2, flag3, flag4, flag5, flag6, flag7, flag8, flag9;

/* initialize dialog box choice flags, and angle_start variable */
choice = flag = 0;
angle_start = 0.0;
flag1 = flag2 = flag3 = flag4 = flag5 = flag6 = flag7 = flag8 = flag9 = 0;

/* if there is no structure then return */
if (curve->next_one == NULL)
{
    user_warning();
    return;
}

/* set default path and filenames */
strcpy (drive, input_drive);
strcpy (path, input_path);
strcpy (filename, "report");

/* initialize dialog box */
db = dg_init_dialog (8, 43, 10, "Produce Cam Report", 0, 0, 4, 0);
dg_add_note (db, 0, 0.75, 5.00, "Driver", 0, 0, 0);
dg_add_note (db, 1, 2.25, 6.00, "Path", 0, 0, 0);
dg_add_note (db, 2, 3.75, 2.00, "Filename", 0, 0, 0);
dg_add_text_string (db, 3, 0.75, 13.25, 2, "%s", drive, 0, 0,
                      DG_RET_ON_SEL);
dg_add_text_string (db, 4, 2.25, 13.25, 27, "%s", path, 0, 0,
                      DG_RET_ON_SEL);
dg_add_text_string (db, 5, 3.75, 13.25, 8, "%s", filename, 0, 0,
                      DG_RET_ON_SEL);
dg_add_note (db, 6, 3.75, 21.70, ".cam", 0, 0, 0);
dg_add_box (db, 7, 5.50, 0.50, 0.05, 41.90, NULL, 0, 0, 0);
dg_add_button (db, 8, 6.20, 5.00, 8, "OK", 0, 0, DG_RET_ON_SEL);
dg_add_button (db, 9, 6.20, 29.50, 8, "Cancel", 0, 0, DG_RET_ON_SEL);
/* draw the dialog box */
dg_draw_dialog (db);

while (choice != 9)
{
    /* start of while loop */
    /* run the dialog box */
    choice = dg_run_dialog (db);

    switch (choice)
    {
        /* start of switch loop */
        case 3:
        {
            strcpy (drive, dg_get_text_string (db, 3));
            break;
        }
        case 4:
        {
            strcpy (path, dg_get_text_string (db, 4));
            break;
        }
        case 5:
        {
            strcpy (filename, dg_get_text_string (db, 5));
            break;
        }
        case 8:
        {
            /* start of case #8 */
            /* open the file for write mode */
            strcpy (name, drive);
            strcat (name, path);
            strcat (name, filename);
            strcat (name, ".cam");

            /* open the data file for writing the cam report */
            file = fopen (name, "w");

            /* include header for cam data */
            fprintf (file,
"**************************************************************************\n");
            fprintf (file,
"*\n");
            fprintf (file,
"Cam Design System - Cam Analysis Report\n");
            fprintf (file, "\nType of Follower: \e\n\n", cam->follower_type);

            /* loop to output each segment of the motion_profile */
            for (d = 1; d <= (strlen (motion_profile)); d++)
        }
{  
  /* assign value to segment_name (full names of seg_type) */
  if (strcmp (prof->seg_type, "R") == 0)  
    {  
      strcpy (segment_name, "Rise ");  
    }  
  else  
  if (strcmp (prof->seg_type, "P") == 0)  
    {  
      strcpy (segment_name, "Fall ");  
    }  
  else  
    {  
      strcpy (segment_name, "Dwell");  
    }

  /* write cam profile data to the beginning of the report */
  fprintf (file, "#d: %s ", prof->seg_no, segment_name);

  if (angle_start < 10.0)  
    {  
      fprintf (file, " %2f\% - ", angle_start);  
    }  
  else  
  if ((angle_start >= 10.0) && (angle_start < 100.0))  
    {  
      fprintf (file, " %2f\% - ", angle_start);  
    }  
  else  
    {  
      fprintf (file, "%.2f - ", angle_start);
    }

  if ((angle_start + prof->angle_range) < 10.0)  
    {  
      fprintf (file, " %.2f\% ", angle_start +  
          prof->angle_range);
    }  
  else  
  if (((angle_start + prof->angle_range) >= 10.0) &&  
    ((angle_start + prof->angle_range) < 100.0))  
    {  
      fprintf (file, " %.2f\% ", angle_start +  
          prof->angle_range);
    }  
  else  
    {  
      fprintf (file, "%.2f ", angle_start + prof->angle_range);
    }

  if (prof->angle_range < 10.0)
{  
  fprintf (file, "  (%.2f) ", prof->angle_range);
}
else
if ((prof->angle_range >= 10.0) && (prof->angle_range < 100.0))
{
  fprintf (file, "  (%.2f) ", prof->angle_range);
}
else
{
  fprintf (file, "  (%.2f) ", prof->angle_range);
}

if (strcmp (prof->seg_type, "D") != 0)
{
  if ((prof->curve_type_no == 7) ||
       (prof->curve_type_no == 8))
  {
    fprintf (file, "%s \t", prof->curve_type);
  }
  else
  {
    fprintf (file, "%s \t\t", prof->curve_type);
  }
}
else
{
  fprintf (file, \n');
}

if (prof->stroke_length != 9999.99)
{
  fprintf (file, "h = %.2f", prof->stroke_length);

  /* sets units for stroke length: (mm) for translating */
  /* follower, (\$) for oscillating follower */
  if ((cam->type_num == 2) || (cam->type_num == 4))
  {
    fprintf (file, " \$\n");
  }
  else
  {
    fprintf (file, " mm\n");
  }
}

  /* increase the angle_start counter */
  angle_start += prof->angle_range;

  /* advance the pointer */
  if (d != (strlen (motion_profile)])
{
    prof = prof->next;
}

/* other cam parameters here */
printf (file, "\nBase Circle Radius: \%.2f mm", analysis->k1);
switch (cam->type_num)
{
    case 1:
    {
        printf (file, "\nOblique Angle: \%.2f\", analysis->k2);
        break;
    }
    case 2:
    {
        printf (file, "\nX Pivot Location: \%.2f mm",
                   analysis->k2);
        printf (file, "\nY Pivot Location: \%.2f mm",
                   analysis->k3);
        break;
    }
    case 3:
    {
        printf (file, "\nOffset Distance: \%.2f mm", analysis->k2);
        printf (file, "\nRoller Radius: \%.2f mm", analysis->k3);
        break;
    }
    case 4:
    {
        printf (file, "\nRoller Radius: \%.2f mm", analysis->k2);
        printf (file, "\nX Pivot Location: \%.2f mm",
                   analysis->k3);
        printf (file, "\nY Pivot Location: \%.2f mm",
                   analysis->k4);
        printf (file, "\nPfollower Arm Length: \%.2f mm",
                   analysis->k5);
        break;
    }

    /* print angular rotation direction */
    if (cam->omega < 0.6)
    {
        printf (file, "\n\nAngular Rotation: CCW (-)");
    }
    else
    {
        printf (file, "\n\nAngular Rotation: CW (+)");
    }
}
/* output the cam hub type */
fprintf (file, "\nCam Hub Type: %d \n", cam->hub_type);

switch (cam->hub_type)
{/n
 case 1:
  {
  fprintf (file, "\nVertical\n");
  break;
  }
 case 2:
  {
  fprintf (file, "\nHorizontal\n");
  break;
  }
}

/* begin specific analysis of report here */
fprintf (file, "\nMAX Pressure Angle: %.2fs\n\n", analysis->k6);

/* print characteristics for each curve type in the cam profile */
/* rewind the profile structure back to the beginning */
rewind_profile (prof);

for (d = 1; d <= (strlen (motion_profile)); d++)
{ /* start of for loop */
  if (d != 1)
  {
   /* advance the curve_type pointer */
   prof = prof->next;
  }

  switch (prof->curve_type_no)
  { /* start of switch loop */
 case 1:
   { /* Ramp Function */
     if (flag1 == 0)
     {
      flag1 = 1;
      fprintf (file, "Ramp Function:\n");
      fprintf (file, " This function is also called a constant ");
      fprintf (file, "velocity function. It provides a\n");
      fprintf (file, " zero acceleration and jerk value.\n");
     }
     break;
   }
 case 2:
   { /* SHM Curve */
     if (flag2 == 0)
     {
     }
flag2 = 1;
fprintf (file, "Simple Harmonic Motion Function:\n");
fprintf (file, " This function is smooth and continuous ");
fprintf (file, " in all derivatives, and is very\n");
fprintf (file, " useful in a wide array of ");
fprintf (file, "applications.\n\n");
break;
}
case 3:
    /* Cycloidal Function */
if (flag3 == 0)
    {
        flag3 = 1;
        fprintf (file, "Half Cycloidal Function:\n");
        fprintf (file, " This function is similar to the Cycloidal ");
        fprintf (file, "Displacement function, except\n");
        fprintf (file, " that the velocity function only completes ");
        fprintf (file, " a = cycle, and the acceleration\n");
        fprintf (file, " function only completes a \- cycle.\n\n");
    }
break;
}
case 4:
    /* Modified Sine Function */
if (flag4 == 0)
    {
        flag4 = 1;
        fprintf (file, "Modified Sine Function:\n");
        fprintf (file, " This function combines 2 sinusoidal curves ");
        fprintf (file, "of different frequency to get an\n");
        fprintf (file, " overall smoother curve with a lower peak ");
        fprintf (file, "value of acceleration. It also has\n");
        fprintf (file, " a lower peak velocity and a smoother jerk ");
        fprintf (file, "function.\n\n");
    }
break;
}
case 5:
    /* 3-4-5 Polynomial Function */
if (flag5 == 0)
    {
        flag5 = 1;
        fprintf (file, "3-4-5 Polynomial Function:\n");
        fprintf (file, " This function has a continuous profile ");
        fprintf (file, " throughout all derivatives. It also\n");
        fprintf (file, " has relatively low values of peak ");
        fprintf (file, "acceleration and velocity.\n\n");
    }
break;
}
case 6:
{
    /* Trapezoidal Acceleration Function */
    if (flag6 == 0)
    {
        flag6 = 1;
        fprintf (file, "Trapezoidal Acceleration Function:\n");
        fprintf (file, " This function is essentially a square ");
        fprintf (file, " wave with the corners 'knocked off'.\n");
        fprintf (file, " The jerk is very discontinuous, and tends to ");
        fprintf (file, "bring about vibrations in the\n");
        fprintf (file, " follower train. It is good for high-speed ");
        fprintf (file, " applications, but might prove to\n");
        fprintf (file, " be difficult to machine.\n");
    }
    break;
}

case 7:
{
    /* Cycloidal Displacement Function */
    if (flag7 == 0)
    {
        flag7 = 1;
        fprintf (file, "Cycloidal Displacement Function:\n");
        fprintf (file, " This function is formed from the path made ");
        fprintf (file, " by a point on the circumference\n");
        fprintf (file, " of a circle when rolled along a straight line. ");
        fprintf (file, "It is made from sine and\n");
        fprintf (file, " cosine functions, and is excellent for use in ");
        fprintf (file, "high-speed applications.\n");
    }
    break;
}

case 8:
{
    /* Modified Trapezoidal Function */
    if (flag8 == 0)
    {
        flag8 = 1;
        fprintf (file, "Modified Trapezoidal Function:\n");
        fprintf (file, " This function helps to minimize extreme ");
        fprintf (file, " acceleration values. It combines\n");
        fprintf (file, " pieces of a sinusoidal acceleration curve with ");
        fprintf (file, "trapezoidal acceleration\n");
        fprintf (file, " curve in order to get a smoother curve at ");
        fprintf (file, " the corners of each interval.\n");
        fprintf (file, " It also provides for smoother transitions at ");
        fprintf (file, "beginning and end of the\n");
        fprintf (file, " intervals throughout all derivatives. ");
    }
fprintf (file, "It has excellent vibration\n");
fprintf (file, "characteristics, and is easy to
fabricate.\n\n");
}
break;
}
case 9:
{  /* 4-5-6-7 Polynomial Function */
if (flag9 == 0):
{  
flag9 = 1;
fprintf (file, "4-5-6-7 Polynomial Function:\n");
fprintf (file, "This function is similar to the 3-4-5 ");
fprintf (file, "polynomial function, and constrains\n");
fprintf (file, "the jerk function to be zero at both ends of
');

fprintf (file, "the interval.\n\n")
}
break;
}
/* end of switch loop */
}  /* end of for loop */

/* rewind the profile structure back to the beginning */
rewind_profile (&prof);

/* close the file for write */
fclose (file);

/* rewind the structure */
rewind_profile (&prof);

/* fool the loop into exiting (done saving...) */
choice = 9;
flag = 1;
break;
}  /* end of case #8 */
}  /* end of switch loop */
}  /* end of while loop */

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

/* display dialog box if data has been saved */
if (flag == 1)
{  /* start of OK button if loop */
/* get length of the filename */
    position = strien (name);
/* enable type conversion from double to int */
pos = position;

/* make new dialog box to indicate file has been saved */
db = dg_init_dialog (6, (pos + 5), 2, "File Saved", 0, 0, 4, 0);
dg_add_note (db, 0, 1.50, 2.5, name, 0, 0, 0);
dg_add_button (db, 1, 4.00, ((position + 5.0) / 2.0 - 4.0), 8, "OK", 0, 0, DG_RET_ON_SEL);

/* draw the dialog box */
dg_draw_dialog (db);

do {  /* start of do-while loop */
   /* run the dialog box */
   choice = dg_run_dialog (db);
}  /* end of do-while loop */
   while (choice != 1);

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

}  /* end of OK button if loop */

}  /* end of report */

/***************************************************************************/
void save_sva.data (void)
/***************************************************************************/
/* Purpose: Writes cam SVAJ data (position, velocity, acceleration and */
/* jerk data to a file. */
/* */
/* Author: Steve Payne Date of creation: 12-25-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
/***************************************************************************/
{  /* start of save_sva.data */

FILE *file;
DIALOG_BOX *db = NULL;
char drive[3], path[35], filename[9], name[50];
double position;
int pos, choice, flag;

/* initialize dialog box choice flags */
choice = flag = 0;

/* if there is no structure then return */
if (curve->next_one == NULL)
{
   user_warning();
   return;

Appendix A: Source Code Listing - OUTPUT.C 186
ائيةلا عالم dialog box */
db = dg_init_dialog (8, 43, 10, "Save SVAJ Data", 0, 0, 4, 0);
dg_add_note (db, 0, 0.75, 5.00, "Drive:", 0, 0, 0);
dg_add_note (db, 1, 2.25, 6.00, "Path:", 0, 0, 0);
dg_add_note (db, 2, 3.75, 2.00, "Filename:", 0, 0, 0);
dg_add_text_string (db, 3, 0.75, 13.25, 2, "%s", NULL, 0, 0,
DG_RET_ON_SEL);  
dg_add_text_string (db, 4, 2.25, 13.25, 27, "%s", NULL, 0, 0,
DG_RET_ON_SEL);  
dg_add_text_string (db, 5, 3.75, 13.25, 8, "%s", NULL, 0, 0,
DG_RET_ON_SEL);  
dg_add_note (db, 6, 3.75, 21.70, ".cam", 0, 0, 0); 
dg_add_box (db, 7, 5.50, 0.50, 0.05, 41.90, NULL, 0, 0, 0);  
dg_add_button (db, 8, 6.20, 5.00, 8, "OK", 0, 0, DG_RET_ON_SEL);  
dg_add_button (db, 9, 6.20, 29.50, 8, "Cancel", 0, 0, DG_RET_ON_SEL);

/* set default path and filenames */
strcpy (drive, input_drive); 
dg_set_text_string (db, 3, drive); 
strcpy (path, input_path); 
dg_set_text_string (db, 4, path); 
strcpy (filename, "svajdata"); 
dg_set_text_string (db, 5, filename);

/* draw the dialog box */
dg_draw_dialog (db);

while (choice != 9)  
{   /* start of while loop */

    /* run the dialog box */
    choice = dg_run_dialog (db);

    switch (choice)
    {   /* start of switch loop */
        case 3:
        {
            strcpy (drive, dg_get_text_string (db, 3));
            break;
        }
        case 4:
        {
            strcpy (path, dg_get_text_string (db, 4));
            break;
        }
        case 5:
        {
            strcpy (filename, dg_get_text_string (db, 5));
            break;
        }
    }
}
case 8:
{
    /* start of case #8 */
    /* open the file for write mode */
    strcpy (name, drive);
    strcat (name, path);
    strcat (name, filename);
    strcat (name, ".cam");

    /* open the data file for writing the SVAJ data */
    file = fopen (name, "w");

    /* write cam profile data (svaj data) to the file */
    fprintf (file, "%f,%d
", curve->index, curve->s, curve->v, curve->a, curve->j, curve->color);

    /* repeat this until end of profile list */
    while (curve->next_one != NULL)
    {
        /* advance the pointer to the next svaj data in structure */
        curve = curve->next_one;

        /* write cam profile data to the file */
        fprintf (file, "%f,%f,%f,%d
", curve->index, curve->s, curve->v, curve->a, curve->j, curve->color);
    }

    /* close the file for write */
    fclose (file);

    /* rewind the svaj/curve data structure to the beginning */
    rewind_curve (&curve);

    /* fool the loop into exiting (done saving...) */
    choice = 9;
    flag = 1;
    break;
} /* end of case #8 */
} /* end of switch loop */
} /* end of while loop */

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

/* display dialog box if data has been saved */
if (flag == 1)
{
    /* start of OK button if loop */
    /* get length of the filename */
    position = strlen (name);
/* enable type conversion from double to int */
pos = position;

/* make new dialog box to indicate file has been saved */
db = dg_init_dialog (6, (pos + 5), 2, "File Saved", 0, 0, 4, 0);
dg_add_note (db, 0, 1.50, 2.5, name, 0, 0, 0);
dg_add_button (db, 1, 4.00, ((position + 5.0) / 2.0 - 4.0), 8, "OK",
0, 0, DG_DET_ON_SEL);

/* draw the dialog box */
dg_draw_dialog (db);

do {   /* start of do-while loop */
    /* run the dialog box */
    choice = dg_runDialog (db);
}   /* end of do-while loop */
  while (choice != 1);

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

}   /* end of OK button if loop */

}   /* end of save_svaJ_data */

/***************PUBLIC**********************************************/

void save_cam_data (void)

/****************Purpose: Writes both structures to a text file.*******************/

/    Author: Steve Payne          Date of creation: 07-11-93
/    Last modified: Steve Payne   Date of last modification: 01-07-94
/    *******************************************************************/
{
    FILE  *file;
    DIALOG BOX *db = NULL;
    int   choice, flag, pos;
    double position;
    char  drive[3], path[35], filename[9], name[50];

    /* initialize dialog box choice flags */
    choice = flag = 0;

    /* if there is no structure then return! */
    if (cam_defined != 1)
    {
        /* make new dialog box to indicate that there is no cam data */
        db = dg_init_dialog (6, 23, 2, "Error", DG_WHITE, DG_RED, 4,

