Acknowledgments

I would like to thank Dr. Dennis Irwin for the opportunity to work on this project. It is my hope that this software package will be maintained, enhanced with new features, and used for educational purposes, for example a laboratory involving control theory such as Flexlab.

I would like to express my appreciation to the Stocker Fellowship program and the Ohio University School of Electrical Engineering and Computer Science for providing the means to attend graduate school. The teaching and research positions gave me the opportunity to share my knowledge with students and to document the controls laboratories. It also gave me the opportunity to work with Charles Alexander and Matthew Sadiku in the production of the solutions manual and the supplement, Problem Solving Made Almost Easy, to the text Fundamentals of Electric Circuits.

Finally, I would like to thank my parents, Daniel and Barbara Huitger. They are truly an integral part of this accomplishment. Without their prayers, continued support, and unconditional love, I could not have persevered.
# Table of Contents

Acknowledgments........................................................................................................ iii

Table of Contents........................................................................................................ iv

List of Figures............................................................................................................... vii

Chapter 1  Introduction ................................................................................................. 1

1.1 Goals ....................................................................................................................... 2

1.2 Available Software Products.................................................................................. 4

1.2.1 Disadvantages of Matlab Compared to Java..................................................... 5

1.2.2 Advantages of Java .......................................................................................... 6

1.2.3 Existing Java Products .................................................................................... 8

1.3 Organization .......................................................................................................... 10

Chapter 2  Graphical Tools for Control System Design............................................. 11

2.1 Overview of the Software Approach .................................................................... 12

2.2 Fundamentals of Object-Oriented Programming ............................................... 13

2.3 Satisfying the Software Design Specifications Using Java ............................... 14

2.3.1 Consistency ................................................................................................... 15

2.3.2 Encapsulation ................................................................................................ 15

2.3.3 Interactive Nature ......................................................................................... 16
2.3.4 Extensibility .......................................................... 16
2.3.5 Reusability ............................................................ 18
2.3.6 Portability ............................................................. 18
2.3.7 Hardcopy Capability .................................................... 19
2.3.8 Platform Independence ............................................... 19
2.4 Features Included in the Package ....................................... 20
   2.4.1 Appearance and Usability of the Package ...................... 21
   2.4.2 Selecting the Desired Options and Displaying the Data ........ 22
   2.4.3 Acceptable Data File Formats .................................... 26
   2.4.4 Modification of the Figure ...................................... 28
   2.4.5 Sample Figures Utilizing Various Plot Types and Features ........ 36

Chapter 3 A New Graphical Technique for Control System Analysis and Design .... 42
3.1 Control System Design Process ........................................ 43
3.2 Design Curves ................................................................ 44
   3.2.1 Problem Statement .................................................. 44
   3.2.2 Relative Stability .................................................... 45
   3.2.3 Relationship Between the Time Domain and Frequency Domain .... 46
3.3 Digital PID Controller Design ............................................ 51
3.4 Example: Digital Compensator Design ............................... 54
   3.4.1 Background .......................................................... 55
   3.4.2 Initial Design of a PID Compensator ............................ 56
   3.4.3 Final Design of a PID Compensator ............................. 62
List of Figures

Figure 2.1: Plot Window – User Interface to Make Desired Selections ......................... 22
Figure 2.2: Open File Dialog – Allows the User to Browse for a File ......................... 23
Figure 2.3: Text Editor – View or Create a Data File ............................................. 23
Figure 2.4: Plot Window – Select the Desired Type of Plot to Be Displayed ............. 24
Figure 2.5: Notification Dialog – Displays a Message for the User ......................... 25
Figure 2.6: Menu to Resize, Move, or Close the Plot Window .............................. 25
Figure 2.7: Figure Window – Window Menu ...................................................... 29
Figure 2.8: Figure Window – Figure Menu ...................................................... 29
Figure 2.9: Figure Window – Customize Menu .................................................. 31
Figure 2.10: Annotate TextBox – Get User Input to Label the Figure .................... 32
Figure 2.11: Manual TextBox – Get User Input to Manually Scale the Axes .......... 32
Figure 2.12: SubGraph TextBox – Select the Graph on which to Perform the Option ... 33
Figure 2.13: Figure Window – PolarPlot Menu ............................................... 34
Figure 2.14: Magnitude TextBox – Get User Input to Add
              a Constant Magnitude Circle ......................................................... 35
Figure 2.15: Phase TextBox – Get User Input to Add a Constant Phase Line .......... 35
Figure 2.16: M-Contour TextBox – Get User Input to Add M-Contours .................. 36
Figure 2.17: Example – X-Y Plot with a Data Point Displayed ........................................ 38
Figure 2.18: Example – Semilogx Plot Using the Subplot Option .................................. 39
Figure 2.19: Example – Polar Plot with All PolarPlot Options and a Data Point ............ 41
Figure 3.1: Closed-Loop Control System Block Diagram ............................................ 44
Figure 3.2: Polar Plot for $GH(j\omega)$ ....................................................................... 46
Figure 3.3: Phase Margin vs. Percent Overshoot for Second-Order Systems ............... 49
Figure 3.4: Crossover Frequency vs. Percent Overshoot as a Function of Settling Time .................................................................................................................. 50
Figure 3.5: Crossover Frequency vs. Percent Overshoot as a Function of Settling Time .................................................................................................................. 50
Figure 3.6: Servo System with a Digital Compensator .................................................. 55
Figure 3.7: Servo System in Rate Control Configuration with Forward Path Attenuation .................................................................................................................. 55
Figure 3.8: Simulated Open-Loop Step Response ....................................................... 56
Figure 3.9: Open-Loop Uncompensated Frequency Response .................................... 58
Figure 3.10: Compensator Frequency Response ......................................................... 59
Figure 3.11: Open-Loop Compensated Frequency Response ...................................... 59
Figure 3.12: Closed-Loop Compensated Frequency Response .................................... 60
Figure 3.13: Simulated Closed-Loop Step Response ................................................... 61
Figure 3.14: Open-Loop Uncompensated Frequency Response .................................... 63
Figure 3.15: Compensator Frequency Response ......................................................... 63
Figure 3.16: Open-Loop Compensated Frequency Response ...................................... 64
Figure 3.17: Closed-Loop Compensated Frequency Response .................................... 65
Figure 3.18: Simulated Closed-Loop Step Response ................................................... 65
Chapter 1

Introduction

The history of control theory has continued to change with new theories and technologies. There was a time when control engineering was primarily motivated by frequency-domain methods in the United States and Western Europe and by time-domain methods in the Soviet Union and Eastern Europe. With the introduction of computers, numerical calculations can be performed more quickly and the data can be plotted for analysis. With the beginning of control system applications in space, there was a necessity for more complex and highly accurate control systems. The field of optimal control arose from the necessity to minimize the weight of satellites and to control them accurately. Now, frequency-domain methods and time-domain methods are utilized simultaneously in the analysis and design of control systems [1].

This thesis will present a situation where the design specifications are given in the time domain. Assuming a second-order system, the time-domain specifications will be translated into design specifications in the frequency domain. After finding an adequate model of the system, the transfer function of the controller can be determined. A simulation of the closed-loop compensated system will determine if the design
specifications have been met. If so, the design is complete. If not, more analysis and re-design must be done.

In the design process, it is necessary to examine the data graphically. There are many software products available to calculate data or plot data. Yet, in control theory, there is a desire to look at the frequency response of the system via a polar plot. To find certain points of interest of the frequency response, such as the phase margin and the crossover frequency, the polar plot should include the capability of displaying a data point, a constant magnitude circle, a constant phase line, and constant closed-loop magnitude contours. There is not a software product that has all of these features readily available. So, this thesis will present a modern solution to this predicament.

1.1 Goals

One goal to be accomplished is the construction of the foundation of an analysis tool, especially for control systems. As the foundation of a larger application, this program gives the user the opportunity:

➢ To view or create a data file using a text editor,

➢ To load a data file in ASCII format,

➢ To plot the data using one of a variety of plot types including:
  x-y, semilogx, semilogy, loglog, or polar,

➢ To display the data as a single plot or two plots using the 2x1 subplot option,
➢ To control the display of the plot by
  • adding grid lines,
  • adding annotation,
  • selecting and displaying a data point,
  • scaling the axes, and

➢ To print a hardcopy.

All of these features are incorporated into a user-friendly graphical user interface (GUI), so that programming experience is not required. With this application, the users can concentrate on the analysis of the system rather than the programming issues.

In control systems analysis and design, a system must have an acceptable transient response. Furthermore, there is an obvious concern for system stability. Because the model used to represent the system is not exact, the simulated results may indicate a stable system, when the actual system is unstable. Thus, it is desirable to design a system that is stable within a margin of safety.

For analysis using a frequency response, two measures of relative stability of a system are gain margin and phase margin; both terms are defined in Section 3.2.2. Assuming a second-order system, the gain margin and phase margin can be used to determine the relationships between time-domain characteristics and frequency-domain characteristics of a system. From these relationships, two sets of design curves can be generated to aid in controller design.
Next, a method for designing a digital Proportional-Integral-Derivative (PID) controller with an emphasis on frequency response techniques is presented [2]. The design equations are developed in the \( \omega \)-plane, where the \( \omega \)-plane is a mapping of the \( z \)-plane via the bilinear transformation. Using relationships between the \( \omega \)-plane controller coefficients and \( z \)-plane controller coefficients, the resulting equations will allow the design of controllers directly in the \( z \)-plane.

In the end, the accomplishment of the goals is verified via an example. The example makes use of the design curves and the digital PID controller design. In addition, the plots for the example are constructed using the graphical application.

