MASCoT: a Model-based Automatic Software Compensation Toolkit for Feedback Systems

by

Patrycja Ewelina Missiuro

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000

Author . . . .
Department of Electrical Engineering and Computer Science
May 22, 2000

Certified by . . . .
James K. Roberge
Professor of Electrical Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MASCoT: a Model-based Automatic Software Compensation Toolkit for Feedback Systems

by

Patrycja Ewelina Missiuro

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2000, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The process of designing a compensator for a feedback system has no well-defined algorithms that can guarantee success. Experience and innovation are often a designer's best weapons in solving feedback control problems. The only reasonably successful attempt at automating the design of compensators, the *Matlab Control System Toolbox*, comes from *Mathworks*. It is an add-on which requires their *Matlab* program and is costly, system dependent, and difficult to use.

My thesis combines different electrical engineering approaches at series compensation of linear feedback systems to design a platform independent and very easy-to-use software toolkit called *MASCoT*. This toolkit allows a designer to quickly visualize and automatically evaluate several solutions to a compensation problem. *MASCoT* is written in Java and can run on any platform as well as on the web. The software is free to use and distribute and can aid students and others interested in system control and compensation techniques and who wish to evaluate and design their own compensators.

Thesis Supervisor: James K. Roberge
Title: Professor of Electrical Engineering

Acknowledgments

Great thanks to my supervisor, *Professor James K. Roberge*, who is responsible for everything I know about electrical engineering principles. He has been a great teacher and a patient mentor throughout my MIT career. He always supported me with direct and constructive comments. His kind words and patience allowed me to complete this project.

Thanks to my wonderful boyfriend, *Ovidiu*, who took the time to proofread and criticize my writing. He patiently withstood stressful times when my thesis was being developed.

Thanks to the rest my family and friends for being there when I needed them: *Denver, Ewey, Carolina, my mom and dad, Ray Paradis, Edith, Justynian, Aurelia Julia, Ewaryst, Beata, Kuba,* and others who know who they are.

And lastly, to *Anne Hunter*, an amazing course 6 administrator, thanks for putting up with me and all of the other confused EECS students.

Denver, MASCoT's mascot

Contents

List of Figures

Chapter 1

Introduction

To love oneself is a beginning of a lifelong romance.

Oscar Wilde

The development of *MASCoT* has been inspired heavily by my learning and subsequently teaching experiences in 6.302, the Linear Feedback Systems class here at MIT. As a student, I found compensation to be a confusing and non-intuitive aspect of the class, often being bogged down with the mathematical details. As a teaching assistant, I realized that most students finish the course without seeing the big picture and understanding the many useful tricks that can make compensation a lot easier.

MASCoT is meant to aid students in analyzing a system and designing an effective compensator to improve the system's performance. The following document describes all the relevant aspects of my work, combining ideas from control system theory and computer science.

1.1 Issues in Compensator Design

The art of designing the best compensator given a control problem is not well-defined. There are no obvious paths that lead to the ultimate solution. Feedback circuit designers rely mainly on their experience and cleverness when compensating a system. Software that can aid a designer by coming up with compensation strategies that meet certain specifications is not generally available because this is a very challenging problem.

A rigorous model of a system usually turns out to be very complex and exhibits behavior that is difficult to predict. To make system analysis tractable, designers simplify systems and model them as *first-order* or as *second order* systems. Much of the success of a designer of feedback compensation is based on experience and pattern recognition. For example, certain performance characteristics of a system imply using one compensator type over another. Knowing what does not work in practice can save hours of trial-and-error attempts.

The approach to automatic compensation presented here is based on the *Bode Obstacle Course*, as described in section 3.6.1. Based on user specifications, *MASCoT* comes up with an asymptotic gain curve satisfying all or some of the specifications. The resulting system is then compared to the original system to determine the compensator. Of course, the behavior of the *MASCoT*'s solution might not be achievable by the original system. Some of the user specifications may only be met by trading off and not achieving others.

1.2 Availability of MASCoT

The only reasonably successful auto-compensation software tool currently available is the *Control System Toolbox* for *MATLAB* from MathWorks, Inc. However, the software is very expensive, and thus not widely available. In addition, the *Control System Toolbox* needs *MATLAB* libraries to run with and is platform specific.

MASCoT is written in Java, which allows it to run on any platform. It can run either as an application or as an applet in a browser. The software is free and intended for educational purposes.