* Appendix A: Source Code Listing - OUTPUT.C *
DG_CANCOLOR);
dg_add_note (db, 0, 1.50, 2.5, "No cam data exists", DG_WHITE, DG_RED,
DG_CANCOLOR);
dg_add_button (db, 1, 4.00, 7.50, 8, "OK", DG_BLUE, DG_WHITE,
DG_RET_ON_SEL);

/* draw the dialog box */
dg_draw_dialog (db);

do { /* start of do-while loop */
    /* run the dialog box */
    choice = dg_run_dialog (db);
} /* end of do-while loop */
while (choice != 1);

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);
return;
}

/* initialize dialog box */
db = dg_init_dialog (8, 43, 10, "Save Cam Data", 0, 0, 4, 0);
dg_add_note (db, 0, 0.75, 5.00, "Driver:", 0, 0, 0);
dg_add_note (db, 1, 2.25, 6.00, "Path:", 0, 0, 0);
dg_add_note (db, 2, 3.75, 2.00, "Filename:", 0, 0, 0);
dg_add_text_string (db, 3, 0.75, 13.25, 2, "%s", NULL, 0, 0,
DG_RET_ON_SEL);
dg_add_text_string (db, 4, 2.25, 13.25, 27, "%s", NULL, 0, 0,
DG_RET_ON_SEL);
dg_add_text_string (db, 5, 3.75, 13.25, 8, "%s", NULL, 0, 0,
DG_RET_ON_SEL);
dg_add_button (db, 6, 3.75, 21.70, ".cam", 0, 0, 0);
dg_add_box (db, 7, 5.50, 0.50, 0.05, 41.90, NULL, 0, 0, 0);
dg_add_button (db, 8, 6.20, 5.00, 8, "OK", 0, 0, DG_RET_ON_SEL);
dg_add_button (db, 9, 6.20, 29.50, 8, "Cancel", 0, 0, DG_RET_ON_SEL);

/* set default path and filenames */
strcpy (drive, input_drive);
dg_set_text_string (db, 3, drive);
strcpy (path, input_path);
dg_set_text_string (db, 4, path);
strcpy (filename, "camedata");
dg_set_text_string (db, 5, filename);

/* draw the dialog box */
dg_draw_dialog (db);

while (choice != 9)
{
    /* start of while loop */
    /* run the dialog box */
choice = dg_run_dialog (db);

switch (choice)
{
    /* start of switch loop */
    case 3:
    {
        strcpy (drive, dg_get_text_string (db, 3));
        break;
    }
    case 4:
    {
        strcpy (path, dg_get_text_string (db, 4));
        break;
    }
    case 5:
    {
        strcpy (filename, dg_get_text_string (db, 5));
        break;
    }
    case 8:
    {
        /* start of case #8 */
        /* open the file for write mode */
        strcpy (name, drive);
        strcat (name, path);
        strcat (name, filename);
        strcat (name, ".cam");

        file = fopen (name, "w");

        /* write cam parameter data to the file */
        fprintf (file, "%d,%s,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,...
/* close the file for write */
fclose (file);

/* fool the loop into exiting (done saving...) */
choice = 9;
flag = 1;
break;
} /* end of case #8 */
} /* end of switch loop */
} /* end of while loop */

/* rewind the structure */
rewind_profile (&prof);

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

if (flag <= 1)
{
    /* start of OK button if loop */
    /* get length of the filename */
    position = strlen (name);

    /* enable type conversion from double to int */
    pos = position;

    /* make new dialog box to indicate file has been saved */
    db = dg_init_dialog (6, (pos + 5), 2, "File Saved", 0, 0, 4, 0);
    dg_add_note (db, 0, 1.50, 2.5, name, 0, 0, 0);
    dg_add_button (db, 1, 4.00, ((position + 5.0) / 2.0 - 4.25), 8, "OK",
                  0, 0, DG_RET_ON_SEL);

    /* draw the dialog box */
    dg_draw_dialog (db);

    do {
        /* start of do-while loop */
        /* run the dialog box */
        choice = dg_run_dialog (db);
    } /* end of do-while loop */
    while (choice != 1);

    /* erase and free memory from dialog box */
    dg_erase_dialog (db);
    dg_free_dialog (db);
}
/* end of OK button if loop */
} /* end of save_cam_data */

/***************************************************************************/

Appendix A: Source Code Listing - OUTPUT.C
void drawing_save (void)
{ /* Purpose: Saves drawing as a part or pattern file. */
    /* Author: Steve Payne  Date of creation: 08-24-93 */
    /* Last modified by: Steve Payne  Date of last modification: 02-24-94 */
    /**************************************************************************/
    /* start of drawing_save */
    DIAlOG_BOX *db = NULL;
    char filename[12], name[45], temp[45], string[12];
    double position, num_double;
    int choice, cancel, flag1, flag2, num, pos;
    /* set the flags to indicate whether a file has been saved to 0 */
    flag1 = 0;
    flag2 = 0;
    num = 0;
    /* initialize dialog box choice flag */
    choice = 0;
    /* if there is no structure then return! */
    if (curve->next_one == NULL)
    { user_warning();
      return;
    }
    /* initialize string for which kind of file to save drawing as */
    strcpy (string, "p.rt");
    /* initialize dialog box */
    db = dg_init_dialog (8, 57, 13, "Save Drawing", 0, 0, 4, 0);
    dg_add_note (db, 0, 0.75, 5.00, "Drive:", 0, 0, 0);
    dg_add_note (db, 1, 2.25, 6.00, "Path:", 0, 0, 0);
    dg_add_note (db, 2, 3.75, 2.00, "Filename:", 0, 0, 0);
    dg_add_text_string (db, 3, 0.75, 13.25, 2, "%s", NULL, 0, 0,
                       DG_RET_ON_SEL);
    dg_add_text_string (db, 4, 2.25, 13.25, 27, "%s", NULL, 0, 0,
                       DG_RET_ON_SEL);
    dg_add_text_string (db, 5, 3.75, 13.25, 8, "%s", NULL, 0, 0,
                       DG_RET_ON_SEL);
    dg_add_note (db, 6, 3.75, 21.70, string, 0, 0, 0);
    dg_add_check (db, 7, 1.45, 44.25, 1, ".p rt File", 0, 0, DG_RET_ON_SEL);
    dg_add_check (db, 8, 3.10, 44.25, 0, ".ptn File", 0, 0, DG_RET_ON_SEL);
    dg_add_box (db, 9, 0.05, 42.50, 5.45, 0.24, NULL, 0, 0, 0);
    dg_add_box (db, 10, 5.50, 0.50, 0.05, 55.90, NULL, 0, 0, 0);
    dg_add_button (db, 11, 6.20, 5.00, 8, "Ok", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 12, 6.20, 43.50, 8, "Cancel", 0, 0, DG_RET_ON_SEL);
/* set variable for Cancel button */
cancel = 12;

/* set default path and filenames */
dg_set_text_string (db, 3, input_drive);
dg_set_text_string (db, 4, input_path);
strcpy (filename, "example");
dg_set_text_string (db, 5, filename);

/* draw the dialog box */
dg_draw_dialog (db);

while (choice != cancel)
{
    /* start of while loop */
    /* run the dialog box */
    choice = dg_run_dialog (db);

    switch (choice)
    {
        /* start of switch loop */
        case 3:
        {
            strcpy (input_drive, dg_get_text_string (db, 3));
            break;
        }
        case 4:
        {
            strcpy (input_path, dg_get_text_string (db, 4));
            break;
        }
        case 5:
        {
            strcpy (filename, dg_get_text_string (db, 5));
            break;
        }
        case 7:
        {
            if ((dg_get_check (db, 7) != 0) && (dg_get_check (db, 8) != 0))
            {
                strcpy (string, ".prt,.ptn");
            }
            if ((dg_get_check (db, 7) != 0) && (dg_get_check (db, 8) == 0))
            {
                strcpy (string, ".");
            }
            if ((dg_get_check (db, 7) == 0) && (dg_get_check (db, 8) != 0))
            {
                strcpy (string, ".ptn");
            }
            if ((dg_get_check (db, 7) == 0) && (dg_get_check (db, 8) == 0))
            {
                strcpy (string, ";");
            }
        }
    }
}
    }
    dq_set_note (db, 6, string);
    break;
}

case 8:
{
    if ((dg_get_check (db, 7) != 0) && (dg_get_check (db, 8) != 0))
    {
        strcpy (string, "prt", "ptn");
    }
    if ((dg_get_check (db, 7) != 0) && (dg_get_check (db, 8) == 0))
    {
        strcpy (string, "ptn");
    }
    if ((dg_get_check (db, 7) == 0) && (dg_get_check (db, 8) != 0))
    {
        strcpy (string, "prt");
    }
    if ((dg_get_check (db, 7) == 0) && (dg_get_check (db, 8) == 0))
    {
        strcpy (string, "");
    }
    dq_set_note (db, 6, string);
    break;
}

case 11:
{
    /* start of case #11 */
    /* open the file for write mode */
    strcpy (name, input_drive);
    strcat (name, input_path);
    strcat (name, filename);

    /* save as a part file */
    if ((dg_get_check (db, 7) != 0))
    {
        strcpy (temp, name);
        strcat (temp, ".prt");
        sys_prt_save (temp, CK_FALSE, CK_TRUE);
        flag1 = 1;
    }

    /* save as a pattern file */
    if ((dg_get_check (db, 8) != 0))
    {
        strcpy (temp, name);
        strcat (temp, ".ptn");

        /* set get mode to get ALL entities displayed in the prime */
        /* (current) viewport */
        ck_getall (CK_PRIME_VP);
}
sys_pnn_save (tep, SYS_SEL_LIST, NULL, 1, NULL, 0.0, 0.0, 0.0, CK_TRUE);
flag2 = 1;
}

/* clear the text at the bottom of the screen */
sprintf (prompt,
"
ck_prompt (prompt);

/* exit this loop */
choice = cancel;
break;
} /* end of case #11 */
} /* end of switch loop */
} /* end of while loop */

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

/* indicate that file has been saved */
if ((flag1 == 1) || (flag2 == 1))
{
    /* get length of the filename */
    position = strlen (name) + 4.0;

    /* enable type conversion from double to int */
    pos = position;

    /* how many files saved */
    if (flag1 <= 1)
    {
        num++; 1;
    }
    if (flag2 == 1)
    {
        num += 1;
    }

    /* enable type conversion from double to int */
    num_double = num;

    /* make new dialog box to indicate file has been saved */
    db = dg_init_dialog ((4 + num * 2), (pos + 5), 3, "File Saved", 0, 0, 4, 0);
    if (num == 1)
    {
        dg_add_note (db, 0, 1.30, 2.5, temp, 0, 6, 0);
    }
} else

if (num == 2)
{
    strcpy (temp, name);
    strcat (temp, ".prt");
    dg_add_note (db, 0, 1.40, 2.5, temp, 0, 0, 0);
    strcpy (temp, name);
    strcat (temp, ".ptn");
    dg_add_note (db, 1, 3.15, 2.5, temp, 0, 0, 0);
}

dg_add_button (db, 2, (1.80 + num_double * 2), ((position + 5.0) / 2.0 -
        4.50), 8, "$\text{OK}$", 0, 0, DG_RET.ON_SEL);

/* draw the dialog box */
dg_draw_dialog (db);

do {    /* start of do-while loop */
    /* run the dialog box */
    choice = dg_run_dialog (db);
}   /* end of do-while loop */
while (choice != 2);

/* erase and free memory from dialog box */
dg_erase_dialog (db);

dg_free_dialog (db);

}   /* end of drawing_save */

//************************************************************************/

void manufacture (void)

/************************************************************************
/* Purpose: Exports manufacturing data (CL data) on the cam. Asks the */
/* user for cutter diameter (in mm) and saves the CL Data to an ASCII */
/* text file. */
/* Author: Steve Payne */
/* Date of creation: 07-06-93 */
/* Last modified by: Steve Payne */
/* Date of last modification: 02-24-94 */
/************************************************************************
{
    /* start of manufacture */
    DIALOG BOX *db = NULL;
    int     choice, diam_flag, pos;
    char    drive[3], path[35], filename[9], name[60];
    double  cut_diam, old_cut_diam, x_val, y_val, z_val, position;
    FILE   *file;

    /* initialize dialog box choice flag and other flags */
    choice = diam_flag = 0;

    /* initialize cutter diameter value to 0.0 */
cut_diam = old_cut_diam = 0.0;

/* initialize other variables to 0.0 */
x_val = y_val = z_val = 0.0;

/* if there is no structure then return */
if (curve->next_one == NULL)
{
    user_warning();
    return;
}

/* set default path and filenames */
strcpy (drive, input_drive);
strcpy (path, input_path);
strcpy (filename, "ncdata");

/* initialize dialog box */
db = dg_init_dialog (10, 43, 14, "Produce NC Data", 0, 0, 4, 0);
dg_add_note (db, 0, 0.75, 5.00, "Drive!", 0, 0, 0);
dg_add_note (db, 1, 2.25, 6.00, "Path!", 0, 0, 0);
dg_add_note (db, 2, 3.75, 2.00, "Filename!", 0, 0, 0);
dg_add_text_string (db, 3, 0.75, 13.25, 2, "%s", drive, 0, 0,
                   DG_RET_ON_SEL);
dg_add_text_string (db, 4, 2.25, 13.25, 27, "%s", path, 0, 0,
                   DG_RET_ON_SEL);
dg_add_text_string (db, 5, 3.75, 13.25, 8, "%s", filename, 0, 0,
                   DG_RET_ON_SEL);
dg_add_note (db, 6, 3.75, 21.70, ".cam", 0, 0, 0);
dg_add_box (db, 7, 5.50, 0.50, 0.05, 41.90, NULL, 0, 0, 0);
dg_add_note (db, 8, 6.15, 2.00, "Cutter Diameter!", 0, 0, 0);
dg_add_text_double (db, 9, 6.15, 20.00, 9, ".2f", cut_diam, 0, 0,
                    DG_RET_ON_SEL);
dg_add_note (db, 10, 6.15, 30.20, "mm!", 0, 0, 0);
dg_add_box (db, 11, 7.70, 0.50, 0.05, 41.90, NULL, 0, 0, 0);
dg_add_button (db, 12, 8.35, 5.00, 8, "OK!", 0, 0, DG_RET_ON_SEL);
dg_add_button (db, 13, 8.35, 29.50, 8, "Cancel!", 0, 0, DG_RET_ON_SEL);

/* draw the dialog box */
dg_draw_dialog (db);

while (choice != 13)
{
    /* start of while loop */
    /* run the dialog box */
    choice = dg_run_dialog (db);

    switch (choice)
    {
        /* start of switch loop */
        case 3:
        {
            strcpy (drive, dg_get_text_string (db, 3));

Appendix A: Source Code Listing - OUTPUT.C
break;
}

case 4:
{
    strcpy (path, dg_get_text_string (db, 4));
    break;
}

case 5:
{
    strcpy (filename, dg_get_text_string (db, 5));
    break;
}

case 9:
{
    old_cut_diam = cut_diam;
    cut_diam = dg_get_text_double (db, 9);
    if (cut_diam <= 0)
    {
        printf (prompt,
                "Cutter diameter must be positive...ENTER to continue...");
        ck_pause (prompt);
        printf (prompt,
                "");
        ck_prompt (prompt);
        cut_diam = old_cut_diam;
        dg_set_text_double (db, 9, cut_diam);
        dg_move_focus (db, 11);
    }
    else
    {
        diam_flag = 1;
    }
    break;
}

case 12:
{
    /* start of case #12 */
    if (diam_flag != 1)
    {
        printf (prompt,
                "You haven't set the cutter diameter...ENTER to continue...");
        ck_pause (prompt);
        printf (prompt,
                "");
        ck_prompt (prompt);
        choice = 11;
        dg_move_focus (db, choice);
    }
    else
    {
        /* start of else loop */
        /* set up for initial dynamic memory allocation of */
        /* the nc structure */
}

Appendix A: Source Code Listing - OUTPUT.C

199
nc = (struct nc_data *) malloc (sizeof(struct nc_data));

/* first NC data in list */
nc->prev = nc->next = NULL;

/* set the nc flag to 1 to produce NC data */
nc_flag = 1;

/* rewind the svaj/curve data structure to the beginning */
rewind_curve (&curve);

/* get the nc data from the cam_render function */
cam_render (&curve);

/* rewind the svaj/curve data structure to the beginning */
rewind_curve (&curve);

/* rewind the nc data structure to the beginning */
rewind_nc (&nc);

/* open the file for write mode */
strcpy (name, drive);
strcat (name, path);
strcat (name, filename);
strcat (name, ".cam");

/* open the data file for writing the NC data */
file = fopen (name, "w");

/* produce the file */
while (nc->next != NULL)
{
    x_val = nc->x + (cut_diam / 2.0) * cos (nc->angle);
    y_val = nc->y + (cut_diam / 2.0) * sin (nc->angle);
    z_val = 3.0;

    fprintf (file, "%f %f %f\n", x_val, y_val, z_val);

    /* advance the pointer */
    nc = nc->next;
}

/* close the file for write */
close (file);

/* free the nc structure from memory */
free (nc);

choice = 13;
break;
} /* end of else loop */
/* end of switch loop */
} /* end of while loop */

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

if (nc_flag == 1)
{ /* start of OK button if loop */
  /* get length of the filename */
  position = strlen (name);

  /* make sure the minimum dialog box width is 29 */
  if (position < 29)
    {
      position = 29;
    }

  /* add 5 to this value for the dialog box border width */
  position *= 5.0;

  /* enable type conversion from double to int */
  pos = position;

  /* make new dialog box to indicate file has been saved */
  db = dg_init_dialog (8, pos, 3, "NC Data Produced", 0, 0, 4, 0);
  dg_add_note (db, 0, 1.25, ((pos - 29.0) / 2.0),
               "Cutter location data saved to", 0, 0, 0);
  dg_add_note (db, 1, 3.45, ((position - strlen (name)) / 2.0), name, 0,
               0, 0);
  dg_add_button (db, 2, 6.00, ((position / 2.5) - 4.25), 8, "OK", 0, 0,
                 DG_RET_ON_SEL);

  /* draw the dialog box */
  dg_draw_dialog (db);

  do { /* start of do-while loop */
      /* run the dialog box */
      choice = dg_run_dialog (db);
    } /* end of do-while loop */
  while (choice != 2);

  /* erase and free memory from dialog box */
  dg_erase_dialog (db);
dg_free_dialog (db);

  /* reset the nc flag */
  nc_flag = 0;
} /* end of OK button if loop */

} /* end of manufacture */
/* necessary include files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ck_cdl.h"    /* for standard CADL functions */
#include "ck_dlg.h"    /* for dialog boxes */
#include "global.h"    /* Cam Design System global variable header file */
#include "cam.h"       /* Cam Design System main header file */

/***********************************************************************/
void motion_segs (void)
/***********************************************************************/
/* Purpose: Allows the user to enter the motion programs for the cam */
/* profile piece-by-piece, either a chosen rise-dwell-fall */
/* sequence, or a custom-made one. */
/* */
/* Author: Steve Payne    Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
*******************************************************************************/
{
    /* start of motion_segs */
    DILOGIN_BOX *db = NULL;
    char dummy[10], old_profile[10];
    int index, flag, old_cam_defined;