### 1.2 Available Software Products

Some products are used for plotting experimental data; others can be used to simulate and plot the response of a given system model. For example, a spreadsheet, like Excel or Lotus 1-2-3, can be used to enter and plot data. Matlab\(^*\) is a numeric computation and visualization software package used by engineers. The basic package can load and plot data. The required functions, relating to specific fields of engineering, used to simulate the response of a system are combined into toolboxes, e.g. the control toolbox and signal processing toolbox. Control system analysis and design can be performed using other specialized programs, such as Program CC, Easy 5, Matrix X, and Control System Program (CSP), in addition to Matlab. Because there is not an existing software product that meets the goal, a new software product is developed using Java\(^*\).
1.2.1 Disadvantages of Matlab Compared to Java

Matlab is a close match to the desired goals for this package. On the other hand, there are disadvantages to the use of Matlab. An important concern for some is cost. There is the cost to purchase Matlab and the required user license.

Then, there are issues concerning the available features of Matlab. Although Matlab does possess most of the software features listed as the goal of the software product, Matlab does not display polar plots as requested. The polar plots are not scaled to display negative decibel magnitudes, with the minimum magnitude at the center of the graph. Thus, the relative stability, including gain margin and phase margin, is not evident by observation of the graph.

In addition, the user cannot easily interact with the graph. For example, the user cannot select a data point to be displayed. The automatic scaling of the axes does not always maximize the area of the graph for the data; there are typically unnecessary blank areas on the sides of the graph. Scaling the axes to view only a portion of the data is not a simple task, without updating to Matlab version 5. These interactive features are positive qualities in any software product used for control system analysis and design.

Currently, Matlab is widely used by control engineers. However, the Java Development Kit (JDK), used to write and maintain code, and the Java Runtime Environment (JRE), used strictly to run compiled Java code, can be downloaded from Sun’s web site at <http://java.sun.com> [3] free of charge. Hence, the cost of using a product developed using Java is dependent on the creator of the product. The intention is for this package to be freeware, available to users for no cost, used for educational
purposes. Developing a Java application provides the opportunity to tailor the appearance and usability to particular needs, specifically those features that are not included in Matlab. The polar plot can be customized to contain options for easier analysis of the data. In addition, the user can interact with the graph to view a portion of the graph or the value of a particular data point.

1.2.2 Advantages of Java

Java can be used to meet the following design specifications, discussed further in Section 2.3:

- Consistency – Various types of plots should perform tasks, such as the selection of a data point and manual scaling, in the same manner and be easy to use.
- Encapsulation – The plots should be able to handle the given tasks, or implement the features of the figure, with respect to the window manager.
- Interactive nature – The package should give the user the opportunity to interact with the graph while examining the data.
- Extensibility – A wide variety of plots should be available, with a focus on plots of a linear system/control nature.
- Reusability – The documentation and software approach should facilitate easy modification and addition.
➢ Portability – The package should be designed for importation into productivity software.

➢ Hardcopy capability.

➢ Platform independence.

These design specifications are best accomplished using an object-oriented programming (OOP) language, which is discussed in more detail in Chapter 2. Presently, the two popular OOP languages, widely used in industry, are C++ and Java. The key advantage of Java is in the ease of development, attributed to Java characteristics such as platform independence, the Java Application Programming Interface (API), and memory management and garbage collection [4, 5, 6, 7].

With platform independence, software can be developed on one platform and run on multiple platforms without recompiling the code or maintaining multiple versions of the code. This is possible due to the interpreted nature of Java. The code is compiled into bytecodes [5], or a sequence of bytes that represent instructions to the Java Virtual Machine (JVM). The JVM translates these instructions into machine code, which is understood by the specific platform of the computer.

There is an ease in development when given a class library, or a set of classes, from which to build a program. Note that a class is a template which contains variables and functions representing its structure, behavior, and attributes. Designed to aid in software development, the Java API is a set of predefined packages that may contain several classes and interfaces. Note that an interface is a collection of abstract behavior
specifications that individual classes can implement in order to add the behaviors of the interface to its current set of behaviors. Some of the packages include:

- java.lang – core classes that provide data types, string manipulation, built-in mathematical functions, threads, the means to handle errors, and methods that provide a degree of control over the runtime environment.

- java.io – the standard input/output library.

- java.awt – the Java Abstract Window Toolkit (AWT) consists of classes to create graphical user interfaces in applets and applications.

- java.awt.event – methods to handle events or actions performed by a user while running a program.

In Java, the programmer does not have to remember to allocate or free memory. Every instance of a class is assigned with the new operator, allocating space for the instance in memory. The memory becomes available for reuse when there are no longer any references to the instance. Reclaiming the allocated memory is known as garbage collection, which runs when the system is idle or when the allocation of memory fails.

1.2.3 Existing Java Products

Java products with the purpose of plotting data are available. There are many graphing products that create line charts, bar charts, pie charts, and more. For example, KavaChart, located at <http://www.ve.com/kavachart/> [8], supports almost any kind of
business, financial, or scientific chart. However, these products do not include logarithmic plots or polar plots that are necessary for control system analysis. They are more applicable to business than engineering.

Other products plot predefined functions. For example, XYGraph, located at <http://home.sprynet.com/sprynet/gbildson/ngraph/xygraph.html> [9], is limited to graphing functions on a linear x-y plot. These products do not allow the user to load a data file. In control system design, an acceptable mathematical model is found through the analysis of data obtained from the system.

Leigh Brookshaw created a Java package. Information concerning the graph package can be found at <http://www.sci.usq.edu.au/staff/leighb/graph> [10]. This package is the closest Java product containing the desired features, stated in Section 1.1. The package is able to load data and apply the use of the mouse to control the scaling of the axes. Some drawbacks include the need for programming knowledge, the limitation to x-y plots, and the exclusion of printing capabilities. Also, the graph package is outdated; it was created using Sun's Java Development Kit (JDK) version 1.0, where the current version is 1.3. Many changes have been made to the JDK; some items will not be supported in future versions of the JDK and other items are additions to enhance current or updated Java code.

The graph package includes some of the desired features. After updating and modifying the graph package, it can be used as the base of a larger package. Expansion of this package would first include the addition of other plot types and the option of subplots. Additional functionality needs to be added to the plots, including polar plot
options and printing capability. To eliminate a requirement of programming knowledge, a user-friendly GUI needs to be created. The new product to be presented in this thesis is the *graphtools* package.

1.3 Organization

Chapter 1 introduces the thesis by explaining the motivation to create the *graphtools* package using Java, to derive the relationships between time-domain and frequency-domain control criteria to form the design curves to aid in controller design, and to present the equations to design a digital PID controller. The introduction includes the goals and specifications for the *graphtools* package. Chapter 2 examines the software approach, the software design specifications, and the features of the package. Chapter 3 examines the relationship between the time domain and the frequency domain for second-order systems, with the aim of deriving design curves to aid in controller design. This chapter also investigates the design of a PID compensator for the Modular Servo System, a system used in the senior level control laboratory at Ohio University. Chapter 4 concludes with recommendations for further work. Finally, the appendix contains general instructions to maintain the Java package, including instructions for installation of the JDK and the *graphtools* package. In addition, instructions for compiling and executing the Java application are provided.
Chapter 2

Graphical Tools for Control System Design

The objective of this chapter is to examine the Java application taking into account the software specifications and features of the package. The chapter begins with an overview of the software approach. The chapter continues with a brief explanation of the fundamentals of object-oriented programming. These sections provide background information to become familiar with the main classes of the package and the terminology used when describing object-oriented programming. At this point, the software design specifications can be discussed, citing details of the package and terminology of object-oriented programming.

With the achievement of the software specifications, the chapter concludes with an inspection of the features included in the package. This section begins with the appearance and usability of the package. It continues with an investigation into the available options to display data, a report on acceptable data file formats, and a description of the various options to modify or customize a figure. The section finishes with a few examples giving step-by-step instructions to create figures using various plot types and features.
2.1 Overview of the Software Approach

There are three main classes at the core of the `graphtools` package. `Axis` controls the appearance of the axes, `DataSet` is designed to hold the data to be plotted, and `Graph2D`, the canvas, is the main plotting class. These three classes are registered with one another to permit `Graph2D` to control the automation of the drawing operations on the graph. There are also supporting classes to manipulate the drawing on the `Graph2D` canvas, such as rotated text to label the y-axis.

The user interface is comprised of four key classes and supporting classes. The four classes are `PlotWin`, `DisplayPlot`, `Figure`, and `G2Dmenu`. `PlotWin` creates the GUI containing the controls to create or view data and to select a data file and plot options. It also has the ability to initiate a call to `DisplayPlot` which loads the data and uses the characteristics from the `Plot` window to create the proper instances of `Axis` and `DataSet` for the graph. Furthermore, a new instance of `Figure`, a window in which to display the graph, is created to hold the new instance of `G2Dmenu`.

The three classes, `Graph2D`, `G2Dint`, and `G2Dmenu`, act as a canvas and are an example of inheritance which is discussed further in Section 2.2. `G2Dmenu` is a subclass of `G2Dint`, which is a subclass of `Graph2D`. `G2Dint` adds interactive features to `Graph2D`, such as typing 'R' to reset the current graph. `G2Dmenu` is to be used in conjunction with `Figure` and the supporting classes. `G2Dmenu` adds interaction with the user via menus in the `Figure` window. The supporting classes enhance the package with the ability to notify the user of errors, the creation of dialog boxes prompting the user for information, and a text editor to view or create a data file.
2.2 Fundamentals of Object-Oriented Programming

The software design takes advantage of the many attributes of Java, an object-oriented programming language. Object-oriented programming is a type of programming in which the program is viewed in its entirety. The approach focuses on the development of self-contained software components, or objects. An object is defined in terms of its data and the functionality which uses and manipulates this data. Some powerful features of object-oriented languages include inheritance, polymorphism, and encapsulation [5, 6, 7].