MASCoT can be used as a compensating engine or as an evaluator of a compensator's performance. Students who are learning feedback compensation theory can experiment with *MASCoT* and get some intuition of what different methods of compensation are capable of achieving. Students who are already familiar with compensation can also test their designs by running *MASCoT*.

1.3 Scope of MASCoT

MASCoT works only with linear feedback systems. It is capable of designing series compensators, where the compensator is in the forward path in series with the controlled element. The toolkit currently does not support non minimum phase systems and nonlinear systems. However, it can be extended to support these types of systems as well as other forms of compensation. Potential extensions are discussed in chapter 10. This document covers the theoretical foundations both from the electrical engineering perspective as well as the computer science perspective. The *MASCoT* design is described in view of object-oriented programming and with an explanation of the implementation of the toolbox.

1.4 Other Work on Designing Feedback Compensators

The field of theoretical feedback compensator analysis and design has an extensive literature in the electrical engineering field. On the software side, however, only MathWorks, Inc. has been reasonably successful at creating an intelligent compensating agent which can aid a feedback system designer. Their *Control System Toolbox* even claims to be able to replace the human designer. Unfortunately, the toolbox is expensive and must be bought as an extension to the regular *Matlab Software Package*. Thus, I have never had the opportunity to examine the merits of their toolbox; the present work has not been compared against it.

1.5 Summary

The ultimate goal of this research is to come up with a software tool that can automatically design a feedback compensator to improve the dynamics of a given system. *MASCoT*, which stands for Model-based Automatic Software Compensation Toolkit, is meant to help a designer of feedback compensators. *MASCoT* can be used as an evaluator of a compensated system's performance or as a compensating engine. Due to a limited amount of time available, *MASCoT* handles only linear systems and only designs compensators which are in series and in the forward path with the controlled element.

1.6 Organization of the Thesis

- Chapter 1 gives an overview of the thesis document and the toolkit along with the rationale behind some of the design choices made.

- Chapter 2 provides theoretical foundations of my work within the signal and system control model literature. The reader is familiarized with different methods of system control and their benefits as well as some specialized terminology used later in the text.

- Chapter 3 describes the time and frequency domain properties of a linear system used to evaluate its performance. Subsequently, the chapter briefly explains a few popular analysis methods such as Bode plots, Nyquist D-Contour analysis, Nichols analysis, and the *Bode Obstacle Course*.
- Chapter 4 describes popular feedback compensation techniques and the relative advantages of each. It brings up some examples of popular applications of feedback in everyday life.
- Chapter 5 describes the overall design of the compensation tool from the higher level perspective. It shows the interconnections between the Graphical User Interface (GUI) modules and the *MASCoT* Engine (MEng) modules.
- Chapter 6 describes the Graphical User Interface and gives a brief User Manual for *MASCoT*.
- Chapter 7 outlines the implementation of the system and its properties in object-oriented domain. SystemModel and Properties class modules are described along with the interfaces they support. Next, the chapter covers the SpecTable module and Spec components which are implemented as separate classes.
- Chapter 8 describes the compensation engines' implementations and the attempted and implemented algorithms for compensation.
- Chapter 9 gives examples of different runs of the system.
- Chapter 10 presents possible future extensions to the work presented in this thesis.
- Chapter 11 is the conclusion for the work done. It provides an overview of the contributions and of the shortcomings of this thesis.

Chapter 2

Linear Feedback System Modeling

Make everything as simple as possible, but not simpler.

Albert Einstein

The art of automatic control of systems dominates every aspect of life in developed countries. It can be a source of order or destruction. Many feedback systems can be found that help us with our daily activities. An alarm clock makes sure that we are in full control of the amount of sleep we get. A thermostat ensures that the temperature in the room stays at some desired level. Hundreds of other control systems have increased the quality, production, and delivery of goods. Automatic systems greatly influence our current way of life and our expectation for the future. People are already able to travel to other planets thanks to control systems which allow them to leave the earth's atmosphere and keep them in the appropriate conditions while in space. People do not even realize that their whole lives evolve around using automatic systems [7], [22].

2.1 Introduction to Systems

A system is a combination of components that act together. Thus the word *system* could be used to describe any physical, biological, or organizational entity. studied, its behavior is represented through mathematical symbolism. A system is most commonly modeled by its behavior, both in the time and in the frequency domains. When it comes to the subject of *system control*, it turns out that knowledge of what resides within a system is not necessary in order to control it well. The important issue is being able to predict the system behavior for any range of inputs. Thus a designer studies a system by applying a variety of test signals to it and monitoring the system output. The resulting model describes the system's behavior rather than its internals.