    /* initialize dialog box choice and other flags */
    index = flag = 0;

    /* set old_cam_defined to current cam_defined variable */
    old_cam_defined = cam_defined;

    /* set up motion_profile variable saver */
    strcpy (old_profile, motion_profile);

    /* initialize the dialog box parameters */
    db = dg_init_dialog (12, 49, 18, "Motion Segments", 0, 0, 4, 0);
    dg_add_button (db, 0, 1.00, 2.85, 0, "Rise", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 1, 2.85, 2.85, 0, "Fall", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 2, 4.70, 2.45, 0, "Dwell", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 3, 7.10, 2.60, 0, "Backup", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 4, 1.00, 14.00, 8, "RF", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 5, 3.70, 14.00, 8, "RFD", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 6, 6.40, 14.00, 8, "RDFD", 0, 0, DG_RET_ON_SEL);
    dg_add_note (db, 7, 1.00, 23.60, ",(Rise-Fall)", 0, 0, 0);
    dg_add_note (db, 8, 3.70, 23.60, ",(Rise-Fall-Dwell)", 0, 0, 0);
    dg_add_note (db, 9, 6.40, 23.60, ",(Rise-Dwell-Fall-Dwell)", 0, 0, 0);
    dg_add_note (db, 10, 10.00, 1.50, "Motion Profile:", 0, 0, 0);
    dg_add_note (db, 11, 10.00, 18.95, motion_profile, 0, 0, 0);
    dg_add_box (db, 12, 0.05, 12.00, 8.95, 0.24, NULL, 0, 0, 0);
    dg_add_box (db, 13, 9.00, 0.50, 0.05, 47.90, NULL, 0, 0, 0);
    dg_add_box (db, 14, 9.80, 18.35, 1.20, 7.75, NULL, 0, 0, 0);
    dg_add_box (db, 15, 9.00, 27.90, 2.70, 0.24, NULL, 0, 0, 0);
    dg_add_button (db, 16, 10.00, 29.65, 6, "OK", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 17, 10.00, 39.25, 8, "Cancel", 0, 0, DG_RET_ON_SEL);

    /* draw the dialog box */
    dg_draw_dialog (db);

    /* start of dialog box running loop */
    do
    { /* start of while (dialog box running) loop */
      /* returns the index of the option selected by the user */
      index = dg_run_dialog (db);

      switch (index)
      { /* start of index switch loop */
        case 0:
        { /* start of case #0 */
          
        } /* end of case #0 */

        } /* end of switch loop */
    } /* end of while (dialog box running) loop */

    return (index);
}

} /* end of start of dialog box running loop */

Appendix A: Source Code Listing - PROFIL.C
/* Rise button selected */
if (strlen (motion_profile) >= 6)
{
    ck_pause ("MAX number of segments = 6...Hit ENTER...");
    sprintf (prompt, "");
    ck_prompt (prompt);
    dg_move_focus (db, 7);
}
else
{
    strcat (motion_profile, "R");
    dg_set_note (db, 11, "");
    dg_set_note (db, 11, motion_profile);
}
break;
} /* end of case # 0 */
case 1:
{ /* start of case # 1 */
/* Fall button selected */
if (strlen (motion_profile) >= 6)
{
    ck_pause ("MAX number of segments = 6...Hit ENTER...");
    sprintf (prompt,");
    ck_prompt (prompt);
    dg_move_focus (db, 7);
}
else
{
    strcat (motion_profile, "F");
    dg_set_note (db, 11, "");
    dg_set_note (db, 11, motion_profile);
}
break;
} /* end of case # 1 */
case 2:
{ /* start of case # 2 */
/* Dwell button selected */
if (strlen (motion_profile) >= 6)
{
    ck_pause ("MAX number of segments = 6...Hit ENTER...");
    sprintf (prompt, "");
    ck_prompt (prompt);
    dg_move_focus (db, 7);
}
else
{
    strcat (motion_profile, "D");
    dg_set_note (db, 11, "");
    dg_set_note (db, 11, motion_profile);
}
break;
}  /* end of case #2 */

case 3:
{  /* start of case #3 */
  /* Backup button selected */
  /* if no string present, then ignore this button */
  if (strlen (motion_profile) == 0)
  {
    strcpy (dummy, motion_profile);
    strcpy (motion_profile, "" );
    strcat (motion_profile, dummy, (strlen (dummy) - 1));
    strcat (motion_profile, " ");
    dg_set_note (db, 11, " ");
    dg_set_note (db, 11, motion_profile);
  }
  break;
}  /* end of case #3 */

case 4:
{  /* start of case #4 */
  if (strlen (motion_profile) > 4)
  {
    ck_pause ("MAX number of segments = 6...Hit ENTER...");
    sprintf (prompt, " ");
    ck_prompt (prompt);
    dg_move_focus (db, 7);
  }
  else
  {
    strcat (motion_profile, "RF");
    dg_set_note (db, 11, " ");
    dg_set_note (db, 11, motion_profile);
  }
  break;
}  /* end of case #4 */

case 5:
{  /* start of case #5 */
  if (strlen (motion_profile) > 3)
  {
    ck_pause ("MAX number of segments = 6...Hit ENTER...");
    sprintf (prompt, " ");
    ck_prompt (prompt);
    dg_move_focus (db, 7);
  }
  else
  {
    strcat (motion_profile, "RFD");
    dg_set_note (db, 11, " ");
    dg_set_note (db, 11, motion_profile);
  }
  break;
}  /* end of case #5 */

case 6:
{ /* start of case #6 */
if (strlen (motion_profile) > 2)
{
    ck_pause ("MAX number of segments = 6...Hit ENTER...");
    printf (prompt, "
    ck_prompt (prompt);
    dg_move_focus (db, 7);
}
else
{
    strcat (motion_profile, "RDPD");
    dg_set_note (db, 11, "
    dg_set_note (db, 11, motion_profile);
}
break;
} /* end of case #6 */
case 16:
{ /* start of case #16 */
/* OK button selected */
/* check to make sure that there is at least 1 segment */
if (strlen (motion_profile) == 0)
{
    ck_pause ("There are no segments defined...ENTER to correct");
    printf (prompt, "
    ck_prompt (prompt);
    index = 7;
    dg_move_focus (db, index);
}
else
{
    /* erase the dialog box */
    dg_erase_dialog (db);

    /* set another flag */
    if (cam_defined == 0)
    {
        flag = 3;
    }

    /* go to enter the exact motion program in this function */
    enter_motion_program ();

    if ((cam_defined > old_cam_defined))
    {
        flag = 2;
    }
else
    {
        flag = 0;
    }
}
if ((cam_defined == old_cam_defined) ||
    ((cam_defined > old_cam_defined) && (flag == 0)))
{
    index = 7;

    if (cam_defined == old_cam_defined)
    {
        flag = 1;
    }

    /* re-draw the dialog box */
    dg_draw_dialog (db);
    dg_move_focus (db, index);
}
break;
} /* end of case #16 */

/* end of index switch loop */
*/
while ((index != 17) && (flag != 2) &&
    ((flag == 0) || (cam_defined == old_cam_defined)))
{
    /* erase dialog box and free memory taken by it */
    if (index == 17)
    {
        dg_erase_dialog (db);
        dg_free_dialog (db);
    }

    /* if not editing cam design */
    if (((edit_flag == 0) && (cam_defined == 1))
    {
        calculate_cam();
    }
} /* end of motion_segs */

/******************************************************************************/
void enter_motion_program (void)
/******************************************************************************/
/* Purpose: Allows the user to enter the motion program piece-by-piece, */
/* including curve type, angle range, and stroke length for each */
/* part of the cam profile. */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
/*************************************************************************/
{
    /* start of enter_motion_program */
    DIALOG_box *dbox = NULL;
    int    number_segments, picked, d, index, ok, cancel, other_d;
    int    case1, case2, case3, flag1, flag2, flag3, flag4, flag5;
    int    count, count2, first_one;
    double dd, num_seg, rows[10], columns[10], spacing;
    double angle_total, total_lift, angle_tot, displacement_tot;
    char   string[5], the_string[2], temp[10], dummy[10];
    char   rise[2], fall[2], dwell[2];

    /* initialize the number of segments (double value) to zero */
    num_seg = 0.0;

    /* initialize dialog box choice flag */
    index = 0;

    /* initialize the first element to set the radio box element as current */
    first_one = -1;

    /* get number of segments */
    number_segments = strlen (motion_profile);

    /* use this as a dummy variable, for type conversion to float */
    num_seg = number_segments;

    /* set initial spacing variable (for spacing the curve-type buttons */
    /* in the dialog box) */
    spacing = (6.0 - num_seg) * 0.05;

    /* define these constants */
    ok = 24 + 4 * number_segments;
    cancel = 25 + 4 * number_segments;
    flag1 = flag2 = flag3 = count = count2 = 1;
    flag4 = flag5 = number_segments;
    angle_tot = displacement_tot = 0.0;

    /* initialize rise, fall, dwell variables */
    strcpy (rise, "R");
    strcpy (fall, "F");
    strcpy (dwell, "D");

    /* sets segment_type variable (essentially parses the variable */
    /* 'motion_profile', piece by piece for this information */
    strcpy (temp, motion_profile);

    /* set first radio box element as current one */
    picked = 0;

Appendix A: Source Code Listing - PROFILC 208
/* begin initializing the dialog box */
dbox = dg_init_dialog ((number_segments + 12), 53, (cancel + 1),
                      "Motion Profile", 0, 0, 0, 0);
dg_add_note (dbox, 0, 0.30, 1.25, "Segment", 0, 0, 0);
dg_add_note (dbox, 1, 1.20, 1.75, "Number", 0, 0, 0);
dg_add_note (dbox, 2, 0.30, 10.00, "Segment", 0, 0, 0);
dg_add_note (dbox, 3, 1.20, 11.25, "Type", 0, 0, 0);
dg_add_note (dbox, 4, 0.30, 20.00, "Curve", 0, 0, 0);
dg_add_note (dbox, 5, 1.20, 20.25, "Type", 0, 0, 0);
dg_add_note (dbox, 6, 0.30, 31.00, "Angle", 0, 0, 0);
dg_add_note (dbox, 7, 1.20, 29.50, "Range (°)", 0, 0, 0);
dg_add_note (dbox, 8, 0.30, 42.25, "Stroke", 0, 0, 0);
/* sets units for stroke length: (mm) for translating follower, */
/* [a] for oscillating follower */
if ((cam->type_num == 2) || (cam->type_num == 4))
{
    dg_add_note (dbox, 9, 1.20, 40.75, "Length (°)", 0, 0, 0);
} 
else
{
    dg_add_note (dbox, 9, 1.20, 40.25, "Length (mm)", 0, 0, 0);
}
dg_add_box (dbox, 10, 2.30, 0.50, 0.05, 51.95, NULL, 0, 0, 0);
dg_add_radio (dbox, 11, 2.60, 17.75, picked, number_segments, NULL, 0, 0, 0);

/* set up the alignment of the rows and columns for the radio box */
for (d = 1; d <= number_segments; d++)
{
    rows[d - 1] = 0.05 + (d - 1) * 1.25;
    columns[d - 1] = 0.00;
}

/* align the radio box */
dg_radio-align (dbox, 11, rows, columns);

for (d = 1; d <= number_segments; d++)
{ /* start of for loop */
    /* use these as dummy variables, for type conversions to float */
    dd = d;
    strcpy (dummy, itoa (d, string, 10));

    if (strcspn (temp, "R") == 0)
    {
        strcpy (the_string, "R");
        strcpy (temp, {strpbrk (temp, "R") + 1});
        if (first_one == -1)
        {
first_one = d;
}
}
else
if (strcspn(temp, "F") == 0)
{
    strcpy(the_string, "F");
    strcpy(temp, (strpbrk(temp, "F") + 1));
    if (first_one == -1)
    {
        first_one = d;
    }
}
else
if (strcspn(temp, "D") == 0)
{
    strcpy(the_string, "D");
    strcpy(temp, (strpbrk(temp, "D") + 1));
}

case1 = 11 + number_segments + d;
case2 = 11 + 2 * number_segments + d;
case3 = 11 + 3 * number_segments + d;

/* prints motion segment # */
dg_add_note(dbox, (11 + d), (1.45 + (dd * 1.25)), 4.00, string, 0, 0, 0);

/* prints motion segment type */
dg_add_note(dbox, case1, (1.45 + (dd * 1.25)), 13.00, the_string, 0, 0, 0);

/* prints motion curve type only if the seg_type */
/* is not a dwell (don't need to specify for this) */
if (strcspn(the_string, "D") == 0)
{
    dg_set_radio_text(dbox, 11, (d - 1), "N/A");

    /* set curve type # to 0 ("N/A" for dwell) */
    prof->curve_type_no = 0;

    /* set curve type to "N/A" for dwell */
    strcpy(prof->curve_type, "N/A");
    flag5 = 1;
}
else
    /* if this isn't the edit function (then a new motion profile) */
if (cam_defined == !)
{
    switch(prof->curve_type_no)
    {

Appendix A: Source Code Listing - PROFILC
case 1:
{
    dg_set_radio_text (dbox, 11, (d - 1), "Ramp");
    break;
}
case 2:
{
    dg_set_radio_text (dbox, 11, (d - 1), "SHM");
    break;
}
case 3:
{
    dg_set_radio_text (dbox, 11, (d - 1), "CyC");
    break;
}
case 4:
{
    dg_set_radio_text (dbox, 11, (d - 1), "Mod Sine");
    break;
}
case 5:
{
    dg_set_radio_text (dbox, 11, (d - 1), "3-4-5");
    break;
}
case 6:
{
    dg_set_radio_text (dbox, 11, (d - 1), "Trap");
    break;
}
case 7:
{
    dg_set_radio_text (dbox, 11, (d - 1), "CyC Disp");
    break;
}
case 8:
{
    dg_set_radio_text (dbox, 11, (d - 1), "Mod Trap");
    break;
}
case 9:
{
    dg_set_radio_text (dbox, 11, (d - 1), "4-5-6-7");
    break;
}
}
else
{
    dg_set_radio_text (dbox, 11, (d - 1), "");
}
/* set to indicate that no curve type has been selected */
prof->curve_type_no = 9999;
}

/* prints double field for angle range */
dg_add_text_double (dbox, case2, (1.45 + (dd * 1.25)), 30.25, 7, "%f",
0.0, 0, 0, DG_RET_ON_SEL);

/* if this isn’t the edit function (then a new motion profile) */
if (cam_defined == 1)
{
    dg_set_text_double (dbox, case2, prof->angle_range);
}
else
{
    /* set to indicate that no angle range has been selected */
    prof->angle_range = 9999;
}

/* prints double field for stroke length */
/* only if the seg_type is not a dwell (undefined) */
if (strcmp (the_string, "D") == 0)
{
    dg_add_note (dbox, case3, (1.55 + (dd * 1.25)), 43.50, "N/A",
0.0, 0, 0);

    /* set the value for stroke length to a dummy data value, since */
    /* for a dwell curve-type, it’s N/A */
    prof->stroke_length = 9999.99;
}
else
{
    dg_add_text_double (dbox, case3, (1.45 + (dd * 1.25)), 42.25, 7,
"%.2f", 0.0, 0, 0, DG_RET_ON_SEL);

    /* if this isn’t the edit function (then a new motion profile) */
    if (cam_defined == 1)
    {
        if (strcmp (the_string, "D") != 0)
        {
            dg_set_text_double (dbox, case3, prof->stroke_length);
        }
    }
    else
    {
        /* set to indicate that no stroke length has been selected */
        prof->stroke_length = 9999;
    }
}

/* set values for profile structure */
prof->seg_no = d;
strcpy (prof->seg_type, the_string);

if (d != number_segments)
{
    /* start of if loop */

    /* if not the edit function */
    if (cam_defined != 1)
    {
        /* set up for dynamic memory allocation of the structure */
        prof->next = (struct profile*) malloc (sizeof(struct profile));

        /* give a reference for the previous segment # */
        prof->next->prev = prof;
    }

    /* advances pointer, so that we can add another segment */
    /* to the profile */
    prof = prof->next;
    /* end of if loop */
};
/* end of for loop */

/* if all dwells, set first_one to zero so no error */
if (first_one == -1)
{
    first_one = 1;
}

/* set this as the last profile element in the structure */
prof->next = NULL;

/* rewind the cam profile structure to the beginning */
rewind_profile (&prof);

/* add a box (just a line) to separate the buttons from the text */
/* and value entry fields */
dg_add_box (dbox, (12 + 4 * number_segments), (2.85 + (num_seg * 1.25)),
0.50, 0.05, 51.90, NULL, 0, 0, 0);
dg_add_box (dbox, (13 + 4 * number_segments), (8.30 + 4.0 * spacing +
    (num_seg * 1.25)), 27.00, 0.05, 25.40, NULL, 0, 0, 0);
dg_add_box (dbox, (14 + 4 * number_segments), (8.30 + 4.0 * spacing +
    (num_seg * 1.25)), 27.00, 1.90 + 0.45 * spacing, 0.24, NULL, 0, 0, 0);

/* add buttons to enter the motion profile curve types (10 of them) */
dg_add_button (dbox, (15 + 4 * number_segments), (3.50 + (num_seg * 1.25)),
2.00, 15, "Ramp Function", DG_RED, 0, 0);
dg_add_button (dbox, (16 + 4 * number_segments), (3.50 + (num_seg * 1.25)),
21.00, 11, "SHM Curve", 0, 0, 0);
dg_add_button (dbox, (17 + 4 * number_segments), (3.50 + (num_seg * 1.25)),
35.00, 16, "Half Cycloidal", DG_RED, 0, 0);
dg_add_button (dbox, (18 + 4 * number_segments), (5.15 + spacing + (num_seg

Appendix A: Source Code Listing · PROFIL.C
* 1.25)), 2.00, 15, "Modified Sine", 0, 0, 0);
dg_add_button (bbox, (19 + 4 * number_segments), (5.15 + spacing + (num_seg * 1.25)), 21.00, 13, "3-4-5 Curve", 0, 0, 0);
dg_add_button (bbox, (20 + 4 * number_segments), (5.15 + spacing + (num_seg * 1.25)), 37.00, 13, "Trapezoidal", DG_RED, 0, 0);
dg_add_button (bbox, (21 + 4 * number_segments), (6.80 + 2.5 * spacing + (num_seg * 1.25)), 2.00, 24, "Cycloidal Displacement", 0, 0, 0);
dg_add_button (bbox, (22 + 4 * number_segments), (6.80 + 2.5 * spacing + (num_seg * 1.25)), 28.50, 22, "Modified Trapezoidal", 0, 0, 0);
dg_add_button (bbox, (23 + 4 * number_segments), (8.45 + 4.0 * spacing + (num_seg * 1.25)), 2.00, 15, "4-5-6-7 Curve", 0, 0, 0);