Recall that a class is a template which contains variables and functions representing its structure, behavior, and attributes. Inheritance enables a class to be built from an existing class. This newly created class is a subclass of the existing class, and the existing class is a superclass of the new class. The new class inherits the data and methods of its superclass. It is then possible to supply only additional data and methods that are unique to the subclass. The new class expands on the basic class in order to create a more complex class which is tailored to meet a particular need. Inheritance is a powerful feature encouraging software reuse.

Polymorphism is the ability of one item to assume many forms. In object-oriented programming, this concept allows multiple functions with the same name, but different in the number and/or type of parameters, to have multiple implementations. This is also known as method overloading. With this form of polymorphism, a single class can behave differently given a variety of circumstances. Another example of polymorphism is known as method overriding. This involves a method of the same
signature, i.e. the same name with the same number and type of parameters, in both a class and its subclass. The determination of which method is invoked is dependent upon the matching of the method invocation to the method, class, and object. Given a particular event, the more complex class is able to operate in a different manner than the basic class.

Encapsulation is the combination of the methods and the data manipulated by the methods into a single component. The methods and data are wrapped defining the behavior of the component. This wrap also protects the methods and data from being misused by other methods. The basis of encapsulation is to formulate a class to represent an abstract set of objects that share a structure and behavior.

2.3 Satisfying the Software Design Specifications Using Java

Java is used as the programming language for product development of the package in order to meet a set of software design specifications. The specifications consist of consistency, encapsulation, interactive nature, extensibility, reusability, portability, hardcopy capability, and platform independence. Many of which are achieved via the object-oriented features of Java. Platform independence is a unique feature of Java. Each criterion is examined and some of the criteria are related to the details of the graphtools package.
2.3.1 Consistency

With consistency, the various types of plots operate based on the same set of principles. This is accomplished through inheritance and polymorphism. For example, the Axis class is extended by LogAxis and PolarAxis. The LogAxis class and the PolarAxis class inherit the data and methods contained in the Axis class. Even so, both classes use method overriding for the methods which draw the horizontal and vertical axes. Thus, polymorphism is necessary to obtain the proper appearance for the desired type of plot.

On the other hand, inheritance allows the different plot types to operate in the same manner. For example, all three types of plots use automatic scaling of the axes in the construction of the graph. Afterwards, manual scaling of the axes and labeling of the axes are two possible options available to the user. Manual scaling of the axes changes the minimum and maximum values for the objects corresponding to the x-axis and y-axis. Labeling of the axes calls a method in Axis to set the title of the axis. This shows the consistency and ease of use to create the graph.

2.3.2 Encapsulation

With encapsulation, the tools should be able to handle the given tasks with respect to the window manager. This is achieved by creating the figure to maximize the size of the graph within the window. Hence, changing the size of the window in turn changes the size of the graph.
The tools should also be able to handle the tasks with respect to the data. This is seen when creating a graph or adding/removing a set of data. The $DataSet$, $Axis$, and $Graph2D$ objects operate as a union. Given the minimum and maximum values of the $DataSet$, the instances of $Axis$ set the minimum and maximum values of the data to be displayed. Then, the methods within $Graph2D$, the canvas for the graph, initiate the drawing of the curve connecting the data points using linear interpolation. When a $DataSet$ is either added or removed from the graph, the minimum and maximum values of the data to be displayed are evaluated and adjusted, if necessary, to maximize the data shown on the graph.

2.3.3 Interactive Nature

Another consideration is for the package to have an interactive nature. It is helpful to the user examining the data to be able to focus on a particular area of the graph. The ability to manually scale the axes provides the option of viewing the entire curve or only part of the curve. It is also beneficial to the user to be able to view the actual data point values. The package is able to display a data point in a panel to the right of the graph. More information concerning these features is available later in this chapter.

2.3.4 Extensibility

Extensibility implies the availability of a wide variety of plots, with a focus on plots of a linear system/control nature. The assortment of plots is achieved through the
use of \textit{Axis} and its subclasses. The \textit{Axis} class is used to create the x-y plot type. The \textit{LogAxis} class can be used to create a semilogx plot, semilogy plot, or loglog plot, where the x-axis, y-axis, and both axes, respectively, are scaled logarithmically. The \textit{PolarAxis} class is used to create the appearance for the polar plot. This includes a grid made up of magnitude circles.

The focus on control system analysis and design utilizes the three plot types above. The output to a given input in the time domain, typically a step response, is created utilizing the x-y plot type. Looking at the frequency response of the system can require one of two cases. Initially, the use of two semilogx plots, one showing the magnitude and the other showing the phase, drawn as two subplots in one figure is used to display the frequency response. Finally, the open-loop frequency response is best displayed as a polar plot.

It is possible to extend the variety of plots in the \textit{graphtools} package to incorporate three-dimensional plots. A \textit{Graph3D} class can be created to fulfill this option, calling the classes and methods of the \textit{Graph2D} class for ease of development. The ability to create three-dimensional plots is useful in the area of nonlinear control.

Extensibility is not only concerned with the variety of plots; it is also concerned with the expansion of the package as a whole. It is possible to extend the functionality of the \textit{graphtools} package to provide the possibility of simulating a system. To complete this task, classes need to be added to represent the model of the system and to perform the calculations of the simulation, where the output of the simulation would be a data file to be loaded and displayed using the current \textit{graphtools} package.
2.3.5 Reusability

Reusability suggests that the documentation and software approach facilitate easy modification and addition. The documentation is comprised of this thesis, this chapter and the appendix in particular, and the comments inserted throughout the code. It is a good practice to include comments at the beginning of each class or method to state its purpose. Comments to explain the use of the variables are also desirable. With a special form of commenting in place, the automated documentation tool, javadoc, converts portions of source files into HyperText Markup Language (HTML). The HTML files for the graphtools package, in addition to the code, are on the compact disk accompanying the thesis. In short, the documentation is helpful to maintain and add to the package.

Object-oriented languages make additions to programs less complicated, because designing classes built from other classes is at the core of object-oriented programming. Hence, there is an emphasis on reusability for any designer using an object-oriented language. A designer can use a group or library of complex objects to assemble or add to a program. The alternative is to compose every program using only a set of basic commands. This creates a program requiring numerous functions to perform a task.

2.3.6 Portability

The package should be designed for importation into productivity software. The implementation of this task is dependent upon the productivity software. If it is also written in Java, there are two options. One option is to use the import command to create objects and call methods of the PlotWin class, which creates the GUI to permit the user to
make selections and display the plot. The other option is to specify the package path name for the class when creating objects and calling methods of the *PlotWin* class.

If the productivity software is not written in Java, it is possible to access the *PlotWin* class using system commands, such as DOS commands, assuming an acceptable version of the JDK and/or the JRE is installed. Software, such as Matlab, has the capability to run system commands. This technique, to launch the windows of the *graphtools* package using system commands, is discussed in more detail in the appendix.

### 2.3.7 Hardcopy Capability

The ability to generate a hardcopy is essential. The ability to send information to a printer is addressed beginning with JDK 1.1, although there are limitations to the printing. The release of JDK 1.2 introduced more control over the page to be printed, such as the option to invoke a print dialog, to get the page format, and to set the physical characteristics of the paper. In the *graphtools* package, the window containing the figure is centered in the 8.5" x 11" page, assuming portrait orientation of the page.

### 2.3.8 Platform Independence

Platform independence is important when considering the issues of portability and prolonged existence. Designers of the Java programming language made decisions to permit programmers to "write once, run anywhere" [5]. Hence, Java is able to work on multiple platforms, rather than a single hardware architecture. The programmer can now
concentrate on writing high-quality code and Java technology will provide the ability to run the program on various platforms, such as Macintosh, IBM (Windows), and UNIX. The architecture-neutral nature of Java technology is also important due to the popularity of the Internet and the wide range of software available.

2.4 Features Included in the Package

Computer users focus solely on the features of the package. Those who design and/or maintain the software focus on the features of the package as well as the software design specifications. Decisions concerning the implementation of the desired features are related to the appearance and usability of the package. Choices made in the design and maintenance will affect the opinion of the users.

To further document the graphical tool presented in this thesis, the features of the graphtools package will be explored. The package gives the user the opportunity:

➢ To view or create a data file using a text editor,
➢ To load a data file in ASCII format,
➢ To plot the data using one of a variety of plot types including:
  x-y, semilogx, semilogy, loglog, or polar,
➢ To display the data as a single plot or two plots using the 2x1 subplot option,
➢ To control the display of the plot by
  • adding grid lines,
  • adding annotation,
  • selecting and displaying a data point,
• scaling the axes, and

➢ To print a hardcopy.

These features are incorporated into a user-friendly graphical user interface, so that the user is not required to have programming experience. With this application, the users can concentrate on the analysis of the system, and those who maintain the software can take care of the programming issues.

2.4.1 Appearance and Usability of the Package

With the intention of better illustrating how these features are attained, the appearance and usability of the package is introduced. A program’s appearance is determined by its presentation to the user. The appearance of a program includes characteristics such as window size, layout of the objects, colors, and GUI controls, such as buttons, pull-down menus, check boxes, and scrollbars. A program’s usability is determined by the user-friendliness of the interactive controls and the functionality of the events triggered by the interactive controls. It is the designer’s selection and implementation of the interactive controls that influence the user’s satisfaction.