Two types of system control are commonly encountered: *open-loop* and *closed-loop* system control. An *open-loop* control system is a system in which the output quantity has no effect upon the quantity of *actuating signal* supplied. A *closed-loop* control system is one where the output has an effect upon the input quantity.

2.2 Toaster Oven Example

2.2.1 Open-loop control

One good example of an open-loop system is a conventional toaster oven. The desired darkness of the bread can be controlled with a simple timer selection. The setting of the "darkness," or timer, represents the quantity of input, and the degree of darkness of the toast achieved represents the output quantity. How much the bread is toasted may vary depending on a variety of conditions, such as the type of bread, the humidity, and the temperature of the surrounds. The resulting darkness of the bread, however, does not have any effect on the input signal. The amount of input signal is preset and does not take into account the state of the output at any point in time.

A functional block diagram, as shown in Figure 2-1 allows for a symbolic representation of an *open-loop* system. In IEEE terminology provided in section 2.3, the desired darkness of the toast is the *command input* also called the *reference signal.* The *reference signal* is interpreted by the *controller*, a dynamic unit which performs the desired control function, in this case heating the toast.

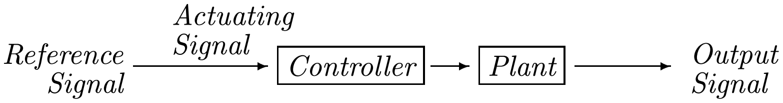


Figure 2-1: Block diagram of an open loop system.

2.2.2 Closed-loop control

We can also extend the toaster oven example to describe a closed-loop system. A human being can be added to obtain a *closed-loop* system from the *open-loop* system as described above. The person's task would be to observe the actual value of the output, in this case the resulting darkness of the toast, with respect to the *input command* that is the desired darkness of the toast. If necessary, the person can adjust the *controller* position to achieve the desired value. Addition of a person provides a means through which the output information is fed back, and the output is compared with the input. The person makes any necessary changes in order to cause the output to equal the desired value. The feedback action of the person controls the input to the *dynamic unit.* Systems in which the output has an effect on the input quantity are *closed-loop* control systems.

Feedback control theory is a whole science focusing on improving the performance of a system by enclosing it in a *closed-loop.* In modern feedback systems, the goal is to replace a person by a mechanical, electrical, or other form of an automated comparison unit. The functional block diagram of a *closed-loop* control system is shown in Figure 2-2. The *reference signal* is compared to the *feedback signal* and the difference between the two is the *actuating signal.* Thus the input and the output quantities are compared for inconsistency. The resulting signal applied is the amount necessary to obtain the desired output value.

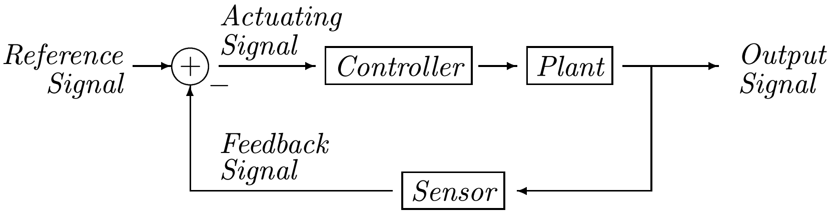


Figure 2-2: Block diagram of a closed loop system.

2.3 Terminology

From the preceding discussion, a few definitions have evolved which I briefly list below. They are based on some of the proposed standards of the IEEE [10]. The variations from the proposed standards are used to simplify some of the aspects of the subject matter. The descriptions are detailed enough for the purpose of my work.

Actuating Signal The signal that is the difference between the *Reference Signal* and the *Feedback Signal.* It actuates the control unit in order to maintain the output at a desired value.

Closed-loop Control System A system in which the output has an effect upon the input quantity in such a manner as to maintain the desired output value.

Command Input or Reference Signal The input signal to the system, which is independent of the output of the system.

Compensator or Controller A dynamic element that reacts to an *Actuating Signal* to produce a desired output. This unit does the main work for controlling the output. This element can be modified by the control systems designer. It is commonly implemented by a power amplifier.