    /* adds 'OK' and 'Cancel' buttons */
dg_add_button (bbox, ok, (8.85 + 4.2 * spacing + (num_seg * 1.25)), 28.75, 8, "OK", 0, 0, 0);
dg_add_button (bbox, cancel, (8.85 + 4.2 * spacing + (num_seg * 1.25)), 43.00, 8, "Cancel", 0, 0, 0);

    /* set the focus to the first curve type available to enter */
dg_set_radio (bbox, 11, first_one - 1);
dg_move_focus (bbox, 11);

    /* draw the dialog box */
dg_draw_dialog (bbox);

    /* start of dialog box running loop */
while (index != cancel)
    {
        /* start of while loop */
        /* returns the index of the option selected by the user */
        index = dg_run_dialog (bbox);

        for (d = 1; d <= number_segments; d++)
            {
                /* start of for loop */

                case1 = 11 + 2 * number_segments + d;
                case2 = 11 + 3 * number_segments + d;

                if (index == case1)
                    { /* start of case #1 if loop */
                        if (d != 1)
                            {
                                forward_it (&prof, d);
                            }

                        if (prof->angle_range == 9999)
                            {
                                flag4 = 1;
                            }
                        else
                            {
                                angle_tot -= prof->angle_range;
                            }
                    }
prof->angle_range = fabs(dg_get_text_double(dbox, case1));
dg_set_text_double(dbox, case1, prof->angle_range);
angle_tot += prof->angle_range;
if (angle_tot > 360.0)
{
    printf(prompt,
        "Error in angle ranges...please check. Hit ENTER...");
    ck_pause(prompt);
    printf(prompt, "
        ");
    ck_prompt(prompt);
    angle_tot -= prof->angle_range;
    prof->angle_range = 9999;
    flag4 += 1;
    dg_set_text_double(dbox, case1, 0.0);
}

if (flag4 == 1)
{
    /* this small routine makes it so that the user does not */
    /* have to enter the last value, to add up to 360 degrees */
    /* it will calculate it for you and enter it automatically */
    /* rewind the structure temporarily */
    rewind_profile(&prof);
    count = 1;

    while (prof->angle_range != 9999)
    {
        /* advance the pointer */
        prof = prof->next;
        count += 1;
    }

    prof->angle_range = 360.0 - angle_tot;
    dg_set_text_double(dbox, (case1-d+count),
        prof->angle_range);
    angle_tot += prof->angle_range;
    flag4 -= 1;

    /* move the structure back to where it was */
    rewind_profile(&prof);
    if (d != 1)
    {
        forward_it(&prof, d);
    }
}
/* end of case #1 if loop */
else
    if (index == case2)
    {
        /* start of case #2 if loop */
        if (d != 1)
{ 
  forward_it (&prof, d);
}

if (prof->stroke_length == 9999) 
{ 
  flag5 = 1;
} 
else 
{ 
  if (strcmp (prof->seg_type, rise) == 0) 
  { 
    displacement_tot -= prof->stroke_length;
  } 
  else 
    if (strcmp (prof->seg_type, fall) == 0) 
      { 
        displacement_tot += prof->stroke_length;
      } 
  
  prof->stroke_length = fabs (dg_get_text_double (dbox, case2));
  dg_set_text_double (dbox, case2, prof->stroke_length);
  if (strcmp (prof->seg_type, rise) == 0) 
    { 
      displacement_tot += prof->stroke_length;
    } 
  else 
    if (strcmp (prof->seg_type, fall) == 0) 
      { 
        displacement_tot -= prof->stroke_length;
      } 

  if (flag5 == 1) 
  { 
    /* this small routine makes it so that the user does not */
    /* have to enter the last value, to make the total overall */
    /* displacement equal to zero. */
    /* it will calculate it for you and enter it automatically */

    /* rewind the structure temporarily */
    rewind_profile (&prof);
    count2 = 1;

    while (prof->stroke_length != 9999) 
    { 
      /* advance the pointer */
      prof = prof->next;
      count2 += 1;
    }

    if ((strcmp (prof->seg_type, rise) == 0) &&
        (displacement_tot > 0.0))
{  
sprintf (prompt,  
    "Error in total rise...please check. Hit ENTER...");  
ck_pause (prompt);  
sprintf (prompt, "");  
ck_prompt (prompt);  
}
else
if ((strcmp (prof->seg_type, fall) == 0) &&
    (displacement_tot < 0.0))
{
    printf (prompt,  
        "Error in total rise...please check. Hit ENTER...");  
ck_pause (prompt);  
sprintf (prompt, "");  
ck_prompt (prompt);  
}
else
{
    prof->stroke_length = fabs (displacement_tot);
    dg_set_text_double (dbox, (case2-d+count2),
        prof->stroke_length);
    displacement_tot += prof->stroke_length;
    if (strcmp (prof->seg_type, rise) == 0)
    {
        displacement_tot += prof->stroke_length;
    }
else
    if (strcmp (prof->seg_type, fall) == 0)
    {
        displacement_tot -= prof->stroke_length;
    }
    flag5 -= 1;
    /* move the structure back to where it was */
    rewind_profile (&prof);
    if (d >= 1)
    {
        forward_it (&prof, d);
    }
}
} /* end of case #2 if loop */

/* set up if loops to assign curve types to structure (from */
/* the user-selectable buttons) */
if ((index >= (15 + 4 * number_segments)) &&
    (index <= (23 + 4 * number_segments)))
{
    picked = dg_get_radio (dbox, 11);
}
/* Ramp Curve button */
if (index == (15 + 4 * number_segments))
{
    if (picked != 0)
    {
        forward_it (&prof, (picked + 1));
    }
    if (strcmp (prof->seg_type, "D") != 0)
    {
        strcpy (prof->curve_type, "Ramp Function");
        prof->curve_type_no = 1;
        dg_set_radio_text (dbox, 11, picked, "      ");
        dg_set_radio_text (dbox, 11, picked, "Ramp");
    }
}

/* SSM Curve Button */
if (index == (16 + 4 * number_segments));
{
    if (picked != 0)
    {
        forward_it (&prof, (picked + 1));
    }
    if (strcmp (prof->seg_type, "D") != 0)
    {
        strcpy (prof->curve_type, "SSM Curve");
        prof->curve_type_no = 2;
        dg_set_radio_text (dbox, 11, picked, "      ");
        dg_set_radio_text (dbox, 11, picked, "SSM");
    }
}

/* Half Cycloidal Curve Button */
if (index == (17 + 4 * number_segments))
{
    if (picked != 0)
    {
        forward_it (&prof, (picked + 1));
    }
    if (strcmp (prof->seg_type, "D") != 0)
    {
        strcpy (prof->curve_type, "Half Cycloidal");
        prof->curve_type_no = 3;
        dg_set_radio_text (dbox, 11, picked, "      ");
        dg_set_radio_text (dbox, 11, picked, "* Cyc");
    }
}

/* Modified Sine Curve Button */
if (index == (18 + 4 * number_segments))
{  
if (picked != 0)  
{  
    forward_it (&prof, (picked + 1));  
}
if (strcmp (prof->seg_type, "D") != 0)  
{  
    strcpy (prof->curve_type, "Modified Sine");  
    prof->curve_type_no = 4;  
    dg_set_radio_text (dbox, 11, picked, "  ");  
    dg_set_radio_text (dbox, 11, picked, "Mod Sine");  
}
}

/* 3-4-5 Curve Button */  
if (index == (19 + 4 * number_segments))  
{  
if (picked != 0)  
{  
    forward_it (&prof, (picked + 1));  
}
if (strcmp (prof->seg_type, "D") != 0)  
{  
    strcpy (prof->curve_type, "3-4-5 Curve");  
    prof->curve_type_no = 5;  
    dg_set_radio_text (dbox, 11, picked, "  ");  
    dg_set_radio_text (dbox, 11, picked, '3-4-5');  
}
}

/* Trapezoidal Curve Button */  
if (index == (20 + 4 * number_segments))  
{  
if (picked != 0)  
{  
    forward_it (&prof, (picked + 1));  
}
if (strcmp (prof->seg_type, "D") != 0)  
{  
    strcpy (prof->curve_type, "Trapezoidal");  
    prof->curve_type_no = 6;  
    dg_set_radio_text (dbox, 11, picked, "  ");  
    dg_set_radio_text (dbox, 11, picked, "Trap");  
}
}

/* Cycloidal Displacement Curve Button */  
if (index == (21 + 4 * number_segments))  
{  
if (picked != 0)  
{  
}
forward_it (&prof, (picked + 1));
}

if (strcmp (prof->seg_type, "D") != 0)
{
  strcpy (prof->curve_type, "Cycloidal Displacement");
  prof->curve_type_no = 7;
  dg_set_radio_text (dbox, 11, picked, "");
  dg_set_radio_text (dbox, 11, picked, "Cyc Disp");
}

/* Modified Trapezoidal Curve Button */
if (index == (22 + 4 * number_segments))
{
  if (picked != 0)
  {
    forward_it (&prof, (picked + 1));
  }
  if (strcmp (prof->seg_type, "D") != 0)
  {
    strcpy (prof->curve_type, "Modified Trapezoidal");
    prof->curve_type_no = 8;
    dg_set_radio_text (dbox, 11, picked, "");
    dg_set_radio_text (dbox, 11, picked, "Mod Trap");
  }
}

/* 4-5-6-7 Curve Button */
if (index == (23 + 4 * number_segments))
{
  if (picked != 0)
  {
    forward_it (&prof, (picked + 1));
  }
  if (strcmp (prof->seg_type, "D") != 0)
  {
    strcpy (prof->curve_type, "4-5-6-7 Curve");
    prof->curve_type_no = 9;
    dg_set_radio_text (dbox, 11, picked, "");
    dg_set_radio_text (dbox, 11, picked, "4-5-6-7");
  }
}

/* fool the loop into exiting (ok = cancel key right now) */
if (index == 0)
{
  /* start of ok if loop */
  /* initialize these check variables */
  angle_total = total_lift = 0.0;
  flag1 = flag2 = flag3 = 0;

  for (other_d = 1; other_d <= number_segments; other_d++)

Appendix A: Source Code Listing - PROFILC 220
{ /* start of for loop */
  if (other_d != 1)
  {
    forward_it (&prof, other_d);
  }

  /* set intermediate variable for angle interval total */
  angle_total += prof->angle_range;

  /* set intermediate variable for total follower lift */
  if (strcmp (prof->seg_type, "R") == 0)
  {
    total_lift += prof->stroke_length;
  }
  if (strcmp (prof->seg_type, "F") == 0)
  {
    total_lift -= prof->stroke_length;
  }

  /* check to see if any values haven't been entered */
  /* that will be needed to calculate the cam profile */
  if (prof->curve_type_no == 9999)
  {
    flag1 = 1;
  }

  if (prof->angle_ranges == 9999)
  {
    flag2 = 1;
  }

  if (prof->stroke_length == 9999)
  {
    flag3 = 1;
  }

  /* rewind the structure pointer */
  rewind_profile (&prof);
} /* end of for loop */

if (flag1 != 0)
{
  printf (prompt,
    "You haven't set some curve types...ENTER to continue");
  ck_pause (prompt);
  printf (prompt, "");
  ck_prompt (prompt);
  break;
}
else
if (flag2 != 0)
{ 
sprintf (prompt, 
    "You haven't set some angle ranges...ENTER to continue");
cp_pause (prompt);
sprintf (prompt, " ");
cp_prompt (prompt);
break;
}
else
if (flag3 != 0) 
{
    sprintf (prompt, 
        "You haven't set some stroke lengths...ENTER to continue");
cp_pause (prompt);
sprintf (prompt, " ");
cp_prompt (prompt);
break;
}
else
if (angle_total != 360.0) 
{
    sprintf (prompt, 
        "Total angle interval <> 360°...Please correct...");
cp_pause (prompt);
sprintf (prompt, " ");
cp_prompt (prompt);
break;
}
else
if (total_lift != 0) 
{
    sprintf (prompt, 
        "Total lift (rise & fall) <> 0...Please correct...");
cp_pause (prompt);
sprintf (prompt, " ");
cp_prompt (prompt);
break;
}
else
{
    index = cancel;
cam_defined = 1;
analysis->k1 = 0.0;
}
} /* end of ok if loop */

/* rewind the cam motion profile structure */
rewind_profile (&prof);

} /* end of for loop */
} /* end of dialog box running (while) loop */
void edit_cam (void)
{
    DialogBox *db = NULL;
    int choice, old_follower_type, type_flag, specs_flag, change_flag;
    double param1, param2, param3, param4, param5, old_rotation;
    char old_motion_profile[10];

    /* initialize dialog box choice and other flags */
    choice = type_flag = specs_flag = change_flag = 0;

    if (curve->next_one == NULL)
    {
        user_warning();
        return;
    }

    /* set the edit flag to 1 */
    edit_flag = 1;

    /* initialize the dialog box parameters */
    db = dg_init_dialog (9, 42, 9, "Modify Cam", 0, 0, 4, 0);
    dg_add_button (db, 0, 2.75, 3.00, 0, "Follower Type", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 1, 4.50, 2.00, 0, "Motion Segments", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 2, 2.75, 24.00, 0, "Motion Program", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 3, 4.50, 24.00, 0, "Cam Parameters", 0, 0, DG_RET_ON_SEL);
    dg_add_box (db, 4, 6.25, 0.50, 0.05, 40.90, NULL, 0, 0, 0);
    dg_add_button (db, 5, 7.10, 5.00, 8, "OK", 0, 0, DG_RET_ON_SEL);
    dg_add_button (db, 6, 7.10, 29.00, 8, "Cancel", 0, 0, DG_RET_ON_SEL);
    dg_add_note (db, 7, 0.40, 8.20, "Choose category to modify!", 0, 0, 0);
    dg_add_box (db, 8, 1.25, 8.00, 0.05, 25.90, NULL, 0, 0, 0);
dg_move_focus (db, 4);

/* draw the dialog box */
dg_draw_dialog (db);

/* start of dialog box running loop */
do
{    /* start of while (dialog box running) loop */
    /* returns the index of the option selected by the user */
    choice = dg_run_dialog (db);

    switch (choice)
    {       /* start of index switch loop */
        case 0:
        {    /* get a flag if the follower type changed */
            if (old_follower_type != cam-type_num)
            {    /* type_flag = 1; */
                specs_flag = 0;
                change_flag = 1;
            }

            if ((old_rotation * cam->omega) < 0.0)
            {    /* change_flag = 1; */
                }

            dg_move_focus (db, 4);
            dg_draw_dialog (db);
            break;
        }
        case 1:
        {    /* if the motion segments have changed, then have the user */
            strcpy (old_motion_profile, motion_profile);
            motion_segs();

            /* check/re-do the motion profile */
            if (strcmp (old_motion_profile, motion_profile) != 0)
            {    /* calculate_cam(); */
                change_flag = 1;
            }

Appendix A: Source Code Listing - PROFIL.C 224
dg_move_focus (db, 4);
dg_draw_dialog (db);
break;
}
}
case 2:
{
dg_erase_dialog (db);
enter_motion_program();
calculate_cam();
change_flag = 1;
dg_move_focus (db, 4);
dg_draw_dialog (db);
break;
}
case 3:
{
dg_erase_dialog (db);
profile_specs (cam->type_num, cam->follower_type, &param1, 
&param2, &param3, &param4, &param5);
specs_flag = 1;
change_flag = 1;
dg_move_focus (db, 4);
dg_draw_dialog (db);
break;
}
}
}
while ((choice != 5) && (choice != 6));

/* must enter new cam parameters for new follower type */
if (((type_flag == 1) && (specs_flag == 0))
{
profile_specs (cam->type_num, cam->follower_type, &param1, &param2,
&param3, &param4, &param5);
specs_flag = 1;
}

/* erase and free memory from dialog box */
dg_erase_dialog (db);
dg_free_dialog (db);

if (((choice == 5) && (change_flag == 1))
{
/* autoscale and redraw the cam before erasing it (to be sure to get */
/* the whole cam view in the viewport) */
ck_set (CK_SET_SCALE, CK_PRIME_vp, scale_cam, 0.0, 0.0);
ck_redraw (CK_PRIME_vp);

/* erase the screen before starting */
erase_screen (CK_PRIME_vp);
/* go to the function to actually draw a rendering of the cam on */
/* the CADKEY screen */
cam_render (&curve);

/* rewind the svaj/curve data structure to the beginning */
rewind_curve (&curve);
}

/* reset the edit flag to 0 (done editing) */
edit_flag = 0;
} /* end of edit_cam */
/******************************

{ /* start of cam render */
  double x0, y0, z0, sign;
  double c, beta, d, radius, cx, cy, c_len, max_alpha, gammas, dummy_c,
  double l, r, psi1, phi, theta, x_loc1, y_loc1, x_loc2, y_loc2, percent,
  double top_val_x, top_val_y, l_init, first_x, first_y, face_length;
  double x_spec, y_spec, x, gam, h, t, psi2;
  double phi_initial, param1, param2, param3, param4, param5;
  int current_color;

  /* initialize other dummy variables */
  param1 = param2 = param3 = param4 = param5 = first_x = first_y = 0.0;
  x_loc1 = y_loc1 = x_loc2 = y_loc2 = top_val_x = top_val_y = 0.0;
  x_spec = y_spec = 0.0;
  percent = 1.0;

  /* initialize beta (oblique angle + 90°) to 0 */
  beta = 0.0;

  /* set other values starting at zero */
  l = l_init = radius = cx = cy = d = c = max_alpha = phi_initial = 0.0;

  /* set the variable sign to indicate direction of cam rotation */
  if ((cam->omega <= 0.0))
    {
      /* negative angular velocity means counterclockwise rotation, which */
      /* is what was assumed in the conjugate geometry derivation */
      sign = 1.0;
    }
  else
    {
      /* positive angular velocity means clockwise rotation, which is */
      /* opposite to what was assumed in the conjugate geometry derivation */
      sign = -1.0;
    }

  /* get more cam parameters */
  /* this calls the function in SPECS.C to get things like base circle */
  /* radius, roller radius, pivot location, etc. */
  if (analysis->k1 == 0.0)
    {
      profile_specs (cam->type_num, cam->follower_type, &param1, &param2,
          &param3, &param4, &param5);
    }
  else
    /* when the cam is already defined, this part of the routine just */
    /* redraws the cam profile using the same values as before, instead */
    /* of having to recalculate it */
    {
      /* start of else loop */
      param1 = analysis->k1;
    }

Appendix A: Source Code Listing - RENDER.C

228
param2 = analysis->k2;
switch (cam->type_num)
{
    /* start of switch loop */
    case 2:
    {
        param3 = analysis->k3;
        break;
    }
    case 3:
    {
        param3 = analysis->k3;
        break;
    }
    case 4:
    {
        param3 = analysis->k3;
        param4 = analysis->k4;
        param5 = analysis->k5;
        break;
    }
    /* end of switch loop */
    /* end of else loop */
}
/* if cancel button was pressed, then exit out of loop */
if (param5 == 9999)
{
    return;
}

/* set the value for the base circle (from the profile_specs function) */
c = param1;