The appearance of the package cannot be obtained solely by reading this thesis. Even so, a black and white appearance of the package can be obtained. On the other hand, one must have hands-on experience to sense the usability of the package and determine if the package satisfies his/her expectations.
2.4.2 Selecting the Desired Options and Displaying the Data

The main window, entitled Plot and shown in Figure 2.1, is revealed upon initiation of the application. This window contains four regions of GUI controls with the purpose of viewing or creating a data file, selecting a data file to be displayed, choosing the type of plot and possibly checking the subplot option, and displaying the plot.

![Plot Window](image)

**Figure 2.1**: Plot Window – User Interface to Make Desired Selections

The bottom left region, “Text Editor”, contains two buttons. Selecting the “View Data File” button opens a file dialog box, entitled Open File and shown in Figure 2.2. Within this dialog box, the user can either type the name of the file, including wildcards, or select a file using the mouse. Selecting the “Cancel” button within Open File dialog box will abort the event and return to the Plot window. Assuming that an acceptable name is entered or a file is selected, the information in the data file will be displayed in a new window, entitled Text Editor and shown in Figure 2.3. However, if the “Create Data
File” button is selected, the Text Editor window is displayed with a blank text area to begin typing the values of the data points. Take note of the usual menu options, shown in Figure 2.3, available for a text editor.

**Figure 2.2:** Open File Dialog – Allows the User to Browse for a File

**Figure 2.3:** Text Editor – View or Create a Data File
The top left region, “Data File”, is used to select a data file in ASCII format. The user can type the path and filename into the text field. The user has the alternative to press the button labeled “...” with the intention of browsing to find a file. The “...” button reveals the Open File dialog box, as in the case of selecting a data file to view. When an acceptable file is chosen, the path and file name are shown in the text field.

The top right region, “2D Plot”, is used to select the desired type of plot. The first decision incorporates a choice or pull-down menu, as shown in Figure 2.4. The other decision is whether or not to utilize the subplot option. At this time, the subplot option is limited to two plots displayed in two rows and one column.

![Plot Window](image)

**Figure 2.4:** Plot Window – Select the Desired Type of Plot to Be Displayed

The bottom right region contains the “Display Plot” button. Obviously, this button initiates the event to create a graph of the data. In the handling of this event, the Open File dialog box will be displayed only if no data file has been chosen from the
“Data File” region. The default values of an x-y plot without utilizing the subplot option are applied in the case that these options are not changed. In the event that an error occurs, such as failure to load the data file, a dialog box, entitled *Notification* and shown in Figure 2.5, will open to display the error message.

![Notification Dialog](image)

**Figure 2.5:** Notification Dialog – Displays a Message for the User

The termination of the application occurs when the window destroy (“X”) button in the upper right corner of the window is pressed. An alternative is to use “Close” from the *Plot* window’s menu, as shown in Figure 2.6, which is viewed upon clicking the icon in the upper left corner.

![Plot Window](image)

**Figure 2.6:** Menu to Resize, Move, or Close the *Plot* Window
2.4.3 Acceptable Data File Formats

Before a graph can be displayed, a data file must be read without errors; this requires a data file in ASCII format. For a graph to be displayed as expected by the user, the data file must have a suitable format. In the software design, assumptions are made as to the format of the data files for various plot types.

First and foremost, each data file consists of the set of data points to display one curve or, in the case of the subplot option, to display two curves with one curve in each of the two plots. To display multiple sets of data points on a single set of axes, one data file is selected to create the plot. Then, other data files can be selected to indicate the set of data points to be added to the graph, as shown in Section 2.4.4.

Furthermore, the data points need to be arranged in columns and separated using only the space or tab key. For clarity, the format of the data file is shown as

\[ \text{[column}_1, \text{column}_2, \ldots, \text{column}_q], \]

where there are \( q \) columns of data.

With the exception of the polar plot, the data file is assumed to be in the form

\[ [x, y] \]

for a single plot. The two columns of data are displayed with the first column on the x-axis and the second column on the y-axis. The general x-y plot type does not scale either of the axes. The semilogx plot type provides a logarithmic scale for only the x-axis. The semilogy plot type provides a logarithmic scale only for the y-axis. Lastly, the loglog plot type provides a logarithmic scale for both the x-axis and the y-axis.
On the other hand, the data file is assumed to be in the form

\[ [x_1, y_1, x_2, y_2] \]

for two plots using the subplot option. The first two columns are displayed on the top graph and the second two columns are displayed on the bottom graph, repeating the method of displaying the single plot.

In control engineering, two types of plots, the semilogx plot type with the subplot option and the polar plot type, are used to view frequency response data. In either case, the data is assumed to be in the form

\[ [\text{magnitude}, \text{phase}, \text{frequency}] \],

where the magnitude is in decibels, the phase is in degrees, and the frequency is in radians per second. The semilogx plot type with the subplot option uses the frequency as the independent variable. The magnitude is displayed on the top graph and the phase is displayed on the bottom graph. Although the frequency data is not used to draw the actual curve for the polar plot type, the frequency data is displayed when a data point is selected. This allows the user to obtain the phase crossover frequency and the gain crossover frequency. The measures of relative stability, gain margin and phase margin, are obtained from either of these plot types.

Note that the package can be modified to permit other assumptions to be made with the purpose of tailoring the package to perform specific tasks. One example is to have multiple columns of data to be displayed on the same graph. In this case, varying the line color or using different line styles can be used to differentiate the curves from one another.
Start by taking the leftmost columns of data and going to the right based on the type of plot. Hence, for a single plot, excluding the polar plot type, the data file format is

\[ [x_1, y_1, x_2, y_2, \ldots, x_n, y_n], \]

where there are 2*n columns of data. This data file format displays a family of n curves. The subplot option can be used when n is even. In this case, the first four columns could represent one set of curves, the next four columns could represent the next set of curves, and so on until all the columns are used.

On the other hand, a frequency response requires three values for each data point. So, the data file format for the semilogx with the subplot option is

\[ [\text{mag}_1, \text{phase}_1, \text{freq}_1, \text{mag}_2, \text{phase}_2, \text{freq}_2, \ldots, \text{mag}_n, \text{phase}_n, \text{freq}_n], \]

where there are 3*n columns. Although more columns are required, this data file format also displays a family of n curves. Note that the polar plot type only displays one set of data points.

2.4.4 Modification of the Figure

The "Display Plot" button of the Plot window initiates the creation of a new window, entitled Figure, in which the graph is displayed. If the plot is not displayed as desired, use the "Close" menu item in the "Window" menu, shown in Figure 2.7, or the window destroy ("X") button to dispose of the window. The Plot window can be used to change the options and display the plot again.
Once the data is displayed using the proper options, there are several features available to control the appearance and usability of the graph. As shown in Figure 2.8, the “Figure” menu contains choices which affect the entire graph.
The choices are listed as follows:

- **"Add Data"** – This item is used for the comparison of data. Once selected, the user is prompted for another data file using the *Open File* dialog box. Assuming no errors are encountered, the new set of data points is added to the current graph.

- **"Remove Data"** – This item is used to remove a set of data from the current graph. This item requires that more than one set of data be displayed. The data to be removed is chosen by the selection of a data point.

- **"Reset to Default Settings"** – This item resets the axes to display all of the data. An alternatively is the manual keystroke “R”. If the axes were manually scaled, this item is a convenient way to reset the axes. This item will also fix those rare occurrences when there is a problem with the automatic scaling while displaying the data, such as when data is added and the axis exponent is added or changed.

- **"Redisplay with Current Settings"** – This item repaints the graph without resetting the axes. An alternatively is the manual keystroke “r”.

- **"Print"** – This item opens the dialog box of the computer's default printer for confirmation of the settings and initiation of the transfer of the current graph to the printer. Upon the transfer, the image of the figure is translated in order to be centered in the 8.5” x 11” page. An assumption is made to use portrait orientation of the page.
The "Customize" menu, shown in Figure 2.9, contains choices which affect parts of the graph.

![Customize Menu](image)

**Figure 2.9: Figure Window – Customize Menu**

The choices are listed as follows:

- **"Grid"** – This option adds dashed lines, or dashed circles for the polar plot, at the value of the tick marks.
- **"Zero Lines"** – This item adds a line at the point where the value of data attached to the axis is zero. This is done for both the horizontal and vertical axes.
- **"Annotate"** – This item is used to label the graph. The *Annotate* dialog box, shown in Figure 2.10, is revealed to prompt the user for the title of the graph, a description for the x-axis ("XLabel"), and a description for the y-axis ("YLabel"). Any of these labels can be changed by repeatedly selecting the
"Annotate" menu item and editing the current string, if any. To remove a label, select the "Annotate" option and clear the text field for the label to be removed.

![Annotate dialog box]

**Figure 2.10:** Annotate TextBox – Get User Input to Label the Figure

➢ "Manual Range" – This item opens the Manual dialog box, shown in Figure 2.11. The current minimum and maximum values for the x-axis and y-axis are displayed. Editing any or all of these values changes the scale of the axes.

![Manual dialog box]

**Figure 2.11:** Manual TextBox – Get User Input to Manually Scale the Axes
➢ “Select Data Point” – This item puts a filled circle on the curve at the position of the first data point and adds a panel to the figure to display the value of the first data point. An error occurs when the first data point is not currently displayed on the graph. After a data point has been selected, the user is able to use the right and left arrow keys to cursor through the data points. The escape key clears the data point.

There are alternatives to selecting and removing a data point. One alternative is to double click the mouse on the graph near the desired data point. This displays the closest data point to the location of the mouse at the time of the double click. The other alternative is to click the mouse near the desired data point and use the manual keystroke “d” to select the closest data point to the location of the mouse at the time of the click. The alternative to erase the data point is the manual keystroke “D”.