Feedback Element The unit that provides the means for feeding back the output quantity or some appropriately scaled function of the output called *Feedback Signal*, in order to compare it with the *Reference Signal.*

Feedback Signal The signal from the feedback loop which is a quantity proportional to the output variable. This variable is subsequently compared to the *Reference Signal* to determine the *Actuating Signal.*

Open-loop Control System A system in which the output has no direct effect on the input signal.

Plant The hardware of process which is under control and is usually that part of the loop which is fixed in advance and constrained.

System A combination of components that act together. The words *systems* could be interpreted to include physical, biological, organizational, and other entities or a combination thereof.

2.4 Feedback System Model

A basic block diagram model shown in Figure 2-3 is used to study systems in the subsequent chapters. $G_c(s)$ is the *compensation function* or *controller*, $G_f(s)$ is the *plant* or fixed elements and $H(s)$ is the *feedback* or *measurement function.* The *input signal* or *command* is denoted by R, the *output* or *controlled variable* is C, and the *actuating signal* or *true error* is E. $L(s)$ is the *return ratio* or the *loop transmission* of a feedback loop, $L(s) = G_c(s)G_f(s)H(s)$.

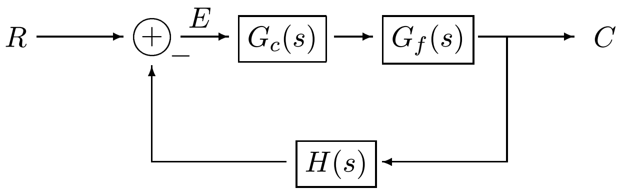


Figure 2-3: Functional block diagram of a closed-loop control system.

The feedback systems studied in this text are limited to linear elements and the compensator is assumed to be in the forward path. From this block diagram, we can derive the relationship between the output and the input signal by applying the *Black's formula*, as shown in equation 2.1. The function $\frac{1}{H(s)}$ represents the *ideal* behavior of a system when $|G_c(s)G_f(s)H(s)| >> 1$ and the difference between the command R and the output C is zero.

$$\frac{C(s)}{R(s)} = \frac{G_c(s)G_f(s)}{1 + G_c(s)G_f(s)H(s)} \tag{2.1}$$

In the chapters that follow, a *unity feedback loop* model, $H(s) = 1$, is used as a standard representation of the feedback systems studied.

2.5 Feedback benefits

Systems have many feedback loops built into them. A few biological systems can serve as examples. Such a seemingly small portion of a human body as a person's eye has many different feedback systems responsible for a wide range of functionality. A feedback system controls the amount of light entering the eye. The pupil of a person's eye gets larger or smaller depending on how bright the person's surroundings are. The adjustment allows the eye to focus properly.

Another example is focusing the eye on the objects that are either up close or at a distance. In order to focus on an object, the lens bends to adjust the clarity of the image. People wear glasses or contact lenses because they encounter problems with this feedback system. If its "dynamic range" narrows the system's error becomes too large to see clearly. A theoretical model of such a system would exhibit properties such as insufficient loop gain, small phase margin, or low bandwidth.

Feedback compensation has many followers for the following reasons:

1. Feedback makes a system less sensitive to changes in the gain and dynamics of certain elements in the loop. This often allows the system to maintain good performance even when element parameters vary substantially.

2. A feedback system can reduce the effects of certain types of disturbances, depending on where in the loop they happen to occur. This can be achieved due to the high-gain elements in the forward path, which amplify the input signal magnitude to be large enough in comparison to the magnitude of a disturbance.

3. Feedback can moderate the effects of nonlinearities by reducing the width of the ”deadzone” and the change in the gain as the output stage ”soft limits”. See [25, page 26], [7], [19] for more information.

Chapter 3

Properties of Linear Feedback Systems

Be good!

Raymond Paradis
Classical High School

In this chapter, I first define sets of properties which are used to evaluate the performance of a system qualified as either *first-order* or *second-order*. I then continue with methods of determining other properties of systems such as the *steady-state error*, the *DC gain*, and the *degree of stability*. The analyses are performed in the time and in the frequency domains.

3.1 Simplify, Simplify

Looking at the general model of a feedback, as depicted in Figure 2-3, one notices that the individual modules, which themselves are systems, are not described via a set of internal components. Instead, a designer works with a dynamic model of each system in the loop. The system model describes the behavior of the system for different loop or input conditions.