/* set up center point for hub of cam */
x0 = 0.0;
y0 = 0.0;
z0 = 0.0;

/* set current color = 0 */
current_color = 0;

/* scale the drawing so it fills the screen */
if (nc_flag == 0)
{
    scale_cam = (130.0) / (2.0 * (c + cam->max_lift)) * 0.55;
    ck_set (CK_SET_SCALE, CK_PRIME_VP, scale_cam, 0.0, 0.0);
}

/* redraw the part when autoscaled */
if (nc_flag == 0)
{
    ck_redraw (CK_PRIME_VP);
/* draw the center point for the cam hub (in a different color) */
if (nc_flag == 0)
{
    ck_set (CK_SET_COLOR, 1);
    ck_point (x0, y0, z0, NULL);
}

/* set the line type to dashed lines (type 2) for the base circle */
/* and draw the base circle for the cam profile */
if (nc_flag == 0)
{
    ck_set (CK_SET_LINETYPE, 2);
    ck_circle (x0, y0, z0, c, NULL);
}

/* set the line type back to solid lines (type 1) and set the line */
/* width to extra thick (width = 5) for the cam surface */
if (nc_flag == 0)
{
    ck_set (CK_SET_LINETYPE, 1);
    ck_set (CK_SET_LINEWIDTH, 5);
}

switch (cam->type_num)
{
    /* start of switch loop */
    case 1:
    {
        /* Start of Translating Flat-Faced Follower */
        /* the conjugate geometry definitions begin here */
        /* beta is the oblique angle plus 90° */
        beta = (param2 + 90.0) * conv1;

        /* the vector r is the vector from the cam center to the flat */
        /* face of the follower, in the direction of the follower stem */
        /* the 'sin (pi - beta)' is a necessary adjustment factor that */
        /* comes into play for all oblique followers */
        r = (c / sin (pi - beta)) + (*temp)->s;

        /* this is the final conjugate geometry result for this case */
        l = (*temp)->v * sin (beta) - r * cos (beta);

        /* converts theta (camshaft angle) to radians */
        theta = (*temp)->index * conv1 * sigs;

        /* solves for the pressure angle (from the conjugate geometry */
        /* results) for each position of the cam */
        pressure->p_angle = analysis->k6 = fabs ((beta / conv1) - 90.0);
        pressure->theta = analysis->k7 = (*temp)->index;

        /* the P vector is defined here - the x and y position of */
/* the cam profile for 0 to 360° */
x_loc1 = first_x = r * cos(theta) + l * cos(theta + beta);
top_val_y = y_loc1 = first_y = r * sin(theta) + l * sin(theta + beta);

/*@ save this data to the nc data structure if */
/*@ NC data being produced */
if (nc_flag == 1)
{
    /* save the cam profile data in the structure */
    nc->x = x_loc1;
    nc->y = y_loc1;
    nc->angle = atan2(y_loc1, x_loc1);
}

/*@ set the color */
if (nc_flag == 0)
{
    current_color = (*temp)->color;
    ck_set(CK_SET_COLOR, current_color);
}

/*@ repeat while still in list - as long as there's still another */
/*@ point of the cam profile to draw */
while ((*temp)->next_one != NULL)
{
    /* start of while loop */
    (*temp) = (*temp)->next_one;
    r = (c / sin(pi - beta)) + (*temp)->s;
    l = (*temp)->v * sin(beta) - r * cos(beta);
    theta = (*temp)->index * conv1 * sign;
    x_loc2 = r * cos(theta) + l * cos(theta + beta);
    y_loc2 = r * sin(theta) + l * sin(theta + beta);

    /* save the nc data to a structure if flag is set */
    if (nc_flag == 1)
    {
        /* set up for dynamic memory allocation of structure */
        nc->next = (struct nc_data *) malloc(sizeof(struct nc_data));

        /* give a reference for the previous nc_data values */
        nc->next->prev = nc;

        /* advances pointer, so that we can add another set of */
        /* NC data to the nc structure */
        nc = nc->next;

        /* save the cam profile data in the structure */
        nc->x = x_loc2;
        nc->y = y_loc2;
        nc->angle = atan2(y_loc2, x_loc2);
    }
}
/* set up for dynamic memory allocation of pressure */
/* angle structure */
p{next} = (struct pressure_analysis*) malloc
(sizeof(struct pressure_analysis));

/* give a reference for the previous pressure angle values */
p{next}->prev = pressure;

/* advances pointer, so that we can add another pressure */
/* angle value can be added to the pressure structure */
p{next} = p{next};

/* set the value for the pressure angle */
p{p_angle} = fabs ((beta / conv1) - 90.0);
p{theta} = (*temp)->index;

/* tests for new top_val (maximum y value of cam profile) */
if (y{loc2} > top_val{y})
{
    top_val{y} = y{loc2};
    top_val{x} = x{loc2};
}

/* tests for maximum pressure angle value */
if (p{p_angle} > analysis{k6})
{
    analysis{k6} = p{p_angle};
    analysis{k7} = p{theta};
}

/* set the color, if different */
if (nc_flag == 0)
{
    if (*current{color} != (*temp)->color)
    {
        current{color} = (*temp)->color;
        ck_set (CK_SET_COLOR, current{color});
    }
}

/* draw the line segment on the CADKEY screen */
if (nc_flag == 0)
{
    ck_line (x{loc1} + x{loc1}, y{loc1} + y{loc1}, z{0}, x{loc2} + x{loc2}, y{loc2} + y{loc2}, z{0}, NULL);
}

/* set the new value for point 1 equal to the former point 2 */
x{loc1} = x{loc2};
y{loc1} = y{loc2};
} /* end of while loop */

/* set max pressure angle value (so that its between 0 and 180°) */
while (analysis->k6 > 180.0)
{
    analysis->k6 -= 180.0;
}

/* set the nc structure for the last value in the structure */
if (nc_flag == 1)
{
    nc->next = NULL;
}
break;
} /* End of Translating Flat-Faced Follower */

case 2:
{
    /* Start of Oscillating Flat-Faced Follower */
    /* the conjugate geometry definitions begin here */
    /* these are the x and y locations of the pivot, measured */
    /* from the cam center location */
    cx = param2;
    cy = param3;

    /* this is the magnitude of the vector c, or the straight line */
    /* distance from the cam center to the pivot location of the */
    /* oscillating follower arm */
    c_len = sqrt(cx * cx + cy * cy);

    /* gamma is the angle of the vector c, between the cam center and */
    /* the pivot location of the follower */
    gamma = atan2(cy, cx);

    /* this defines the value for the initial angle of the follower */
    /* arm with the cam - all rises and falls are measured from this */
    phi_initial = gamma + pi - asin(c / c_len);

    /* defines the rise/fall of the follower arm (a rise is shown */
    /* by a negative s value) */
    phi = phi_initial - (*temp)->s;

    /* this is the final conjugate geometry result for this case */
    l = l_init = (-c_len * cos(phi)) / (1 - (*temp)->v);

    /* converts theta (camshaft angle) to radians */
    theta = (**temp)->index * conv1 * sign;

    /* solves for the pressure angle (from the conjugate geometry */
    /* results) for each position of the cam */
    pressure->p_angle = analysis->kt = 0.0;
    pressure->theta = analysis->k7 = (**temp)->index;
/* the P vector is defined here - the x and y position of */
/* the cam profile for 0 to 360 degrees */
x_loc1 = first_x = c_len * cos (theta + gamma) + l * cos (theta + phi + gamma);
top_val_y = y_loc1 = first_y = c_len * sin (theta + gamma) + cy + l * sin (theta + phi + gamma);

/* save this data to the nc data structure if */
/* NC data being produced */
if (nc_flag == 1)
{
    /* save the cam profile data in the structure */
    nc->x = x_loc1;
    nc->y = y_loc1;
    nc->angle = atan2 (y_loc1, x_loc1);
}

/* set the color */
if (nc_flag == 3)
{
    current_color = (*temp)->color;
    ck_set (CK_SET_COLOR, current_color);
}

/* repeat while still in list - as long as there's still another */
/* point of the cam profile to draw */
while ((*temp)->next_one != NULL)
{
    /* start of while loop */
    (*temp) = (*temp)->next_one;
    phi = phi_initial - (*temp)->s;
    l = (-c_len * cos (phi)) / (1 - (*temp)->v);
    theta = (*temp)->index * conv1 * sign;
    x_loc2 = c_len * cos (theta + gamma) + l * cos (theta + phi + gamma);
    y_loc2 = c_len * sin (theta + gamma) + cy + l * sin (theta + phi + gamma);

    /* save the nc data to a structure if flag is set */
    if (nc_flag == 1)
    {
        /* set up for dynamic memory allocation of structure */
        nc->next = (struct nc_data *) malloc (sizeof(struct nc_data));

        /* give a reference for the previous nc_data values */
        nc->next->prev = nc;

        /* advances pointer, so that we can add another set of */
        /* NC data to the nc structure */
        nc = nc->next;

Appendix A: Source Code Listing - RENDER.C
/* save the cam profile data in the structure */
nc->x = x_loc2;
nc->y = y_loc2;
nc->angle = atan2 (y_loc2, x_loc2);
}

/* set up for dynamic memory allocation of pressure */
/* angle structure */
pressure->next = (struct pressure_analysis *) malloc
(sizeof(struct pressure_analysis));

/* give a reference for the previous pressure angle values */
pressure->next->prev = pressure;

/* advances pointer, so that we can add another pressure */
/* angle value can be added to the pressure structure */
pressure = pressure->next;

/* set the value for the pressure angle */
pressure->p_angle = 0.0;
pressure->theta = (*temp)->index;

/* tests for new top_val (maximum y value of cam profile) */
if (y_loc2 > top_val_y)
{
    top_val_y = y_loc2;
    top_val_x = x_loc2;
}

/* tests for maximum pressure angle value */
if (pressure->p_angle > analysis->k6)
{
    analysis->k6 = pressure->p_angle;
    analysis->k7 = pressure->theta;
}

/* set the color, if different */
if (nc_flag == 0)
{
    if (current_color != (*temp)->color)
    {
        current_color = (*temp)->color;
        ck_set (CK_SET_COLOR, current_color);
    }
}

/* draw the line segment on the CADKEY screen */
if (nc_flag == 0)
{
    ck_line (x0 + x_loc1, y0 + y_loc1, z0, x0 + x_loc2, y0 + y_loc2, z0, NULL);
} /* set the value for point 1 equal to the former point 2 */
  x_loc1 = x_loc1;
y_loc1 = y_loc2;
} /* end of while loop */

/* set max pressure angle value (so that its between 0 and 180°) */
while (analysis->k6 > 180.0)
{
  analysis->k6 = 180.0;
}

/* set the nc structure for the last value in the structure */
if (nc_flag == 1)
{
  nc->next = NULL;
}

break;
} /* End of Oscillating Flat-Faced Follower */

/* Start of Translating Roller Follower */
/* the conjugate geometry definitions begin here */
/* d is the offset of the follower stem from a line parallel */
/* to the follower stem passing through the cam center */
d = param2;

/* this is the radius of the roller follower */
radius = param3;

/* the r vector is usually c + s + radius of roller */
/* this is also dependent on the offset value, and makes this */
/* expression quite complex */
x = c + (**temp)->s + radius;
gam = aain (d / x);
h = sqrt (2.0 * pow (x, 2.0) * (1.0 - cos (gam)));
t = sqrt (pow (h, 2.0) - pow (d, 2.0));
r = x - t;

/* this is the final conjugate geometry result for this case */
psi = psi2 = atan2 ((d - (**temp)->v), r);

/* adjust the psi value */
if (psi < (pi / 2.0))
{
  psi += pi;
}

/* convert theta (camshaft angle) to radians */
theta = (**temp)->index * conv1 * sign;
/* solves for the pressure angle (from the conjugate geometry */
/* results for each position of the cam */
pressure->p_angle = analysis->k6 = fabs((psi / conv1) - 180.0);
pressure->theta = analysis->k7 = (*temp)->index;

/* the P vector is defined here - the x and y position of */
/* the cam profile for 0 to 360 */
x_loc1 = first_x = r * cos(theta) + d * cos(theta + (pi / 2.0)) +
radius * cos(theta + psi);
top_val_y = y_loc1 = first_y = r * sin(theta) + d * sin(theta + (pi
/ 2.0)) + radius * sin(theta + psi);

/* save this data to the nc data structure if */
/* NC data being produced */
if (nc_flag == 1)
{
    /* save the cam profile data in the structure */
    nc->x = x_loc1;
    nc->y = y_loc1;
    nc->angle = atan2(y_loc1, x_loc1);
}

/* set the color */
if (nc_flag == 0)
{
    current_color = (*temp)->color;
    ck_set (CK_SET_COLOR, current_color);
}

/* repeat while still in list - as long as there's still another */
/* point of the cam profile to draw */
while ((*temp)->next_one != NULL)
{
    /* start of while loop */
    (*temp) = (*temp)->next_one;
    x = c + (*temp)->s + radius;
gam = asin(d / x);
h = sqrt(2.0 * pow(x, 2.0) * (1.0 - cos(gam)));
t = sqrt(pow(h, 2.0) - pow(d, 2.0));
r = x - t;
psi = psi2 = atan2((d - (*temp)->v), r);
if (psi < (pi / 2.0))
{
    psi += pi;
}

theta = (*temp)->index * conv1 * sign;
x_loc2 = r * cos(theta) + d * cos(theta + (pi / 2.0)) + radius *
    cos(theta + psi);
y_loc2 = r * sin(theta) + d * sin(theta + (pi / 2.0)) + radius *
    sin(theta + psi);
/* save the nc data to a structure if flag is set */
if (nc_flag == 1)
{
    /* set up for dynamic memory allocation of structure */
    nc->next = (struct nc_data *) malloc (sizeof(struct nc_data));

    /* give a reference for the previous nc_data values */
    nc->next->prev = nc;

    /* advances pointer, so that we can add another set of */
    /* NC data to the nc structure */
    nc = nc->next;

    /* save the cam profile data in the structure */
    nc->x = x_loc2;
    nc->y = y_loc2;
    nc->angle = atan2 (y_loc2, x_loc2);
}

/* gets the 90 degree position in order to draw in the */
/* roller follower */
if (sign == 1.0)
{
    if (((*temp)->index < 90.10) && ((*temp)->index > 89.90))
    {
        x_spec = x_loc2;
        y_spec = y_loc2;
    }
}
else
{
    if (((*temp)->index < 270.10) && ((*temp)->index > 269.90))
    {
        x_spec = x_loc2;
        y_spec = y_loc2;
    }
}

/* set up for dynamic memory allocation of pressure */
/* angle structure */
power->next = (struct pressure_analysis *) malloc
(sizeof(struct pressure_analysis));

/* give a reference for the previous pressure angle values */
power->next->prev = power;

/* advances pointer, so that we can add another pressure */
/* angle value can be added to the pressure structure */
power = power->next;
/* set the value for the pressure angle */
pressure->p_angle = fabs((ps12 / ccnv1) - 180.0);
pressure->theta = (*temp)->index;

/* tests for new top_val (maximum y value of cam profile) */
if (y_loc2 > top_val_y)
{
    top_val_y = y_loc2;
    top_val_x = x_loc2;
}

/* tests for maximum pressure angle value */
if (pressure->p_angle > analysis->k6)
{
    analysis->k6 = pressure->p_angle;
    analysis->k7 = pressure->theta;
}

/* set the color, if different */
if (nc_flag == 0)
{
    if (current_color != (*temp)->color)
    {
        current_color = (*temp)->color;
        ck_set (CK_SET_COLOR, current_color);
    }
}

/* draw the line segment on the CADKEY screen */
if (nc_flag == 0)
{
    ck_line (x0 + x_loc1, y0 + y_loc1, z0, x0 + x_loc2, y0 + y_loc2, z0, NULL);
}

/* set the new value for point 1 equal to the former point 2 */
x_loc1 = x_loc2;
y_loc1 = y_loc2;
} /* end of while loop */

/* set max pressure angle value (so that its between 0 and 180°) */
while (analysis->k6 > 180.0)
{
    analysis->k6 -= 180.0;
}

/* set the nc structure for the last value in the structure */
if (nc_flag == 1)
{
    nc->next = NULL;
}
break;
}

} /* End of Translating Roller Follower */

case 4:
{
  /* Start of Oscillating Roller Follower */
  /* the conjugate geometry definitions begin here */
  /* this is the radius of the roller follower */
  radius = param2;

  /* these are the x and y locations of the pivot, measured */
  /* from the cam center location */
  cx = param3;
  cy = param4;

  /* this is the length of the follower arm, measured from the */
  /* pivot location to the center of the roller pin */
  i = l_init = param5;

  /* this is the magnitude of the vector c, or the straight line */
  /* distance from the cam center to the pivot location of the */
  /* oscillating follower arm */
  c_len = sqrt((cx * cx + cy * cy));

  /* gamma is the angle of the vector c, between the cam center and */
  /* the pivot location of the follower */
  gamma = atan2(cy, cx);

  /* this defines the value for the initial angle of the follower */
  /* arm with the cam - all rises and falls are measured from this */
  phi_initial = gamma + pi - acos((1.0 + c_len * c_len - pow((c +
           radius), 2.0)) / (2.0 * l * c_len));

  /* defines the rise/fall of the follower arm (a rise is shown */
  /* by a negative s value) */
  phi = phi_initial - (*temp)->s;

  /* this is the final conjugate geometry result for this case */
  psi = atan2((c_len * sin(phi)), (-c_len * cos(phi) + i * (1 -
           (*temp)->v)));

  /* convert theta (camshaft angle) to radians */
  theta = (*temp)->index * convl * sign;

  /* solves for the pressure angle (from the conjugate geometry */
  /* results) for each position of the cam */
  pressure->p_angle = analysis->k6 = fabs((psi / convl) - 90.0);
  pressure->ttheta = analysis->k7 = (*temp)->index;

  /* the P vector is defined here - the x and y position of */
  /* the cam profile for 0 to 360 */
  x_loc1 = first_x = c_len * cos(theta + gamma) + l * cos(theta + phi

Appendix A: Source Code Listing - RENDER.C
+ gamma) + radius * cos (theta + phi + psi + gamma);

top_val_y = y_loc1 = first_y = c_len * sin (theta + gamma) + cy + l
* sin (theta + phi + gamma) + radius * sin (theta + phi + psi + gamma);

/* save this data to the nc data structure if */
/* NC data being produced */
if (nc_flag == 1)
{
/* save the cam profile data in the structure */
nc->x = x_loc1;
nc->y = y_loc1;
nc->angle = atan2 (y_loc1, x_loc1);
}

/* set the color */
if (nc_flag == 0)
{
current_color = (**temp)->color;
ck_set (CK_SET_COLOR, current_color);
}

/* repeat while still in list - as long as there's still another */
/* point of the cam profile to draw */
while (**temp)->next_one != NULL)
{
/* start of while loop */
(**temp) = (**temp)->next_one;
phi = phi_initial - (**temp)->s;
psi = atan2 ((c_len * sin (phi)), (-(-c_len * cos (phi) + l * (1 - (**temp)->v))));
theta = (**temp)->index * convl * sign;
x_loc2 = c_len + cos (theta + gamma) + l * cos (theta + phi + gamma) + radius * cos (theta + phi + psi + gamma);
y_loc2 = c_len * sin (theta + gamma) + cy + l * sin (theta + phi + gamma) + radius * sin (theta + phi + psi + gamma);