Some of the previous options have special circumstances when the subplot option from the Plot window is checked or enabled. For instance, the user can reset the top graph, bottom graph, or both graphs. Hence, when the “Reset to Default Settings” option is chosen, the Subgraph dialog box, shown in Figure 2.12, prompts the user to choose.

![Subgraph Dialog Box]

Figure 2.12: SubGraph TextBox – Select the Graph on which to Perform the Option
Also, it is only possible to annotate and manually scale one graph at a time. Thus, when either of these options are chosen, the Subgraph dialog box again prompts the user to choose. In these cases, the option of “Both” graphs creates an error.

There is an additional option only when the semilogx plot type is used. The “Wrap Phase” option is to be used for the phase plot of a frequency response. This option will display the phase with a minimum of \(-180^\circ\) and a maximum of \(180^\circ\). Values outside this range are displayed with equivalent values within the range, e.g. \(360^\circ\) is \(0^\circ\).

The “PolarPlot” menu, shown in Figure 2.13, is only available when the data is displayed using the polar plot type.

![Figure 2.13: Figure Window – PolarPlot Menu](image)

The choices are listed as follows:

> “Constant Magnitude” – This item opens the Magnitude dialog box, shown in Figure 2.14, prompting the user to enter a value in decibels. A constant
magnitude circle is added to the current graph. In addition, the value of the constant magnitude circle is displayed to the right of the graph.

![Figure 2.14: Magnitude TextBox – Get User Input to Add a Constant Magnitude Circle](image)

- "Constant Phase" – This item opens the *Phase* dialog box, shown in Figure 2.15, prompting the user to enter a value in degrees. A constant phase line is added to the graph. In addition, the value of the constant phase line is displayed to the right of the graph.

![Figure 2.15: Phase TextBox – Get User Input to Add a Constant Phase Line](image)
“M-Contour” – This item opens the MContour dialog box, shown in Figure 2.16, prompting the user to enter a tolerance in decibels. The constant closed-loop magnitude contours are added to the graph. In addition, the values of the constant closed-loop magnitude contours are displayed to the right of the graph.

![MContour dialog box](image)

**Figure 2.16: M-Contour TextBox – Get User Input to Add M-Contours**

All of the features included in the package and described above are not necessary for all plot types. Hence, unnecessary features are disabled. A feature is disabled if the menu item is concealed or written in gray text, rather than black text.

### 2.4.5 Sample Figures Utilizing Various Plot Types and Features

The following three examples illustrate the use of the three main plot types used in control engineering and various features included in the package. These examples are not intended to be a complete illustration of the plot types and features of the package.
Figure 2.17: Example – X-Y Plot with a Data Point Displayed

The second example uses the semilogx plot type and the subplot option to create a frequency response plot. The steps to create Figure 2.18 are as follows:

1. Create an ASCII file containing the data points. The data file can be in the form of three columns or four columns. For the case of three columns, the first, second, and third columns correspond to the magnitude, phase, and frequency, respectively. For the case of four columns, the first, second, third, and fourth columns correspond to the frequency, magnitude, frequency repeated, and phase, respectively. The frequency has units of radians per second, the magnitude has units of decibels, and the phase has units of degrees. Save the file with “txt” as the extension, i.e. “filename.txt”.

2. Run the program. Refer to the appendix for more details.

3. From the Plot window, select a data file by typing the path and filename in the “Data File” text field or by pressing the “…” button to browse and open a file.
4. Select “semilogx” from the “2D Plot” choice menu. Check the “Subplot” box
to enable the subplot feature.

5. Press the “Display Plot” button.

6. Select the “Annotate” menu item from the “Customize” menu in the Figure
window. First, select the top graph. Type “Frequency Response” in the
“Title” text field, type “Frequency (radians/second)” in the “XLabel” text
field, and type “Magnitude (decibels)” in the “YLabel” text field.

7. Select the “Annotate” menu item again. Now, choose the bottom graph. Type
“Frequency (radians/second)” in the “XLabel” text field and type “Phase
(degrees)” in the “YLabel” text field.

Figure 2.18: Example – Semilogx Plot Using the Subplot Option
The final example uses the polar plot type to create an open-loop frequency response plot. The steps to create Figure 2.19 are as follows:

1. Create an ASCII file containing the data points. The data file needs to be in the form of three columns, where first column corresponds to the magnitude in decibels, the second column corresponds to the phase in degrees, and the third column corresponds to the frequency in radians/second. Save the file with "txt" as the extension; i.e. "filename.txt".

2. Run the program. Refer to the appendix for more details.

3. From the Plot window, select a data file by typing the path and filename in the "Data File" text field or by pressing the "..." button to browse and open a file.

4. Select "polar" from the "2D Plot" choice menu.

5. Press the "Display Plot" button.

6. Using the "PolarPlot" menu, select the "Constant Magnitude" menu item. In the Magnitude dialog box, type "-16" in the text field. Press the "OK" button.

7. Using the "PolarPlot" menu, select the "Constant Phase" menu item. In the Phase dialog box, type the value "223" or "-137" in the text field. Press the "OK" button.

8. Using the "PolarPlot" menu, select the "M-Contour" menu item. In the MContour dialog box, type "3" in the text field. Press the "OK" button.

9. Double click the mouse on the curve near the desired data point. Use the left and right arrow keys to cursor through the data points until the desired data point is displayed.
Figure 2.19: Example – Polar Plot with All PolarPlot Options and a Data Point

These examples conclude the discussion of the available features. They also provide the user with a starting point to become familiar with the software. Error messages are provided, as necessary, in an attempt to provide instructions for the user.
Chapter 3

A New Graphical Technique for Control System Analysis and Design

This chapter begins with a brief discussion of the control system design process. A new graphical technique, for systems of approximately second order, is introduced. This technique involves the use of two sets of design curves that are derived to relate time-domain specifications to frequency-domain specifications. Thus, a controller can be designed using frequency-domain techniques given design specifications in the time domain.

Then, a method to design a digital PID controller with an emphasis on frequency response techniques is presented. The chapter concludes with an example that uses the design curves and the controller design method in an attempt to improve the transient response of a system. Within the example, the data is shown graphically using the Java application presented in Chapter 2. The example should demonstrate the value of the design curves. In addition, the example should demonstrate the value of the Java package by showing the appearance of plots, including the use of several features, to display various sets of data points.
3.1 Control System Design Process

A goal of control engineering is to achieve improved performance of a system to provide a solution to an actual need. To accomplish this task, there are several steps in the control system design process that are performed. First, analysis of the system needs to be completed in order to establish the system goals, the variables to control, and the specifications for these variables. This leads to the identification of a sensor to measure the controlled variable. Clearly, an actuator to drive the system must be chosen.

System identification techniques are performed in order to calculate an adequate mathematical model to represent the system. Indisputably, this is not a trivial task. The model represents the internal components, which determine the dynamic response, of the system. The model also represents external factors, such as noise or other disturbances that may be injected into the system. It is desirable to model the system in its linear range and design a controller to filter the noise and to reject any disturbances.

A controller is developed based on the mathematical model and the control criteria. The system, including the prospective controller, is evaluated analytically and by simulation. If the controller does not meet the design requirements, it needs to be modified. When a controller has successfully met the design requirements on the mathematical model, it can be physically implemented and tested on the actual system. Successful operation on the physical system will verify the accuracy of the model.

The control system design presented in this thesis focuses on developing a controller using frequency-domain design techniques. Two sets of design curves are formed showing the relationship between time-domain and frequency-domain
characteristics of a system. Equations to design a digital PID controller, resulting in a z-plane transfer function, are shown. An example makes use of the design curves and the digital PID controller design. In addition, the plots for the example are constructed using the Java application presented earlier in this thesis.

3.2 Design Curves

In the process of designing a controller, there is an obvious concern for system stability. However, stability is not the only interest in the design process. For instance, a stable system must have an acceptable transient response. Also, the model used to characterize a system is never exact. The model may indicate that the system is stable, when in reality the physical system is unstable. Hence, it is generally required that a system be stable within a margin of safety.

3.2.1 Problem Statement

Speculate a second-order type-one system, where type one implies one pole of $G(s)$ at the origin and $G(s)$ is the forward path transfer function as shown in Figure 3.1.

![Figure 3.1: Closed-Loop Control System Block Diagram](image)
Assume a plant transfer function in the form,

\[ G(s) = \frac{\omega_n^2}{s(s + 2\zeta \omega_n)} \quad (3.1) \]

Also, assume negative unity feedback, or \( H(s) = 1 \). Hence, the closed-loop transfer function is

\[ T(s) = \frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)} \]

\[ T(s) = \frac{\omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2} \quad (3.2) \]

### 3.2.2 Relative Stability

Two measures of relative stability of a system are gain margin and phase margin. The gain margin is the factor by which the open-loop gain must be changed to cause marginal stability of the closed-loop system. The gain margin (GM) is defined as the reciprocal of the gain \(|GH(j\omega)|\) (of unitless magnitude) at the frequency at which the phase angle crosses -180°. If

\[ GM = \frac{1}{|GH(j\omega)|} = \frac{1}{d} \quad (3.3) \]

then,

\[ GM = -20 \log(d) \text{ dB} \quad (3.4) \]

Hence, it is also the negative of the magnitude in decibels. The frequency in which this occurs is known as the phase crossover frequency.
The phase margin ($\Phi_{pm}$) is defined as the angle through which the $|GH(j\omega)|$ locus must be rotated in order that the unity magnitude (unitless) point passes through the (-1, 0) point in the $GH(j\omega)$ plane. The (-1, 0) point is equivalent to a logarithmic magnitude of 0 dB at a phase angle of 180°. The frequency in which this occurs is known as the gain crossover frequency, $\omega_c$, and hereafter referred to as crossover frequency. At times, it is referred to as the open-loop bandwidth.