The design and analysis of a complex control system can become very tedious for a higher order system because of the many parameters it exhibits. However, years of practice and experimentation have taught engineers that many seemingly complex systems in reality have one or two poles which dominate their time or frequency behavior in the region of interest. This makes the designer's life simpler since *first-order* and *second-order* systems can be used to approximate the characteristics of more complex control systems. This approximation simplifies analysis of the system's performance in the time and in the frequency domains.

3.2 Time Domain Characteristics

The time domain evaluation of a system focuses on the properties of the output response of a system to input signals. Most properties have a time unit associated with their magnitude. When evaluating a system from the time domain perspective, we would like to know how fast the system responds to a given input, how quickly it can settle within a satisfying range on the output, or whether its signal response is volatile and exhibits significant overshoot or is sluggish.

3.2.1 First-order system model

When evaluating a system in the time domain, the concept of dominant modes is used to approximate the behavior of a system with many singularities[1] in terms of a *first-order* or of a *second-order* system.

Many thermal, mechanical, hydraulic, and pneumatic devices can be quite effectively modeled as first-order systems. These systems tend to have relatively slow time responses, which are dominated by the time constant of their mechanical components. The *time constant* determines how fast a system responds. The *time constant* of a system characterized as a *first-order* system translates into the location of the *dominant pole*. The *dominant pole* is the lowest frequency pole of the system. An example of a *first-order* system built using an integrator block is shown in Figure 3-1.

Some of the more important parameters of a first-order system are that:

1. The transfer function is

[1] Throughout this text, I use the word 'singularity' to denote either a system pole or a system zero. The correct standard is to use the word 'singularities' only when referring to system poles.

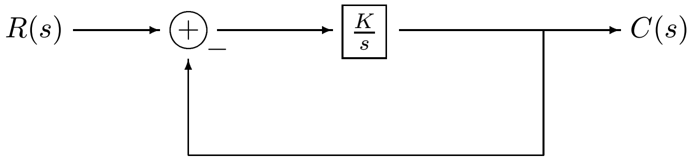


Figure 3-1: First order system implemented with an integrator.

$$\frac{C}{R} = \frac{1}{\tau s + 1} \tag{3.1}$$

2. Taking the inverse Laplace transform allows us to get the impulse response equation:

$$c(t) = \frac{1}{\tau} e^{\frac{-t}{\tau}} \quad \text{for} \quad r(t) = \delta(t) \tag{3.2}$$

3. The step response equation is:

$$c(t) = 1 - e^{\frac{-t}{\tau}} \quad \text{for} \quad r(t) = u(t) \tag{3.3}$$

   Figure 3-2 shows the step response in terms of the function parameters.

4. The time constant for a first-order system $\frac{1}{\tau s+1}$ is τ seconds.

5. The speed of response is proportional to $\frac{1}{\tau}$. Moving the dominant pole of the system away from the origin speeds up the system.

6. The settling time measures the time it takes for the output to settle within a given percentage of its final value. Two typical measures are:

   5% settling time= 3τ seconds

   2% settling time= 4τ seconds

3.2.2 Second-order system model

Unfortunately not all systems can be approximated as having only one pole in their transfer function. In real life systems oscillate, overshoot, or exhibit other characteristics which cannot be