/* save the nc data to a structure if flag is set */
if (nc_flag == 1)
{
/* set up for dynamic memory allocation of structure */
nc->next = (struct nc_data *) malloc (sizeof(struct nc_data));

/* give a reference for the previous nc_data values */
nc->next->prev = nc;

/* advances pointer, so that we can add another set of */
/* NC data to the nc structure */
nc = nc->next;

/* save the cam profile data in the structure */
nc->x = x_loc2;
nc->y = y_loc2;
num_angle = atan2 (y_loc2, x_loc2);

/* set up for dynamic memory allocation of pressure */
/* angle structure */
pressure->next = (struct pressure_analysis *) malloc
(sizeof(struct pressure_analysis));

/* give a reference for the previous pressure angle values */
pressure->next->prev = pressure;

/* advances pointer, so that we can add another pressure */
/* angle value can be added to the pressure structure */
pressure = pressure->next;

/* set the value for the pressure angle */
pressure->p_angle = fabs ((psi / convl) - 90.0);
pressure->theta = (*temp)->index;

/* tests for new top_val (maximum y value of cam profile) */
if (y_loc2 > top_val_y)
{
  top_val_y = y_loc2;
  top_val_x = x_loc2;
}

/* tests for maximum pressure angle value */
if (pressure->p_angle > analysis->k6)
{
  analysis->k6 = pressure->p_angle;
  analysis->k7 = pressure->theta;
}

/* set the color, if different */
if (nc_flag != 0)
{
  if (current_color != (*temp)->color)
  {
    current_color = (*temp)->color;
    ch_set (CK_SET_COLOR, current_color);
  }
}

/* draw the line segment on the CADKEY screen */
if (nc_flag == 0)
{
  ch_line (x0 + x_loc1, y0 + y_loc1, z0, x0 + x_loc2, y0 +
           y_loc2, z0, NULL);
}

/* set the new value for point 1 equal to the former point 2 */
x_loc1 = x_loc2;
y_loc1 = y_loc2;
} /* end of while loop */

/* set max pressure angle value (so that its between 0 and 180\degree) */
while (analysis->k6 > 180.0)
{
    analysis->k6 -= 180.0;
}

/* set the nc structure for the last value in the structure */
if (nc_flag == 1)
{
    nc->next = NULL;
}

break;
} /* End of Oscillating Roller Follower */
} /* end of switch loop */

/* rewind pressure angle structure to beginning */
rewind_pressure (&pressure);

/* set the edit cam flag to 1 - since now the user can edit the design */
edit_flag = 1;

/* draw the last line segment on the CADKEY screen */
if (nc_flag == 0)
{
    ck_line (x_loc2, y_loc2, z0, first_x, first_y, z0, NULL);
}

/* set the line width back to normal (width = 1) */
if (nc_flag == 0)
{
    ck_set (CK_SET_LINEWIDTH, 1);
}

/* draw the cam hub */
if (nc_flag == 0)
{
    ck_set (CK_SET_COLOR, 13);
}

/* the size of the hub is parametrically based on the base circle radius */
dummy_c = 0.17 * c;

if (rc_flag == 0)
{
    switch (cam->hub_type)
    { /* start of switch loop */
case 1: /* vertical (0\degree) hub */
{
    /* start of case #1 */
    ck_arc (x0, y0, z0, dummy_c, 290.0, 610.0, NULL);
    ck_line (x0 + dummy_c * cos (610.0 * conv1), y0 + dummy_c * sin (610.0 * conv1), 0.0, x0 + dummy_c * cos (610.0 * conv1), y0 + dummy_c * sin (610.0 * conv1) - dummy_c * 0.5, 0.0, NULL);
    ck_line (x0 + dummy_c * cos (290.0 * conv1), y0 + dummy_c * sin (290.0 * conv1), z0, x0 + dummy_c * cos (290.0 * conv1), y0 + dummy_c * sin (290.0 * conv1) - dummy_c * 0.5, z0, NULL);
    ck_line (x0 + dummy_c * cos (610.0 * conv1), y0 + dummy_c * sin (610.0 * conv1) - dummy_c * 0.5, z0, x0 + dummy_c * cos (290.0 * conv1), y0 + dummy_c * sin (290.0 * conv1) - dummy_c * 0.5, z0, NULL);
    ck_circle (x0, y0, z0, 1.9 * dummy_c, NULL);
    break;
} /* end of case #1 */

/* horizontal (90\degree) hub */
{
    /* start of case #2 */
    ck_arc (x0, y0, z0, dummy_c, 20.0, 340.0, NULL);
    ck_line (x0 + dummy_c * cos (340.0 * conv1), y0 + dummy_c * sin (340.0 * conv1), 0.0, x0 + dummy_c * cos (340.0 * conv1) + dummy_c * 0.5, y0 + dummy_c * sin (340.0 * conv1), 0.0, NULL);
    ck_line (x0 + dummy_c * cos (20.0 * conv1), y0 + dummy_c * sin (20.0 * conv1), z0, x0 + dummy_c * cos (20.0 * conv1) + dummy_c * 0.5, y0 + dummy_c * sin (20.0 * conv1), z0, NULL);
    ck_line (x0 + dummy_c * cos (340.0 * conv1) + dummy_c * 0.5, y0 + dummy_c * sin (340.0 * conv1), z0, x0 + dummy_c * cos (20.0 * conv1) + dummy_c * 0.5, y0 + dummy_c * sin (20.0 * conv1), z0, NULL);
    ck_circle (x0, y0, z0, 1.9 * dummy_c, NULL);
    break;
} /* end of case #2 */
/* end of switch loop */

/* draw the follower */
ck_set (CK_SET_COLOR, 9);

switch (cam->type_num)
{
    /* start of switch loop */
    case 1:
    {
    /* calculate length of flat-faced follower */
    face_length = c * 0.85;

    /* set the line width to pretty wide (width = 3) */
    ck_set (CK_SET_LINEWIDTH, 5);

    /* draw the follower face */
    ck_line (top_val_x + (face_length / 2.0), top_val_y + 2.0, z0, top_val_x - (face_length / 2.0), top_val_y + 2.0, z0, NULL);
/* set the line width to a little wide (width = 3) */
ck_set (CK_SET_LINEWIDTH, 3);

/* draw the follower stem */
ck_line (top_val_x, top_val_y + 2.0, z0, top_val_x + c * 1.10 * cos ((180.0 - (beta / convl)) * convl), top_val_y + c * 1.10 * sin ((180.0 - (beta / convl)) * convl) + 2.0, z0, NULL);

break;
}

case 2:
{
/* calculate length of flat-faced follower */
face_length = c * 0.85;

/* set the face length scale factor */
percent = ((face_length / 2.6) / l_init);

/* set the line width to pretty wide (width = 5) */
ck_set (CK_SET_LINEWIDTH, 5);

/* draw the follower face */
ck_line (cx + l_init * cos (phi_initial) * (1.0 + percent), cy + l_init * sin (phi_initial) * (1.0 + percent), z0, cx + l_init * cos (phi_initial) * (1.0 - percent), cy + l_init * sin (phi_initial) * (1.0 - percent), z0, NULL);

/* set the line width to a little wide (width = 3) */
ck_set (CK_SET_LINEWIDTH, 3);

/* draw the follower arm */
ck_line (cx, cy, z0, cx + l_init * cos (phi_initial) * (1.0 - percent), cy + l_init * sin (phi_initial) * (1.0 - percent), z0, NULL);

/* change colors for the follower pivot point */
ck_set (CK_SET_COLOR, 8);

/* draw the follower pivot point */
ck_point (cx, cy, z0, NULL);

break;
}

case 3:
{
/* set the line width to pretty wide (width = 5) */
ck_set (CK_SET_LINEWIDTH, 5);

/* draw the roller follower */
ck_circle (top_val_x, top_val_y + radius + 2.0, z0, radius, NULL);

 Appendix A: Source Code Listing · RENDER.C 245
/* set the line width to a little wide (width = 3) */
ck_set (CK_SET_LINEWIDTH, 3);

/* draw the follower stem */
ck_line (top_val_x, top_val_y + radius + 2.0, z0, top_val_x,
        top_val_y + radius + 2.0 + c * 1.20, z0, NULL);

break;
}
}
case 4:
{
    /* set the line width to pretty wide (width = 5) */
ck_set (CK_SET_LINEWIDTH, 5);

/* draw the roller follower */
ck_circle (cx + l_init * cos (phi_initial), cy + l_init * sin
        (phi_initial), z0, radius, NULL);

/* set the line width to a little wide (width = 3) */
ck_set (CK_SET_LINEWIDTH, 3);

/* draw the follower arm */
ck_line (cx, cy, z0, cx + l_init * cos (phi_initial), cy + l_init
        * sin (phi_initial), z0, NULL);

/* change colors for the follower pivot point */
ck_set (CK_SET_COLOR, 8);

/* draw the follower pivot point */
ck_point (cx, cy, z0, NULL);

break;
}

/* end of switch loop */

/* set the line width back to normal (width = 1) */
if (nc_flag == 0)
{
    ck_set (CK_SET_LINEWIDTH, 1);
}

/* end of cam_render */
Appendix A: Source Code Listing

File: SPECS.C

/*******************************************************************************/
/* Cam Design System - Cam Parameter Specification Functions */
/*******************************************************************************/
/* File: SPECS.C */
/*******************************************************************************/
/* Programmed by: */
/* */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* */
/* c/o Dr. Charles F. Reinholtz */
/* */
/* 106 Randolph Hall */
/* */
/* Department of Mechanical Engineering */
/* */
/* Virginia Tech */
/* */
/* Blacksburg, VA 24061-0238 */
/* */
/*******************************************************************************/
/* Purpose: These functions allow the user to enter specific cam and */
/* follower parameters for full profile design. */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-03-94 */
/*******************************************************************************/

/* necessary include files */
#include <stdio.h>
#include <string.h>
#include "ck_cdl.h" /* for standard CADL functions */
#include "ck_dig.h" /* for dialog boxes */
#include "global.h" /* Cam Design System global variable header file */
#include "cam.h" /* Cam Design System main header file */

/*******************************************************************************/
void cam_params (void)
/*******************************************************************************/
/* Purpose: Allows the user to modify the cam parameters, such as the */
/* angular velocity (omega), and the follower type. */
/* */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-03-94 */
/*******************************************************************************/
{ /* start of cam_params */
    DIALOG_Box *db = NULL;
    static char *radioset1[] = {"Trans. Flat", "Oscill. Flat",
        "Trans. Roller", "Oscill. Roller"};
    static char *radioset2[] = {"CCW [-]", "CW [+]"};
    int d, picked, old_picked, index, default_follower_type_num;
    int picked2, old_picked2, choice;
    double rows[5], columns[5], default_w;
    char default_follower_type[25];

    /* if there is a cam already defined, and this isn't the edit function */
    /* prompt user to start a new cam design or not */
    if ((cam_defined == 1) && (edit_flag != 1))
    {
        /* initialize dialog box choice */
        choice = 0;

        /* make new dialog box to indicate that there is no cam data */
        db = dg_init_dialog (6, 27, 3, "New Cam", 0, 0, 4, 0);
        dg_add_note (db, 0, 1.50, 2.0, "Create new cam profile?", 0, 0, 0);
        dg_add_button (db, 1, 4.00, 3.00, 8, "Yes", 0, 0, DG_RET_ON_SEL);
        dg_add_button (db, 2, 4.00, 16.25, 8, "No", 0, 0, DG_RET_ON_SEL);

        /* draw the dialog box */
        dg_draw_dialog (db);
        do { /* start of do-while loop */
            /* run the dialog box */
            choice = dg_run_dialog (db);
        } /* end of do-while loop */
        while ((choice != 1) && (choice != 2));

        /* erase and free memory from dialog box */
        dg_erase_dialog (db);
        dg_free_dialog (db);

        /* go to start_new_cam function if user says it's okay */
        if (choice == 1)
        {
            edit_flag = 1;
            start_new_cam();
            edit_flag = 0;
        }

        /* abort this function if user wants to keep the current design */
        if (choice == 2)
        {
            return;
        }
    }

Appendix A: Source Code Listing - SPECS.C 248
/* set default follower type to Translating Flat-Faced */
/* (if not already set) */
if (cam->type_num == 1)
{
    picked = 0;
}
else
if (cam->type_num == 2)
{
    picked = 1;
}
else
if (cam->type_num == 3)
{
    picked = 2;
}
else
if (cam->type_num == 4)
{
    picked = 3;
}
else
{
    picked = 0;
    cam->type_num = 1;
    strcpy (cam->follower_type, "Translating Flat-Faced");
}

/* set defaults (when entering the function), to be used later */
/* for the 'Cancel' option */
default_w = cam->omega;
default_follower_type_num = cam->type_num;
strcpy (default_follower_type, cam->follower_type);

/* set the new value in this variable */
old_picked = picked;

/* set this value to be displayed */
if (cam->omega < 0.0)
{
    picked2 = 0;
}
else
{
    picked2 = 1;
}

/* set the new value in this variable */
old_picked2 = picked2;
/** initialize dialog box */
db = dg_init_dialog (15, 58, 14, "Cam Parameters", 0, 0, 4, 0);
dg_add_note (db, 0, 0.40, 22.20, "Follower Type:", 0, 0, 0);
dg_add_radio (db, 1, 3.60, 11.30, picked, 4, radioSet1, 0, 0, 0);
dg_add_note (db, 2, 13.00, 1.50, "Cam Rotacion:", 0, 0, 0);
dg_add_radio (db, 3, 13.00, 16.75, picked2, 2, radioSet2, 0, 0, 0);
dg_add_box (db, 5, 1.30, 22.20, 0.05, 13.75, NULL, 0, 0, 0);
dg_add_box (db, 6, 12.00, 0.60, 0.05, 56.90, NULL, 0, 0, 0);

/** get data for icons in the dialog box */
intialize_icons (db);

dg_add_button (db, 11, 12.95, 39.05, 8, "OK", 0, 0, 01);
dg_add_button (db, 12, 12.95, 48.00, 8, "Cancel", 0, 0, 01);
dg_add_box (db, 13, 12.00, 37.30, 2.80, 0.24, NULL, 0, 0, 0);

/** set up the alignment of the rows and columns for the 1st radio box */
for (d = 1; d <= 4; d++)
{
    if (d <= 2)
    {
        rows[d - 1] = (d - 1) * 4.85;
        columns[d - 1] = 0.00;
    }
    else
    {
        rows[d - 1] = (d - 3) * 4.85;
        columns[d - 1] = 28.90;
    }
}

/** align the radio box */
dg_radio_align (db, 1, rows, columns);

/** set up the alignment of the rows and columns for the 2nd radio box */
for (d = 1; d <= 2; d++)
{
    rows[d - 1] = 0.00;
    columns[d - 1] = ((d - 1) * 11.25);
}

/** align the radio box */
dg_radio_align (db, 3, rows, columns);

/** set the focus to the first curve type available to enter */
dg_set_radio (db, 3, picked2);
dg_set_radio (db, 1, picked);
dg_move_focus (db, 1);

/** draw the dialog box */
dg_draw_dialog (db);
/* start of dialog box running loop */
do {
    /* returns the index of the option selected by the user */
    index = dg_run_dialog (db);

    /* returns the index of the item selected in the radio boxes */
picked = dg_get_radio (db, 1);
picked2 = dg_get_radio (db, 3);

    if (old_picked != picked)
    { /* start of if loop */
        old_picked = picked;
        switch (picked)
        { /* start of picked switch loop */
            case 0:
                {
                    strcpy (cam->follower_type, "Translating Flat-Faced");
                    cam->type_num = 1;
                    break;
                }
            case 1:
                {
                    strcpy (cam->follower_type, "Oscillating Flat-Faced");
                    cam->type_num = 2;
                    break;
                }
            case 2:
                {
                    strcpy (cam->follower_type, "Translating Roller");
                    cam->type_num = 3;
                    break;
                }
            case 3:
                {
                    strcpy (cam->follower_type, "Oscillating Roller");
                    cam->type_num = 4;
                    break;
                }
        } /* end of picked switch loop */
    } /* end of if loop */

    /* set the angular rotation direction of the cam */
    if (old_picked2 != picked2)
    { /* start of if loop */
        old_picked2 = picked2;
        switch (picked2)
        { /* start of picked2 switch loop */
            case 0:
                {
                    cam->omega = -1.0;
                }
break;
}
}
}

switch (index)
{
    /* start of index switch loop */

case 7:
{
    strcpy (cam->follower_type, "Translating Flat-Faced");
cam->type_num = 1;
dg_set_radio (db, 1, 0);
picked = 0;
dg_move_focus (db, 1);
break;
}

case 8:
{
    strcpy (cam->follower_type, "Oscillating Flat-Faced");
cam->type_num = 2;
dg_set_radio (db, 1, 1);
picked = 1;
dg_move_focus (db, 1);
break;
}

case 9:
{
    strcpy (cam->follower_type, "Translating Roller");
cam->type_num = 3;
dg_set_radio (db, 1, 2);
picked = 2;
dg_move_focus (db, 1);
break;
}

case 10:
{
    strcpy (cam->follower_type, "Oscillating Roller");
cam->type_num = 4;
dg_set_radio (db, 1, 3);
picked = 3;
dg_move_focus (db, 1);
break;
}

case 11:
{
    /* OK button */
break;
}
case 12:  
{  /* Cancel button */  
cam->omega = default_w;  
cam->type_num = default_follower_type_num;  
strcpy (cam->follower_type, default_follower_type);  
break;  
}  /* end of index switch loop */  
}  /* end of dialog box running loop */
while ((index := 11) && (index != 12));

/* erase dialog box and free memory taken by it */
dg.erase_dialog (db);
dg.free_dialog (db);

/* if not editing cam design */
if (edit_flag == 0) && (index == 11))
{
    motion_segs();
}
}  /* end of cam_params */

*******************************************************************************/
void profile_specs (int type, char name[25], double *p1, double *p2,
    double *p3, double *p4, double *p5)
*******************************************************************************/
/* Purpose: Displays a rendering of the cam profile on the CADKEY screen. */
/* *
/*    Author: Steve Payne             Date of creation: 10-01-93 */
/*    Last modified by: Steve Payne      Date of last modification: 03-03-94 */
*******************************************************************************/
{
    /* start of profile_specs */
    DIALOG_BOX *bbox = NULL;
    int  choice, length, flag1, flag2, flag3, flag4, cancel, old_type;
    char  tempname[25];
    double  len, pos, old1, old2, old3, old4, old5;

    /* initialize dialog box choice and other flags */
    choice = flag1 = flag2 = flag3 = flag4 = cancel = 0;

    /* initialize old_type hub_type variable */
    old_type = cam->hub_type;