The gain margin and phase margin can be seen in Figure 3.2, which shows the general shape of the open-loop frequency response of $GH(j\omega)$.

![Polar Plot for $GH(j\omega)$]

**Figure 3.2:** Polar Plot for $GH(j\omega)$

### 3.2.3 Relationship Between the Time Domain and Frequency Domain

The phase margin is used to determine the relationship between time-domain specifications and the frequency-domain specifications. The properties of the point at which the phase margin is determined are used to find a function to describe the
crossover frequency, $\omega_c$, in terms of the damping ratio, $\zeta$, and the natural frequency, $\omega_n$. For the second-order system described by the transfer function,

$$GH(j\omega) = \frac{\omega_n^2}{j\omega(j\omega + 2\zeta\omega_n)}, \quad (3.5)$$

the magnitude of the frequency response is equal to one (unitless) at $\omega_c$. Thus,

$$|GH(j\omega_c)| = \frac{\omega_n^2}{\omega_c \left(\omega_c^2 + 4\zeta^2 \omega_n^2\right)^{1/2}} = 1$$

$$\omega_n^4 = \omega_c^2 \left(\omega_c^2 + 4\zeta^2 \omega_n^2\right)$$

$$\omega_c^4 + 4\zeta^2 \omega_c^2 \omega_n^2 - \omega_n^4 = 0$$

$$\frac{\omega_c^2}{\omega_n^2} = \left(4\zeta^4 + 1\right)^{1/2} - 2\zeta^2$$

$$\omega_c = \omega_n \sqrt{(4\zeta^4 + 1)^{1/2} - 2\zeta^2} \quad (3.6)$$

The phase margin for the system is found in terms of the damping ratio [1].

$$\Phi_{pm} = 180^\circ - 90^\circ - \tan^{-1}\left(\frac{\omega_c}{2\zeta \omega_n}\right)$$

$$\Phi_{pm} = 90^\circ - \tan^{-1}\left(\frac{\omega_c}{2\zeta \omega_n}\right)$$

$$\Phi_{pm} = \tan^{-1}\left(\frac{\omega_n}{2\zeta \omega_c}\right)$$

$$\Phi_{pm} = \tan^{-1}\left(\frac{1}{2\zeta \left(4\zeta^4 + 1\right)^{1/2} - 2\zeta^2}\right)^{1/2} \quad (3.7)$$
These two equations correspond to the design criteria of crossover frequency and phase margin. The design criteria of percent overshoot and settling time are typically established for a system with the goal of improving the transient response. At this point, the difficulty arises. To perform frequency-domain controller design techniques, it is necessary to link the frequency-domain design criteria to the time-domain design criteria. Then, two sets of design curves can be produced to assist in the development of a controller.

As shown in [1], the percent overshoot $PO$ is

$$PO = 100e^{-\zeta\sqrt{1-\zeta^2}}, \tag{3.8}$$

and the settling time $T_s$ is

$$T_s = \frac{4}{\zeta \omega_n}. \tag{3.9}$$

Given a value for $\zeta$, the $PO$ can be calculated. Repeating this process for a range of values where $1 > \zeta > 0$ leads to a vector for the percent overshoot where $0 > PO > 100$. Using the same set of values for $\zeta$, calculate the range of values corresponding to the phase margin. A plot of the phase margin versus the percent overshoot generates the first set of design curves, shown in Figure 3.3.

Given a value for $\zeta$ and assuming a particular $T_s$, i.e. $\omega_n$, a value for $\omega_c$ is calculated. Hence, the range of values for $\zeta$ where $1 > \zeta > 0$ and a specific $T_s$, i.e. a range of corresponding $\omega_n$, are used to find a vector of crossover frequencies. Continuing to select a range of settling times, rather than a specific value, produces a
matrix relating to the crossover frequencies. The matrix can be formed so that each column of the matrix corresponds to a particular $T$, and each row of the matrix corresponds to a particular $\zeta$, i.e. $PO$. Now, this matrix is used to generate a plot of crossover frequency versus percent overshoot as a function of settling time. This family of curves, shown in Figure 3.4 and 3.5, is the second set of design curves.

Therefore, these two relationships, used to form the sets of design curves, provide the correlation between the time-domain response and the frequency-domain response. When given time-domain specifications, such as percent overshoot and settling time, frequency-domain design techniques, in which phase margin and crossover frequency are chosen, can be performed.

**Figure 3.3:** Phase Margin vs. Percent Overshoot for Second-Order Systems
Figure 3.4: Crossover Frequency vs. Percent Overshoot as a Function of Settling Time

Figure 3.5: Crossover Frequency vs. Percent Overshoot as a Function of Settling Time
3.3 Digital PID Controller Design

The Proportional-Integral-Derivative (PID) controller is widely used in industrial processes. The PID controller is the superposition of the P, PI, and PD controllers. The following is derived in [2].

The $z$-domain transfer function of the PID controller in a parallel configuration is

$$D(z) = K_p + \frac{K_i T (z + 1)}{2(z - 1)} + \frac{K_D (z - 1)}{T z}, \tag{3.10}$$

where $T$ is the sampling period. The first term represents the proportional part of the controller with weighting constant $K_p$. The second term is a numerical trapezoidal integrator with weighting constant $K_i$. The third term is a numerical differentiator with weighting constant $K_D$.

Equivalently, the PID controller can be represented as

$$D(z) = \frac{az^2 + bz + c}{z(z - 1)}. \tag{3.11}$$

In this case, the weighting constants of Equation 3.10 are related to the coefficients of Equation 3.11 by

$$K_p = \frac{a - b - 3c}{2}$$
$$K_i = \frac{a + b + c}{T} \tag{3.12}$$

and

$$K_D = T c.$$

Analysis and design techniques for continuous systems cannot be used when given a discrete system. This is due to the different regions for stability. The stability
region for a sampled-data system in the $z$-plane is bound by the unit circle; whereas, the
stability region for a continuous-data system in the $s$-plane is bound by the imaginary
axis. Thus, to use analysis and design techniques for continuous systems, the $z$-plane
must be mapped into the $w$-plane, where the unit circle of the $z$-plane is mapped to the
imaginary axis of the $w$-plane. This is accomplished through the use of the bilinear
transformation [11] defined as

$$
z = \frac{1 + (T/2)w}{1 - (T/2)w} \quad \text{or} \quad w = \frac{2}{T} \left[ \frac{z - 1}{z + 1} \right]. \quad (3.13)
$$

The stability regions for the $s$-plane and the $w$-plane are both bound by the imaginary
axis. So, given a discrete-time system, analysis and design techniques for continuous-
time systems can be performed on the $w$-plane translation of the $z$-domain system.

In the $w$-plane, the PID controller is represented as

$$
D(w) = \frac{A w^2 + B w + C}{w(w+1)}. \quad (3.14)
$$

The $w$-plane coefficients and the $z$-plane coefficients are related by

$$
A = \frac{a - b + c}{2},
$$

and

$$
B = a - c, \quad (3.15)
$$

or

$$
C = \frac{a + b + c}{2},
$$

$$
a = \frac{A + B + C}{2},
$$

or

$$
b = C - A, \quad (3.16)
$$

and

$$
c = \frac{A - B + C}{2}.\)
The $w$-plane coefficients and the PID controller weighting constants are

$$K_p = \frac{C}{T}$$

$$K_i = B - C$$

and

$$K_D = \frac{(A - B + C)T}{2}.$$  \hspace{1cm} (3.17)

In the design process, it is desirable to achieve a specified magnitude, $M_1$, and phase, $\Theta_1$, at a specified frequency, $\omega_1$. Suppose,

$$\left| G(e^{j\omega_1 T}) \right| = M_1 \quad \text{and} \quad \angle G(e^{j\omega_1 T}) = \Theta_1$$ \hspace{1cm} (3.18)

or

$$G(e^{j\omega_1 T}) = x + jy.$$ \hspace{1cm} (3.19)

In the $w$-plane,

$$x + jy = \frac{-A\omega_{w1}^2 + C + jB\omega_{w1}}{j\omega_{w1}(j\omega_{w1} + 1)},$$ \hspace{1cm} (3.20)

where $\omega_{w1} = \tan(\omega_1 T/2)$.

Cross multiplying and equating real and imaginary parts yield

$$-A\omega_{w1}^2 + C = -\omega_{w1}^2 x - \omega_{w1} y$$ \hspace{1cm} (3.21)

and

$$B\omega_{w1} = \omega_{w1} x - \omega_{w1}^3 y.$$ \hspace{1cm} (3.22)

This is a set of two equations and three unknowns. A third specification will provide a solution to this set of equations. It can be verified that the velocity error constant is

$$K_v = \frac{2C}{T} \quad \rightarrow \quad C = \frac{K_v T}{2}.$$ \hspace{1cm} (3.23)
Assuming that the specification for $K_v$ is known, then

$$A = \frac{\omega_w^2 x + \omega_w y + (K_v T/2)}{\omega_w^2} \quad (3.24)$$

and

$$B = x - \omega_w y. \quad (3.25)$$

Obtaining the values for $A$, $B$, and $C$ complete the development of the PID controller in the $w$-plane. The transformations of Equation 3.16 determine values for $a$, $b$, and $c$ which are the coefficients for the $z$-domain representation of the PID controller. Therefore, the design is performed in the $w$-plane but the outcome is a $z$-domain transfer function for a PID controller. Now, designing in the $w$-plane is no longer necessary to obtain a $z$-domain transfer function for a PID controller.

### 3.4 Example: Digital Compensator Design

The example is an existing experiment in Control Laboratory at Ohio University. The purpose of the experiment is to develop a discrete-time compensator for the Modular Servo System from Feedback Instruments Limited, hereafter referred to as the servo system. Specifically, a Proportional-Integral-Derivative (PID) compensator will be designed in an attempt to improve the time-domain characteristics of the system in a rate control configuration with analog rate feedback.
3.4.1 Background

A block diagram of the system is shown in Figure 3.6. Figure 3.7 is a block diagram showing the structure of $G(z)$ from Figure 3.6, where $K_f$ is a forward path attenuation of 60% to ensure that the system remains in the linear region of operation and $K_r$ is the rate feedback.

![Figure 3.6: Servo System with a Digital Compensator](image)

Before designing a compensator, an experimental model is obtained via frequency response estimation. A normally distributed random sequence is the input to the servo system and the tachometer voltage is sampled. This experimental response data is processed by the Transfer Function Determination Code [12] in order to generate a discrete-time transfer function for the servo system. The model is
\[ G(z) = \frac{-0.0001062 z^2 + 0.001464 z + 0.00595}{z^2 - 0.907 z + 0.9153}. \]  \hfill (3.26)

Given the model, the completion of the example will focus only on simulated results. The actual system has nonlinearities which complicate the objective of verifying the use of the design curves to acquire a controller.

The simulated open-loop step response is shown in Figure 3.8. The graph shows an approximate percent overshoot of 19 and settling time equal to 1 second.

![Simulated Open-Loop Step Response](Figure)

**Figure 3.8: Simulated Open-Loop Step Response**

### 3.4.2 Initial Design of a PID Compensator

To improve the time-domain characteristics of the system, the design specifications for the compensator have been established as follows.
\[ PO \leq 30 \]
\[ T_s \leq 0.75\text{s} \]

and

\[ K_v = 25\% . \]  

These time-domain specifications and the design curves, as shown in Figures 3.3, 3.4 and 3.5 of Section 3.2.3, are used to determine the frequency-domain specifications which are

\[ \Phi_{pm} \geq 40^\circ \]  

and

\[ 10\text{rad/s} \leq \omega_c \leq 20\text{rad/s}. \]  

Using Equations 3.8 and 3.7 and \( \zeta = 0.36 \),

\[ PO = 29.75 \quad \text{and} \quad \Phi_{pm} = 39.30^\circ. \]  

Also, using Equations 3.9 and 3.6 and \( \zeta = 0.36 \),

\[ T_s = 1.0\text{s} \quad \text{yields} \quad \omega_c = 9.77\text{rad/s} \]

\[ T_s = 0.75\text{s} \quad \text{yields} \quad \omega_c = 13.03\text{rad/s} \]  

and

\[ T_s = 0.5\text{s} \quad \text{yields} \quad \omega_c = 19.55\text{rad/s}. \]

This range of crossover frequencies gives a more precise idea of what the settling time should be for the various designs, assuming the same phase margin for each design.

The method of compensator design, summarized earlier in this chapter, follows that contained in [2]. In short, the design process wishes to achieve a specified magnitude and phase at a specified frequency. The frequency is chosen to be the closest point of the frequency vector that is less than the desired crossover frequency; the magnitude and phase at this frequency are found using the actual frequency response data.
The open-loop uncompensated frequency response with the closest data point to $\omega_c = 13.03 \text{ rad/s}$ is shown in Figure 3.9. The data point is

$$\omega_c = 12.885 \text{ rad/s}, \quad M_p = -5.043 \text{ dB}, \quad \text{and} \quad P_p = -2.262 \text{ rad} = -129.578^\circ. \quad (3.31)$$

Using this data point, $\Phi_{pm} = 39.30^\circ$, and $K_v = 0.25$, the PID compensator is

$$D(z) = \frac{-0.75615 z^2 + 3.2906 z - 2.5319}{z^2 - z}. \quad (3.32)$$

The frequency response of the compensator is shown in Figure 3.10.

The open-loop compensated transfer function is

$$\frac{0.000080333 z^4 - 0.0014569 z^3 + 0.00058827 z^2 + 0.015873 z - 0.015066}{z^4 - 2.9066 z^3 + 2.8219 z^2 - 0.91528 z}. \quad (3.33)$$

The frequency response of the open-loop compensated system is shown in Figure 3.11.
Figure 3.10: Compensator Frequency Response

Figure 3.11: Open-Loop Compensated Frequency Response
This graph of Figure 3.11 demonstrates that the design curves and compensator design produce the expected results. It shows the data point of the crossover frequency to be

$$\omega_c = 12.885 \text{ rad/s}, \ M_p = 0.0 \text{ dB}, \text{ and } P_p = -2.456 \text{ rad} = -140.7^\circ.$$  \hspace{0.5cm} (3.34)

Hence,

$$\Phi_{pm} = 39.3^\circ.$$  \hspace{0.5cm} (3.35)

The closed-loop compensated transfer function is

$$\frac{0.000080333 z^4 - 0.0014569 z^3 + 0.00058827 z^2 + 0.015873 z - 0.015066}{1.0001 z^4 - 2.9081 z^3 + 2.8225 z^2 - 0.89941 z - 0.015066}. \hspace{0.5cm} (3.36)$$

The frequency response of the closed-loop compensated system is shown in Figure 3.12.
The simulated closed-loop step response is shown in Figure 3.13. This graph shows an approximate percent overshoot of 36 and a settling time greater than 1 second. The percent overshoot and the settling time have not improved.

![Simulated Closed-Loop Step Response](image)

**Figure 3.13: Simulated Closed-Loop Step Response**

The initial attempt of designing a PID compensator fails to meet the design specifications. Actually, the open-loop uncompensated step response is closer to meeting the design specifications than the closed-loop compensated step response. Even so, this does not imply that the design curves are useless.

There is an explanation for this initial failure. The design assumes a system of approximately second order. As seen in the closed-loop step response of Figure 3.13, the upward trend indicates that the system actually has third-order characteristics. Even so, the design curves can be used to predict the response of a third-order system.
3.4.3 Final Design of a PID Compensator

Compensator design is an iterative process. In order to meet the design specifications, the compensator must be modified. As seen in Figure 3.3, decreasing the percent overshoot is equivalent to increasing the phase margin. From Figure 3.4 or 3.5, with a constant percent overshoot, decreasing the settling time is equivalent to increasing the crossover frequency.

Using Equations 3.8 and 3.7 and $\zeta = 0.46$,

$$PO = 19.64 \quad \text{and} \quad \Phi_{pm} = 48.50^\circ. \quad (3.37)$$

Also, using Equations 3.9 and 3.6 and $\zeta = 0.46$,

$$T_s = 0.3s \quad \text{yields} \quad \omega_c = 23.60 \text{rad/s}. \quad (3.38)$$

The open-loop uncompensated frequency response with the closest data point to $\omega_c = 23.60 \text{rad/s}$ is shown in Figure 3.14. The data point is

$$\omega_c = 23.317 \text{rad/s}, \: M_p = -16.261 \text{dB}, \: \text{and} \: P_p = -2.91 \text{rad} = -166.669^\circ. \quad (3.39)$$

Using this data point, $\Phi_{pm} = 48.50^\circ$, and $K_c = 0.25$, the PID compensator is

$$D(z) = \frac{21.1321z^2 - 37.385z + 16.2553}{z^2 - z}. \quad (3.40)$$

The frequency response of the compensator is shown in Figure 3.15.

The open-loop compensated transfer function is

$$-0.002245 \frac{z^4 + 0.034918z^3 + 0.069273z^2 - 0.19865z + 0.096727}{z^4 - 2.9066z^3 + 2.8219z^2 - 0.91528z}. \quad (3.41)$$

The frequency response of the open-loop compensated system is shown in Figure 3.16.
Figure 3.14: Open-Loop Uncompensated Frequency Response

Figure 3.15: Compensator Frequency Response
Figure 3.16: Open-Loop Compensated Frequency Response

Again, the graph demonstrates that the design curves and compensator design produce the expected results. It shows the data point of the crossover frequency to be

$$\omega_c = 23.317 \text{ rad/s}, \quad M_p = 0.0 \text{ dB}, \quad \text{and} \quad P_p = -2.295 \text{ rad} = -131.5^\circ. \quad (3.42)$$

Hence,

$$\Phi_{pm} = 48.5^\circ. \quad (3.43)$$

The closed-loop compensated transfer function is

$$\frac{-0.002245 z^4 + 0.034918 z^3 + 0.069273 z^2 - 0.19865 z + 0.096727}{0.99775 z^4 - 2.8717 z^3 + 2.8911 z^2 - 1.1139 z + 0.096727}. \quad (3.44)$$

The frequency response of the closed-loop compensated system is shown in Figure 3.17.

The simulated closed-loop step response is shown in Figure 3.18. This graph shows an approximate percent overshoot of 28 and an approximate settling time of 0.5 seconds. The percent overshoot and the settling time have improved greatly to not only meet but exceed the design specifications.
Figure 3.17: Closed-Loop Compensated Frequency Response

Figure 3.18: Simulated Closed-Loop Step Response
Therefore, the example clearly illustrates the iterative nature of control system analysis and design. Although the initial attempt to meet the design specifications is not successful, the response characteristics are within an order of magnitude of the desired values. Hence, the compensator must be adjusted. Because the model is not exact, the coefficients of the controller need to be modified to account for the inaccuracies of the model. The design curves are analyzed to aid in this modification and to predict the response of the system. A new compensator is calculated and a simulation of the compensated system is performed.