    /* initialize these variables which save the initial data in case */
    /* the user uses the cancel option */
    old1 = analysis->x1;
    old2 = analysis->x2;
old3 = analysis->k3;
old4 = analysis->k4;
old5 = analysis->k5;

/* get string length of follower type string */
length = strlen (name);

/* convert int to double value for string length */
len = length;

/* append a ':' to the end of the follower type name */
strncpy (tempname, name);
strcat (tempname, ":");

/* set position to center follower type heading */
pos = (38.0 / 2.0) - ((len + 1) / 2.0) + 1.1;

/* set the cam parameters for drawing the profile */
switch (type)
{
    /* start of switch loop */
    case 1:
    {
        /* start of case #1 (Translating Flat-Faced) */
        /* set value for cancel button in dialeg box */
cancel = 14;

        /* initialize dialog box */
        dbbox = dg_init_dialog (11, 41, 15, "Cam Profile Specifications", 0,
                                0, 4, 0);
        dg_add_note (dbbox, 0, 0.60, pos, tempname, 0, 0, 0);
        dg_add_box (dbbox, 1, 1.40, pos, 0.05, len, NULL, 0, 0, 0);
        dg_add_note (dbbox, 2, 3.00, 2.00, "Base Circle Radius (m)":", 0, 0,
                     0);
        dg_add_note (dbbox, 3, 4.65, 7.00, "Oblique Angle (A)":", 0, 0, 0);
        dg_add_text_double (dbbox, 4, 3.00, 27.00, 8, "%.2f", 0.0, 0, 0,
                            DG_RET_ON_SEL);
        dg_add_text_double (dbbox, 5, 4.65, 27.00, 8, "%.2f", 0.0, 0, 0,
                            DG_RET_ON_SEL);
        dg_add_note (dbbox, 6, 3.00, 36.60, "mm", 0, 0, 0);
        dg_add_note (dbbox, 7, 4.65, 36.60, "g", 0, 0, 0);
        dg_add_box (dbbox, 8, 5.90, 0.50, 0.05, 39.90, NULL, 0, 0, 0);
        dg_add_note (dbbox, 9, 6.70, 2.00, "Cam Hub Type:", 0, 0, 0);
        dg_add_text_int (dbbox, 10, 6.70, 17.45, 2, "#d", cam->hub_type, 0, 0,
                         DG_RET_ON_SEL);
        dg_add_note (dbbox, 11, 6.70, 22.40, "{1 - 2 available}", 0, 0, 0);
        dg_add_box (dbbox, 12, 8.30, 0.60, 0.05, 39.85, NULL, 0, 0, 0);
        dg_add_button (dbbox, 13, 9.15, 7.06, 8, "OK", 0, 0, DG_RET_ON_SEL);
        dg_add_button (dbbox, cancel, 9.15, 26.00, 8, "Cancel", 0, 0,
                       DG_RET_ON_SEL);

        /* set the default values in the text boxes */
        if (edit_flag == 1)


dg_set_text_double (bbox, 4, analysis->k1);
dg_set_text_double (bbox, 5, analysis->k2);

/* set the default oblique angle value to 0.0 */
if (edit_flag == 1)
{
  *p2 = analysis->k2 = 0.0;
}

/* draw the dialog box */
dg_draw_dialog (bbox);

/* start of dialog box running loop */
do {
  /* returns the index of the option selected by the user */
  choice = dg_run_dialog (bbox);

  switch (choice)
  {
  case 4:
  {
    *p1 = analysis->k1 = dg_get_text_double (bbox, 4);
    flag1 = 1;
    break;
  }
  case 5:
  {
    *p2 = analysis->k2 = dg_get_text_double (bbox, 5);
    break;
  }
  case 10:
  {
    old_type = cam->hub_type;
    cam->hub_type = dg_get_text_int (bbox, 10);
    if ((cam->hub_type < 1) || (cam->hub_type > 2))
    {
      printf (prompt,
        "Error in cam hub type...Hit ENTER to continue...");
      ck_pause (prompt);
      cam->hub_type = old_type;
      dg_set_text_int (bbox, 10, cam->hub_type);
      printf (prompt, "");
      ck_prompt (prompt);
      break;
    }
    break;
  }
  case 13:
  {

Appendix A:  Source Code Listing - SPECS.C
if ((flag1 == 0) && (edit_flag == 1))
{
    sprintf(prompt,
        "You haven't set the Base Circle Radius (c)...ENTER to continue");
    ck_pause(prompt);
    sprintf(prompt, "");
    ck_prompt(prompt);
    choice = 7;
}
else
    /* check for positive base circle radius */
if (*p1 < 0.0)
{
    sprintf(prompt,
        "Base circle radius must be positive...ENTER to continue");
    ck_pause(prompt);
    sprintf(prompt, "");
    ck_prompt(prompt);
    dg_set_text_double(&box, 5.0, 0.0);
    flag1 = 0;
    choice = 7;
}
else
{
    break;
}
}
}

/* end of dialog box running loop */
while ((choice != cancel) && (choice != (cancel - 1)))
    break;
/* end of case #1 (Translating Flat-Faced) */
case 2:
    /* start of case #2 (Oscillating Flat-Faced) */
    /* set value for cancel button in dialog box */
    cancel = 17;

/* initialize dialog box */
dbox = dg_init_dialog (14, 41, 18, "Cam Profile Specifications", 0,
    0, 4, 0);
dg_add_note (dbox, 0, 0.60, pos, tempfile, 0, 0, 0);
dg_add_box (dbox, 1, 1.40, pos, 0.05, len, NULL, 0, 0, 0);
dg_add_note (dbox, 2, 3.00, 2.00, "Base Circle Radius (c):", 0, 0,
    0);
dg_add_note (dbox, 3, 4.65, 4.00, "Center Distance (Cx):", 0, 0, 0);
dg_add_note (dbox, 4, 6.30, 4.00, "Center Distance (CY) :", 0, 0, 0);
dg_add_text_double (dbox, 5, 3.00, 27.00, 8, "%2f", 0.0, 0, 0,
DG_RET_ON_SEL);
dg_add_text_double (dbox, 6, 4.65, 27.00, 8, "%2f", 0.0, 0, 0,
DG_RET_ON_SEL);
dg_add_text_double (dbox, 7, 6.30, 27.00, 8, "%2f", 0.6, 0, 0,
DG_RET_ON_SEL);
dg_add_note (dbox, 8, 3.00, 36.60, "mm", 0, 0, 0);
dg_add_note (dbox, 9, 4.65, 36.60, "mm", 0, 0, 0);
dg_add_note (dbox, 10, 6.30, 36.60, "mm", 0, 0, 0);
dg_add_box (dbox, 11, 8.40, 0.50, 0.05, 39.90, NULL, 0, 0, 0);
dg_add_note (dbox, 12, 9.25, 2.00, "Cam Hub Type:", 0, 0, 0);
dg_add_text_int (dbox, 13, 9.25, 17.45, 2, "%d", cam->hub_type, 0, 0,
DG_RET_ON_SEL);
dg_add_note (dbox, 14, 9.25, 22.40, "(1 = 2 available)", 0, 0, 0);
dg_add_box (dbox, 15, 11.00, 0.50, 0.05, 39.90, NULL, 0, 0, 0);
dg_add_button (dbox, 16, 12.00, 7.00, 8, "OK", 0, 0, DG_RET_ON_SEL);
dg_add_button (dbox, cancel, 12.00, 26.00, 8, "Cancel", 0, 0,
DG_RET_ON_SEL);

/* set the default values in the text boxes */
if (edit_flag == 1)
{
    dg_set_text_double (dbox, 5, analysis->k1);
    dg_set_text_double (dbox, 6, analysis->k2);
    dg_set_text_double (dbox, 7, analysis->k3);
}

/* set the default y center distance value (CY) to 0.0 */
if (edit_flag == 1)
{
    *p3 = analysis->k3 = 0.0;
}

/* draw the dialog box */
dg_draw_dialog (dbox);

/* start of dialog box running loop */
do {
    /* returns the index of the option selected by the user */
    choice = dg_run_dialog (dbox);

    switch (choice)
    {
    case 5:
    {
        *p1 = analysis->k1 = dg_get_text_double (dbox, 5);
        flag1 = 1;
        break;
    }
    case 6:
{ 
  *p2 = analysis->x2 = dg_get_text_double (dbox, 6);
  flag2 = 1;
  break;
}

case 7:
{
  *p3 = analysis->x3 = dg_get_text_double (dbox, 7);
  break;
}

case 13:
{
  old_type = cam->hub_type;
  cam->hub_type = dg_get_text_int (dbox, 13);
  if ((cam->hub_type < 1) || (cam->hub_type > 2))
  {
    sprintf (prompt,
     "Error in cam hub type...Hit ENTER to continue...");
    ck_pause (prompt);
    cam->hub_type = old_type;
    dg_set_text_int (dbox, 13, cam->hub_type);
    sprintf (prompt, "
    ck_prompt (prompt);
    break;
  }
  break;
}

case 16:
{
  if ((flag1 == 0) & (edit_flag != 1))
  {
    sprintf (prompt,
     "You haven't set the Base Circle Radius (c) ...ENTER to continue");
    ck_pause (prompt);
    sprintf (prompt, "
    ck_prompt (prompt);
    choice = 8;
  }
  else
  if ((flag2 == 0) & (edit_flag != 1))
    {
      sprintf (prompt,
      "You haven't set the Center Distance (Cx) ...ENTER to continue");
      ck_pause (prompt);
      sprintf (prompt, "
      ck_prompt (prompt);
      choice = 8;
    }
  else
    /* check for positive base circle radius */
    if (*pl < 0.0)
{  
  printf (prompt,  
      "Base circle radius must be positive...ENTER to continue");  
  ck_pause (prompt);  
  printf (prompt, "");  
  ck_prompt (prompt);  
  dg_set_text_double (dbox, 5, 0.0);  
  flag1 = 0;  
  choice = 8;  
}
else  
{  
  break;  
}
}
}  
/* end of dialog box running loop */  
while ((choice != cancel) && (choice != (cancel - 1)))  
{  
  break;  
}  
/* end of case #2 (Oscillating Flat-Faced) */

/* initialize dialog box */

dbox = dg_init_dialog (14, 41, 18, "Cam Profile Specifications",  
           0, 0, 4, 0);  
dg_add_note (dbox, 0, 0.60, pos, tempname, 0, 0, 0);  
dg_add_box (dbox, 1, 1.40, pos, 0.05, len, NULL, 0, 0, 0);  
dg_add_note (dbox, 2, 3.00, 2.00, "Base Circle Radius (c) Join", 0, 0, 0);  

dg_add_note (dbox, 3, 4.65, 5.00, "Offset Distance (d) Join", 0, 0, 0);  
dg_add_note (dbox, 4, 6.30, 7.00, "Roller Radius (r) Join", 0, 0, 0);  
dg_add_text_double (dbox, 5, 3.00, 27.00, 8, "%.2f", 0.0, 0, 0,  
           DG_RET_ON_SEL);  
dg_add_text_double (dbox, 6, 4.65, 27.00, 8, "%.2f", 0.0, 0, 0,  
           DG_RET_ON_SEL);  
dg_add_text_double (dbox, 7, 6.30, 27.00, 8, "%.2f", 0.0, 0, 0,  
           DG_RET_ON_SEL);  
dg_add_note (dbox, 8, 3.00, 36.60, "mm", 0, 0, 0);  
dg_add_note (dbox, 9, 4.65, 36.60, "mm", 0, 0, 0);  
dg_add_note (dbox, 10, 6.30, 36.60, "mm", 0, 0, 0);  
dg_add_box (dbox, 11, 8.40, 0.50, 0.05, 39.90, NULL, 0, 0, 0);  
dg_add_note (dbox, 12, 9.25, 2.00, "Cam Hub Type", 0, 0, 0);  

Appendix A: Source Code Listing - SPECS.C

259
dg_add_text_int (dbox, 13, 9.25, 17.45, 2, 0, 0, 0, 2, 0, "%d", cam->hub_type, 0, 0, DG_RET_ON_SEL);
dg_add_note (dbox, 14, 9.25, 22.40, "%{1 - 2 available} ", 0, 0, 0, 0, DG_RET_ON_SEL);
dg_add_box (dbox, 15, 11.00, 0.60, 0.05, 39.85, NULL, 0, 0, 0, 0, DG_RET_ON_SEL);
dg_add_button (dbox, 16, 12.00, 7.00, 8, "OK", 0, 0, DG_RET_ON_SEL);
dg_add_button (dbox, cancel, 12.00, 26.00, 8, "Cancel", 0, 0, DG_RET_ON_SEL);

/* set the default values in the text boxes */
if (edit_flag == 1)
{
    dg_set_text_double (dbox, 5, analysis->k1);
    dg_set_text_double (dbox, 6, analysis->k2);
    dg_set_text_double (dbox, 7, analysis->k3);
}

/* set the default offset distance to 0.0 */
if (edit_flag == 1)
{
    *p2 = analysis->k2 = 0.0;
}

/* draw the dialog box */
dg_draw_dialog (dbox);

/* start of dialog box running loop */
do{
    /* returns the index of the option selected by the user */
    choice = dg_run_dialog (dbox);

    switch (choice)
    {
    case 5:
    {
        *p1 = analysis->k1 = dg_get_text_double (dbox, 5);
        flag1 = 1;
        break;
    }
    case 6:
    {
        *p2 = analysis->k2 = dg_get_text_double (dbox, 6);
        break;
    }
    case 7:
    {
        *p3 = analysis->k3 = dg_get_text_double (dbox, 7);
        flag2 = 1;
        break;
    }
    case 13:
    {

old_type = cam->hub_type;
cam->hub_type = dg_set_text_int (dbox, 13);
if (!(cam->hub_type < 1) || (cam->hub_type > 2))
{
    sprintf (prompt,
              "Error in cam hub type...Hit ENTER to continue...");
    ck_pause (prompt);
    cam->hub_type = old_type;
    dg_set_text_int (dbox, 13, cam->hub_type);
    printf (prompt, "
              1:"
);  
    ck_prompt (prompt);
    break;
}
break;
}
}
}
case 16:
{
if ((flag1 == 0) && (edit_flag != 1))
{
    sprintf (prompt,
              "You haven't set the Base Circle Radius (c)...ENTER to continue");
    ck_pause (prompt);
    sprintf (prompt, "
              *");
    ck_prompt (prompt);
    choice = 8;
}
else
if ((flag2 == 0) && (edit_flag != 1))
{
    sprintf (prompt,
              "You haven't set the Roller Radius (r)...ENTER to continue");
    ck_pause (prompt);
    sprintf (prompt, "
              *");
    ck_prompt (prompt);
    choice = 8;
}
else
/* check for positive base circle radius */
if (*p1 < 0.0)
{
    sprintf (prompt,
              "Base circle radius must be positive...ENTER to continue");
    ck_pause (prompt);
    sprintf (prompt, "
              *");
    ck_prompt (prompt);
    dg_set_text_double (dbox, 5, 0.0);
    flagi = 0;
    choice = 8;
}
else
/* check for positive roller radius */
if (*p3 < 0.0)
{
    sprintf(prompt,
    "Roller radius must be positive...ENTER to continue");
    ck_pause(prompt);
    sprintf(prompt,"
    ");
    ck_prompt(prompt);
    dg_set_text_double(dbox, 7, 0.0);
    flag2 = 0;
    choice = 8;
}
else
{
    break;
}

}  

/* end of dialog box running loop */
while ((choice != cancel) && (choice != (cancel - 1)))
break;
}  /* end of case #3 (Translating Roller) */

case 4:
{
    /* start of case #4 (Oscillating Roller) */
    /* set value for cancel button in dialog box */
cancel = 23;

    /* initialize dialog box */
    dbox = dg_init_dialog(16, 41, 24, "Cam Profile Specifications", 0, 0, 4, 0);
    dg_add_note(dbox, 0, 0.60, pos. tempname, 0, 0, 0);
    dg_add_box(dbox, 1, 1.40, pos, 0.05, len. NULL, 0, 0, 0);
    dg_add_note(dbox, 2, 3.00, 2.00, "Base Circle Radius (c)":", 0, 0, 0);
    dg_add_note(dbox, 3, 4.60, 7.00, "Roller Radius (r)":", 0, 0, 0);
    dg_add_note(dbox, 4, 6.20, 4.00, "Center Distance (Cx)":", 0, 0, 0);
    dg_add_note(dbox, 5, 7.80, 4.00, "Center Distance (Cy)":", 0, 0, 0);
    dg_add_note(dbox, 6, 9.40, 7.00, "Length Of Arm (l)":", 0, 0, 0);
    dg_set_text_double(dbox, 7, 3.00, 27.00, 8, "%.2f", 0.0, 0, 0, 
    DG_RET_ON_SEL);
    dg_set_text_double(dbox, 8, 4.60, 27.00, 8, "%.2f", 0.0, 0, 0, 
    DG_RET_ON_SEL);
    dg_set_text_double(dbox, 9, 6.20, 27.00, 8, "%.2f", 0.0, 0, 0, 
    DG_RET_ON_SEL);
    dg_set_text_double(dbox, 10, 7.80, 27.00, 8, "%.2f", 0.0, 0, 0, 
    DG_RET_ON_SEL);
/* set the default values in the text boxes */
if (edit_flag == 1)
{
    dg_set_text_double (dbox, 7, analysis->k1);
    dg_set_text_double (dbox, 8, analysis->k2);
    dg_set_text_double (dbox, 9, analysis->k3);
    dg_set_text_double (dbox, 10, analysis->k4);
    dg_set_text_double (dbox, 11, analysis->k5);
}

/* set the default y center distance value (Cy) to 0.0 */
if (edit_flag != 1)
{
    *p4 = analysis->k4 = 0.0;
}

/* draw the dialog box */
dg_draw_dialog (dbox);

/* start of dialog box running loop */
do {
    /* returns the index of the option selected by the user */
    choice = dg_run_dialog (dbox);

    switch (choice)
    {
    case 7:
    {
    *p1 = analysis->k1 = dg_get_text_double (dbox, 7);
    flag1 = 1;
    break;
    }
    case 8:
    {