The system performs as expected. In the end, the transient response of a system is improved to meet the specifications. The example demonstrates the value of the design curves and compensator design. Throughout the example, the value of the *graphtools* package is demonstrated via the use of the package to view the simulated data graphically.
Chapter 4

Conclusions and Recommendations for Future Work

The two main goals of this project are the development of a Java application to examine data graphically and the derivation of two sets of design curves to aid in controller design. In addition, the thesis presents a method for designing a PID controller with an emphasis on frequency-domain techniques. The example, digital compensator design of a servo system, utilizes the design curves and the PID controller design method in an attempt to improve the time-domain characteristics of the system. The Java application is used to view the data graphically, showing the success or failure of the design.

4.1 Conclusion

This thesis presents a Java package that was developed to be the foundation of a control system analysis tool. The graphtools package is designed to permit the user:
➢ To view or create a data file using a text editor,

➢ To load a data file in ASCII format,

➢ To plot the data using one of a variety of plot types including:
  x-y, semilogx, semilogy, loglog, or polar,

➢ To display the data as a single plot or two plots using the 2x1 subplot option,

➢ To control the display of the plot by
  • adding grid lines,
  • adding annotation,
  • selecting and displaying a data point,
  • scaling the axes, and

➢ To print a hardcopy.

It is desired to implement the features of the package in Java to meet the software design specifications. It is also desirable to take advantage of Java traits such as platform independence, the Java Application Programming Interface, and garbage collection, as discussed in Section 1.2.2.

The appearance and usability of the package is discussed in Section 2.4. It includes the steps to create a figure, an explanation of the required format of the data files, and a discussion of the options to modify a figure. Steps to create sample figures form a simple tutorial for the user. The sample figures show the accomplishment of many features included in the package. In particular, the polar plot is capable of displaying a data point, a constant magnitude circle, a constant phase line, and constant
closed-loop magnitude contours. It also shows points of interest of the open-loop frequency response, such as the gain margin and the phase margin, which measure the relative stability of the system.

In control system design, there is an obvious concern for system stability. Because a mathematical model used to represent a physical system is never exact, it is desirable to design a system that is stable within a margin of safety. The two sets of design curves shown in Figures 3.3, 3.4, and 3.5 are used to predict the phase margin and crossover frequency of a second-order system given the time-domain specifications of percent overshoot and settling time.

The value of the graphtools package and the design curves, along with the digital PID controller design method, is seen in the example. Therefore, the accomplishment of the goals is confirmed. Yet, this package is intended to be the beginning, or foundation, of a simulation package. Thus, there are several areas for future work.

4.2 Future Work

As the foundation of a control system analysis tool, the graphtools package is designed for easy modification and addition. Building upon the package to make a more complete analysis tool, a graphical block diagram display should be added. The user would then have the ability to choose from a set of blocks, change the properties of these blocks, and add lines to connect the blocks in order to form an open-loop or a closed-loop system.
At this point, the graphical display could be enhanced by the user’s ability to simulate the system. This added feature could be accomplished in various ways, including the use of Java. Using Java would require many hours of programming, which is not practical if another reasonable alternative can be found. The code could be written in C++ and implemented from the Java program using the Java Native Interface (JNI). This situation lends itself to the use of previous work by Edward Thomas as presented in "A C++ Class Library Capable of Handling Matrix, Polynomial, Transfer Function, State Space, and Frequency Response Data Types" [13]. This possibility may be a good approach. Nevertheless, it would be worth investigating the use of Matlab. Simulating the system could create an m-file, start the Matlab engine, run the m-file using the Matlab toolboxes, and save the output in the proper format to be displayed by the graphtools package.

With or without the addition of the system block diagram and simulation features, it is possible to modify the graphtools package to be used on the Internet as a Java applet. Currently, the graphtools package is a Java application to be run on a local computer with the JDK or JRE. The data files are accessed from the local computer, although a simple modification could permit the data files to be retrieved from a Uniform Resource Locator (URL), the global address of documents on the World Wide Web [14].

At the completion of these enhancements, the software package would be a valuable contribution to a control theory class or laboratory, especially one that is available via the Internet to multiple universities. The class or laboratory can include theory of various practical applications of control theory, which are accessible over the
Internet, such as Flexlab at Ohio University [15]. After learning the theory, the students can operate the hardware and create an output file. The software can plot the data, be used to create a block diagram of the hardware system, and finally simulate the block diagram in an open-loop or a closed-loop situation. After an open-loop simulation, a controller can be designed to achieve the desired results. After the closed-loop controlled simulation, the output will show the success or failure of the controlled system.

Before and after making modifications and additions to the package, extensive testing of the package by multiple users should be done. Users within the organization that developed the software complete the first stage of testing, called alpha testing [14]. The second stage of testing, called beta testing, involves selected external users [14]. The two stages of testing will show that the package is capable of performing the tasks of the required features. They will also reveal the level of user-friendliness of the package. Initial use of the package by test groups could reveal bugs in the code. Also, suggestions from the test groups could give insight into new features to increase the appeal of the package.
References


Appendix

Using and Maintaining the *graphtools* Package

This appendix contains general instructions to use and maintain the Java package. Section A.1 contains instructions to install the JDK and the *graphtools* package. A list of the files included in the *graphtools* package is contained in Section A.2. Instructions to compile and run the application are provided in Section A.3.

A.1 Installation Instructions

To install the JDK, the self-extracting executable file must be downloaded from the Internet. Start at Sun’s Java web site [3]. Select the Java platform as “JDK” from the choice list, or pull-down menu. Select the JDK 1.1.x release version. Alternatively, go directly to the download page for the JDK at <http://java.sun.com/products/jdk/1.1/>.

Read and accept the license agreement. Follow the steps to download the JDK software. Save this file to a temporary directory. Following the instructions from the JDK download web page, start the installation setup by running the self-extracting executable. After completion of the setup, set the PATH and CLASSPATH variables as described in
the installation notes from the "readme.txt" file. Otherwise, the full path is required to
run the Java compiler or interpreter.

The graphtools package should be copied from the "graphtools" directory to the
"~\java\graphtools\" directory. Note that "~\java\" should be replaced by the full path of
the directory structure for the directory containing the JDK. To be sure all of the files are
copied correctly see Section A.2 for a list of the files.

A.2 List of Files in the graphtools Package

The following is a list of files included in, supporting, or used by the graphtools
package. Note that because these Java source files create a package, they must be located
in a directory called "graphtools." The text file, markers.txt, contains various shapes to
be drawn at each data point to create different line types. The remaining three text files
are the data files used to create the sample figures of Section 2.4.5.

Java source files:
Axis.java       LoadData.java       PolarAxis.java
DataSet.java    LogAxis.java       RTextLine.java
DisplayPlot.java Markers.java     SpecialFunction.java
Figure.java     MyMenuBar.java     TextBox.java
G2DInt.java     Notification.java TextLine.java
G2Dmenu.java    Plot.java         TextState.java
Graph2D.java    PlotWin.java      ViewData.java

Supporting text file:
markers.txt

Data files used to create sample figures:
expolar.txt     exsemix.txt     exstep.txt
### A.3 Compilation and Execution

The Java source files can be viewed or edited using a standard text editor. The Java compiler, `javac`, translates the text of the Java source file into instructions for the Java Virtual Machine (JVM). The instructions are stored into a class file containing bytecodes. In other words, “filename.java” is compiled to create “filename.class” to be interpreted by the JVM. The JVM is implemented by the Java interpreter, `java`. The interpreter translates the bytecodes into machine code, understandable by the specific platform of the computer.

Thus, to compile the `graphtools` package, go to a command prompt and change directories to the `graphtools` directory, “~/java/graphtools/”. Then, type “javac *.java” at the command prompt. This compiles all Java source files within the current directory and displays any errors or warnings. Assuming that there are no errors from the compilation, the package is ready for use. Go back in the directory structure by one directory to “~/javal” and type “java graphtools.Plot” to execute the program.

There are many alternatives to these steps which include the use of an integrated development environment (IDE). If there is a desire to use an IDE, follow the instructions given by the developer of the IDE. Many IDEs exist to provide shortcuts for compilation and execution of Java programs. They also provide the opportunity to view a list of the source files of the package, one or more source files, the compilation results, and any messages sent to the monitor.
HUITGER, CATHERINE ANN. M.S. November 2000
Electrical Engineering

**Visual Package for Control System Design Using a Java Infrastructure**  (76 pp.)

Director of Thesis: Dr. R. Dennis Irwin

One main goal of this project is the development of a Java application to examine
data graphically. The tool to graphically display data, or *graphtools* package, is
constructed to provide the opportunity to view or create a data file using a text editor, to
load a data file, and to display a linear interpolation of the data points using a variety of
plot types, including a 2x1 subplot option, with a focus on plots of a linear system/control
nature. To find certain points of interest of a system’s frequency response, the polar plot
includes the unique capability of displaying a constant magnitude circle, a constant phase
line, and constant closed-loop magnitude contours, in addition to displaying a data point.
The user has the ability to interact with the graph to control the display and to print a
hardcopy of the graph. These features are incorporated into a user-friendly graphical user
interface. Hence, the user can concentrate on the analysis of the data rather than
programming issues.

The other main goal is the derivation of two sets of design curves to aid in
to

controller design. The design curves are used to provide a correlation between the time-
domain response and the frequency-domain response of a system. Thus, when given
time-domain specifications, such as percent overshoot and settling time, frequency-
domain design techniques, choosing a phase margin and crossover frequency, can be
performed to obtain a controller to meet the design specifications.