Appendix A: Source Code Listing - SPECSC.C  263
*p2 = analysis->k2 = dg_get_text_double (dbox, 8);
flag2 = 1;
break;
}

case 9:
{
*p3 = analysis->k3 = dg_get_text_double (dbox, 9);
flag3 = 1;
break;
}

case 10:
{
*p4 = analysis->k4 = dg_get_text_double (dbox, 10);
break;
}

case 11:
{
*p5 = analysis->k5 = dg_get_text_double (dbox, 11);
flag4 = 1;
break;
}

case 19:
{
old_type = cam->hub_type;
cam->hub_type = dg_get_text_int (dbox, 19);
if ((cam->hub_type < 1) || (cam->hub_type > 2))
{
    printf (prompt,
        "Error in cam hub type...Hit ENTER to continue...");
    ck_pause (prompt);
    cam->hub_type = old_type;
    dg_set_text_int (dbox, 19, cam->hub_type);
    sprintf (prompt, "
    ");
    ck_prompt (prompt);
    break;
}
break;
}

case 22:
{
if ((flag1 == 0) && (edit_flag != 1))
{
    printf (prompt,
        "You haven't set the Base Circle Radius (c)...ENTER to continue");
    ck_pause (prompt);
    sprintf (prompt, "
    ");
    ck_prompt (prompt);
    choice = 12;
}
else
if ((flag2 == 0) && (edit_flag != 1))
{ 
sprintf (prompt,  
   "You haven't set the Roller Radius (r)...ENTER to continue");  
ck_pause (prompt);  
sprintf (prompt, "  
ck_prompt (prompt);  
choice = 12;  
}
else
if ((flag3 == 0) && (edit_flag != 1))
{  
sprintf (prompt,  
   "You haven't set the Center Distance (Cx)...ENTER to continue");  
ck_pause (prompt);  
sprintf (prompt, "  
ck_prompt (prompt);  
choice = 12;  
}
else
if ((flag4 == 0) && (edit_flag != 1))
{  
sprintf (prompt,  
   "You haven't set the Length Of Arm (l)...ENTER to continue");  
ck_pause (prompt);  
sprintf (prompt, "  
ck_prompt (prompt);  
choice = 12;  
}
else
/* check for positive base circle radius */
if ("p1 < 0.0")
{  
sprintf (prompt,  
   "Base circle radius must be positive...ENTER to continue");  
ck_pause (prompt);  
sprintf (prompt, "  
ck_prompt (prompt);  
dg_set_text_double (dbox, 5, 0.0);  
flag1 = 0;  
choice = 12;  
}
else
/* check for positive roller radius */
if ("p2 < 0.0")
{  
sprintf (prompt,  
   "Roller radius must be positive...ENTER to continue");  
ck_pause (prompt);  
sprintf (prompt, "  
ck_prompt (prompt);  
dg_set_text_double (dbox, 8, 0.0);  
}
flag2 = 0;
choice = 12;
}
else
    /* check for positive arm length */
if (*p5 < 0.0)
    {
        printf(prompt,
                "Arm length must be positive...ENTER to continue");
        ck_pause (prompt);
        printf(prompt, "");
        ck_prompt (prompt);
        dg_set_text_double (dbox, 11, 0.0);
        flag4 = 0;
        choice = 12;
    }
else
    {
        break;
    }
}
case 23:
    {
        *p5 = 9999;
        break;
    }
}
/* end of dialog box running loop */
while ((choice != cancel) && (choice != (cancel - 1)))
    {
        break;
    }  /* end of case #4 (Oscillating Roller) */
/* end of switch loop */

/* if the cancel button was pressed, reset all variables to their */
/* original values */
if (choice == cancel)
    {
        *p1 = analysis->k1 = old1;
        *p2 = analysis->k2 = old2;
        *p3 = analysis->k3 = old3;
        *p4 = analysis->k4 = old4;
        *p5 = analysis->k5 = old5;
    }

/* erase dialog box and free memory taken by it */
dg_erase_dialog (dbox);
dg_free_dialog (dbox);
}  /* end of profile_specs */
Appendix A: Source Code Listing

File: UTILS.C

/********************************************
/* Cam Design System - Miscellaneous Utility Functions */
********************************************************************/
/* File: UTILS.C */
********************************************************************/
/* Programmed by: */
/* */
/* Virginia Tech Robotics & Mechanisms Group */
/* C/o Dr. Charles F. Reinholdz */
/* 106 Randolph Hall */
/* Department of Mechanical Engineering */
/* Virginia Tech */
/* Blacksburg, VA 24061-0238 */
********************************************************************/
/* Purpose: A collection of miscellaneous utility functions for */
/* manipulating structures, starting the Cam Design System in */
/* the CADKEY Applications menu, etc. */
/* */
/* Author: Steve Payne Date of creation: 07-06-93 */
/* Last modified by: Steve Payne Date of last modification: 03-03-94 */
********************************************************************/

/* necessary include files */
#include <stdio.h>
#include <stdlib.h>
#include "ck_cdl.h" /* for standard CADL functions */
#include "ck_dig.h" /* for dialog boxes */
#include "ck_sys.h" /* for system commands */
#include "global.h" /* Cam Design System global variable header file */
#include "cam.h" /* Cam Design System main header file */

/*******************************
void rewind_profile (struct profile **dummy)
*******************************
/* Purpose: Rewinds the prof doubly-linked-list pointer to the beginning. */
/* */
/* Author: Steve Payne Date of creation: 07-09-93 */
/* Last modified by: Steve Payne Date of last modification: 01-24-94 */
void forward_it (struct profile **dummy, int num)  

/* Purpose: Advances the doubly-linked-list pointer to the NUMth position. */  
/* */  
/* Author: Steve Payne Date of creation: 07-11-93 */  
/* Last modified by: Steve Payne Date of last modification: 10-06-93 */  
/***************************************************************************/  
{  
    int d;  
    
    for (d = 1; d <= (num - 1); d++)  
    {  
        (*dummy) = (*dummy)->next;  
    }  
}  
  
/***************************************************************************/  

void rewind_curve (struct svaj **dummy)  

/***************************************************************************/  
/* Purpose: Rewinds the svaj doubly-linked-list pointer to the beginning. */  
/* */  
/* Author: Steve Payne Date of creation: 10-01-93 */  
/* Last modified by: Steve Payne Date of last modification: 01-24-94 */  
/***************************************************************************/  
{  
    while ((*dummy)->prev_one != NULL)  
    {  
        (*dummy) = (*dummy)->prev_one;  
    }  
}  
  
/***************************************************************************/  

void rewind_pressure (struct pressure_analysis **dummy)  

/***************************************************************************/  
/* Purpose: Rewinds the pressure doubly-linked-list pointer to the */
void rewind_rc (struct nc_data **dummy)

/* Purpose: Rewinds the nc doubly-linked-list pointer to the beginning. */
/* Author: Steve Payne Date of creation: 02-12-94 */
/* Last modified by: Steve Payne Date of last modification: 02-12-94 */

void install_it (void)

/* Purpose: Contains the commands to set up the Cam Design System CDE file in the CADKEY Applications Menu. */
/* Author: Steve Payne Date of creation: 07-29-93 */
/* Last modified by: Steve Payne Date of last modification: 11-17-93 */

void erase_screen (int the_viewport)

/* Purpose: Erases the current CADKEY viewport screen (and database) */
/* through the use of changing levels. */
/ * Author: Steve Payne Date of creation: 11-17-93 */
/* Last modified by: Steve Payne Date of last modification: 11-17-93 */
*******************************************************************************/
{
  /* start of erase_screen */
  /* set get mode to get ALL entities displayed in the prime */
  /* (current) viewport */
  ck_getall (the_viewport);

  /* now delete this whole selected list from the CADKEY database */
  /* this and the next command serve to erase what is already in the */
  /* CADKEY database and on the CADKEY screen */
  ck_delent (CK_SEL_LIST);

  /* erase screen {before program starts; */
  ck_cls (CK_ALL_VP);
} /* end of erase_screen */

*******************************************************************************/
void user_warning (void)
*******************************************************************************/
/* Purpose: Gives the user a warning when he/she attempts to manipulate a */
/* cam profile when one does not yet exist. */
/* */
/* Author: Steve Payne Date of creation: 02-24-94 */
/* Last modified by: Steve Payne Date of last modification: 02-24-94 */
*******************************************************************************/
{
  /* start of user_warning */
  DIALOG_BOX *db = NULL;
  int choice;

  /* initialize dialog box choice */
  choice = 0;

  /* make new dialog box to indicate that there is no cam data */
  db = dg_init_dialog (6, 25, 2, "Error", DG_WHITE, DG_RED, 4, DG_CANCOLOR);
  dg_add_note (db, 0, 1.50, 2.0, "No cam profile exists", DG_WHITE, DG_RED,
               DG_CANCOLOR);
  dg_add_button (db, 1, 4.00, 8.25, 8, "OK", DG_BLUE, DG_WHITE,
                 DG_RET_ON_SEL);

  /* draw the dialog box */
  dg_draw_dialog (db);

do { /* start of do-while loop */
  /* run the dialog box */
  choice = dg_run_dialog (db);
} /* end of do-while loop */
while (choice != 1);

  /* erase and free memory from dialog box */
  dg_erase_dialog (db);
  dg_free_dialog (db);
  }            /* end of user_warning */

void initialize_icons (DIALOG_BOX *d)

/**********************************************/
/* Purpose: Initializes all icons coordinates and definitions for use in    */
/* the cam parameters dialog box.                                          */
/*                                                                            */
/* Author: Steve Payne          Date of creation: 12-14-93        */
/* Last modified by: Steve Payne    Date of last modification: 01-23-94 */
/***********************************************/
{
  void         *icon;
  double     x[8], y[8];

  /* Translating Flat-Faced Follower */
  /* start the icon definition */
  icon = dg_icon_def_start (*"tranflat", *"icons", 8, 8.0, 4.0);

  /* define all components of this icon */
  x[0] = 0.0;
  y[0] = 0.0;
  x[1] = 8.0;
  y[1] = 4.0;
  dg_icon_def_part (icon, 0, DG_ICON_BAR, DG_WHITE, 2, x, y);
  x[0] = 1.7;
  y[0] = 1.6;
  x[1] = 4.9;
  y[1] = 2.53;
  x[2] = 4.6;
  y[2] = 2.73;
  x[3] = 1.4;
  y[3] = 1.8;
  dg_icon_def_part (icon, 1, DG_ICON_POLY, DG_RED, 4, x, y);
  x[0] = 0.0;
  y[0] = 1.5;
  x[1] = 2.0;
  y[1] = 2.0;
  dg_icon_def_part (icon, 2, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 2.0;
  y[0] = 2.0;
  x[1] = 3.5;
  y[1] = 2.5;
dg_icon_def_part (icon, 3, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 3.5;
y[0] = 2.5;
x[1] = 5.0;
y[1] = 3.0;
dg_icon_def_part (icon, 4, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 5.0;
y[0] = 3.0;
x[1] = 6.5;
y[1] = 4.0;
dg_icon_def_part (icon, 5, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 3.23;
y[0] = 2.0;
x[1] = 5.3;
y[1] = 0.2;
dg_icon_def_part (icon, 6, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 3.76;
y[0] = 2.12;
x[1] = 6.0;
y[1] = 0.2;
dg_icon_def_part (icon, 7, DG_ICON_LINE, DG_BLACK, 2, x, y);
dg_add_icon (d, 7, 2.2, 2.00, CK_FALSE, "tranflat", "icons", 0);

/* Oscillating Flat-Faced Follower */
/* start the icon definition */
icon = dg_icon_def_start ("osciflat", "icons", 9, 8.0, 4.0);

/* define all components of this icon */
  x[0] = 0.0;
y[0] = 0.0;
x[1] = 8.0;
y[1] = 4.0;
dg_icon_def_part (icon, 0, DG_ICON_BAR, DG_WHITE, 2, x, y);
  x[0] = 1.7;
y[0] = 1.6;
x[1] = 4.9;
y[1] = 2.53;
x[2] = 4.5;
y[2] = 2.73;
x[3] = 1.4;
y[3] = 1.8;
dg_icon_def_part (icon, 1, DG_ICON_POLY, DG_RED, 4, x, y);
  x[0] = 0.0;
y[0] = 1.5;
x[1] = 2.0;
y[1] = 2.0;
dg_icon_def_part (icon, 2, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 2.0;
y[0] = 2.0;
x[1] = 3.5;
y[1] = 2.5;

dg_icon_def_part (icon, 3, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 3.5;
y[0] = 2.5;
x[1] = 5.0;
y[1] = 3.0;
dg_icon_def_part (icon, 4, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 5.0;
y[0] = 3.0;
x[1] = 6.5;
y[1] = 4.0;
dg_icon_def_part (icon, 5, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 4.95;
y[0] = 2.5;
x[1] = 6.7;
y[1] = 3.0;
dg_icon_def_part (icon, 6, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 4.65;
y[0] = 2.75;
x[1] = 6.6;
y[1] = 3.25;
dg_icon_def_part (icon, 7, DG_ICON_LINE, DG_BLACK, 2, x, y);
  x[0] = 6.9;
y[0] = 2.95;
x[1] = 7.2368;
y[1] = 3.0432;
x[2] = 7.35;
y[2] = 3.2;
x[3] = 7.2368;
y[3] = 3.3568;
x[4] = 6.9;
y[4] = 3.45;
x[5] = 6.5732;
y[5] = 3.3568;
x[6] = 6.45;
y[6] = 3.2;
x[7] = 6.5732;
y[7] = 3.0432;
dg_icon_def_part (icon, 8, DG_ICON_POLY, DG_BLACK, 8, x, y);
dg_add_icon (d, 8, 7.1, 2.00, CK_FALSE, "osciflat", "icons", 0);

/* Translating Roller Follower */
/* start the icon definition */
icon = dg_icon_def_start ("tranroll", "icons", 8, 8.0, 4.0);

/* define all components of this icon */
x[0] = 0.0;
y[0] = 0.0;
x[1] = 8.0;
y[1] = 4.0;
dg_icon_def_part (icon, 0, DG_ICON_BAR, DG_WHITE, 2, x, y);
  x[0] = 5.3;

Appendix A: Source Code Listing - UTILS.C
y[0] = 1.7;  
\( x(1) = 6.1250; \)
\( y(1) = 1.9250; \)
\( x(2) = 6.5; \)
\( y(2) = 2.4; \)
\( x(3) = 6.1250; \)
\( y(3) = 2.8750; \)
\( x(4) = 5.3; \)
\( y(4) = 3.1; \)
\( x(5) = 4.4750; \)
\( y(5) = 2.8750; \)
\( x(6) = 4.1; \)
\( y(6) = 2.4; \)
\( x(7) = 4.4750; \)
\( y(7) = 1.9250; \)

dg_icon_def_part (icon, 1, DG_ICON_POLY, DG_RED, 8, x, y);
\( x[0] = 0.0; \)
\( y[0] = 1.5; \)
\( x[1] = 2.0; \)
\( y[1] = 2.0; \)

dg_icon_def_part (icon, 2, DG_ICON_LINE, DG_BLACK, 2, x, y);
\( x[0] = 2.0; \)
\( y[0] = 2.0; \)
\( x[1] = 3.5; \)
\( y[1] = 2.5; \)

dg_icon_def_part (icon, 3, DG_ICON_LINE, DG_BLACK, 2, x, y);
\( x[0] = 3.5; \)
\( y[0] = 2.5; \)
\( x[1] = 4.9; \)
\( y[1] = 3.05; \)

dg_icon_def_part (icon, 4, DG_ICON_LINE, DG_BLACK, 2, x, y);
\( x[0] = 4.9; \)
\( y[0] = 3.05; \)
\( x[1] = 6.5; \)
\( y[1] = 4.0; \)

dg_icon_def_part (icon, 5, DG_ICON_LINE, DG_BLACK, 2, x, y);
\( x[0] = 5.27; \)
\( y[0] = 1.65; \)
\( x[1] = 5.8; \)
\( y[1] = 0.1; \)

dg_icon_def_part (icon, 6, DG_ICON_LINE, DG_BLACK, 2, x, y);
\( x[0] = 5.76; \)
\( y[0] = 1.77; \)
\( x[1] = 5.35; \)
\( y[1] = 0.1; \)

dg_icon_def_part (icon, 7, DG_ICON_LINE, DG_BLACK, 2, x, y);
dg_add_icon (d, 9, 2.2, 30.75, CK_FALSE, "tranroll", "icons", 0);

/* Oscillating Roller Follower */
/* start the icon definition */
icon = dg_icon_def_start ("osciroll", "icons", 9, 8.0, 4.0);
/* define all components of this icon */
x[0] = 0.0;
y[0] = 0.0;
x[1] = 3.0;
y[1] = 4.0;
dg_icon_def_part (icon, 0, DG_ICON_BAR, DG_WHITE, 2, x, y);
x[0] = 2.3;
y[0] = 0.7;
x[1] = 3.1250;
y[1] = 0.9250;
x[2] = 3.5;
y[2] = 1.4;
x[3] = 3.1250;
y[3] = 1.8750;
x[4] = 2.3;
y[4] = 2.1;
x[5] = 1.4750;
y[5] = 1.8750;
x[6] = 1.1;
y[6] = 1.4;
x[7] = 1.4750;
y[7] = 0.9250;
dg_icon_def_part (icon, 1, DG_ICON_POLY, DG_RED, 8, x, y);
x[0] = 0.0;
y[0] = 1.5;
x[1] = 2.0;
y[1] = 2.05;
dg_icon_def_part (icon, 2, DG_ICON_LINE, DG_BLACK, 2, x, y);
x[0] = 2.0;
y[0] = 2.05;
x[1] = 3.5;
y[1] = 2.5;
dg_icon_def_part (icon, 3, DG_ICON_LINE, DG_BLACK, 2, x, y);
x[0] = 3.5;
y[0] = 2.5;
x[1] = 5.0;
y[1] = 3.0;
dg_icon_def_part (icon, 4, DG_ICON_LINE, DG_BLACK, 2, x, y);
x[0] = 5.0;
y[0] = 3.0;
x[1] = 6.5;
y[1] = 4.0;
dg_icon_def_part (icon, 5, DG_ICON_LINE, DG_BLACK, 2, x, y);
x[0] = 3.5;
y[0] = 1.3;
x[1] = 6.3;
y[1] = 1.8;
dg_icon_def_part (icon, 6, DG_ICON_LINE, DG_BLACK, 2, x, y);
x[0] = 3.39;
y[0] = 1.56;
x[1] = 6.2;
y[1] = 2.05;
dg_icon_def_part (icon, 7, DG_ICON_LINE, DG_BLACK, 2, x, y);
x[2] = 6.4;
y[2] = 1.75;
x[3] = 6.7268;
y[3] = 1.8432;
x[4] = 6.85;
y[4] = 2.0;
x[5] = 6.7268;
y[5] = 2.1568;
x[6] = 6.4;
y[6] = 2.25;
x[7] = 6.0732;
y[7] = 2.1568;
x[8] = 5.95;
y[8] = 2.0;
x[9] = 6.0732;
y[9] = 1.8432;
dg_icon_def_part (icon, 8, DG_ICON_POLY, DG_BLACK, 8, x, y);
dg_add_icon (d, 10, 7.1, 30.75, CK_FALSE, "osciroll", "icons", 0);
} /* end of initialize_icons */
Vita

Steve Payne was born October 10, 1969 in Bethesda, Maryland. After spending all his youth in Springfield, Virginia, he attended Virginia Tech on the '5-year plan,' receiving Bachelor's degrees in Mechanical Engineering and Economics, as well as a minor in Mathematics in May 1992. After mulling over life in general in Europe for 2 months, he returned to Virginia Tech, where he finished his Master of Science degree in Mechanical Engineering in February 1994. After graduation he first plans to take several months off to once again roam places far and near. He will then begin employment in the summer with Cummins Engine Company, Inc. in Columbus, Indiana as a Mechanical Engineer in their Injector & Valve Train Systems Group.

[Signature]

Stephen Payne