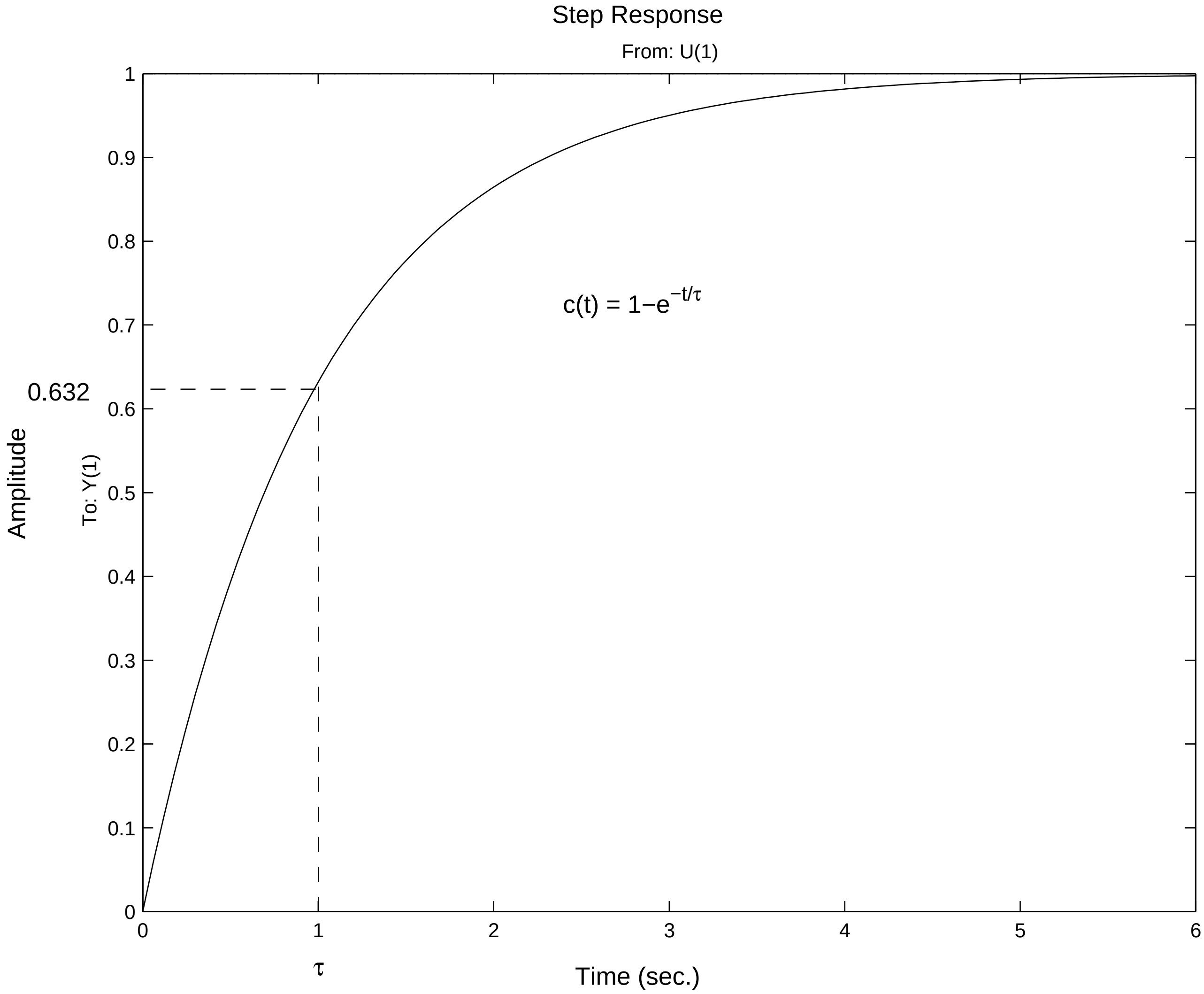


Figure 3-2: Step response of a first order system.

modeled with only one pole. Introduction of a second pole allows us to approximate this behavior more effectively. A simplified *second-order* system block diagram is shown in Figure 3-3.

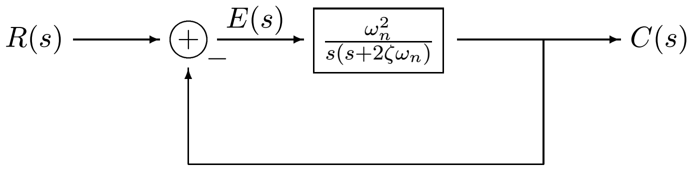


Figure 3-3: Generic block diagram of a second order system.

The general closed-loop transfer function for a second-order system is:

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{3.4}$$

The dynamics of a second-order system can be described in terms of two parameters, ζ and ω_n which stand for the *damping ratio* and the *natural frequency*, respectively. Depending on the value of ζ the system can be *underdamped* ($0 < \zeta < 1$), *critically damped* ($\zeta = 1$), or *overdamped* ($\zeta > 1$). Each one of these cases results in a different step response. The lower the value of the damping term, the longer the system will oscillate. Figure 3-4 shows the s-plane plot of the complex pole pair along with a few parameters.

The following parameters are used to characterize the step response of a system which can be classified as *second-order*::

1. The percentage overshoot is the maximum output value for a step input as compared to its final value.

$$P.O. = 100e^{\frac{-\pi}{\tan\theta}} \tag{3.5}$$

2. The time-to-peak is the time at which the peak of the step response occurs.

$$t_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} \tag{3.6}$$

3. The risetime of the response is defined as the time it takes for the output to go from 10% to 90% of its final value.

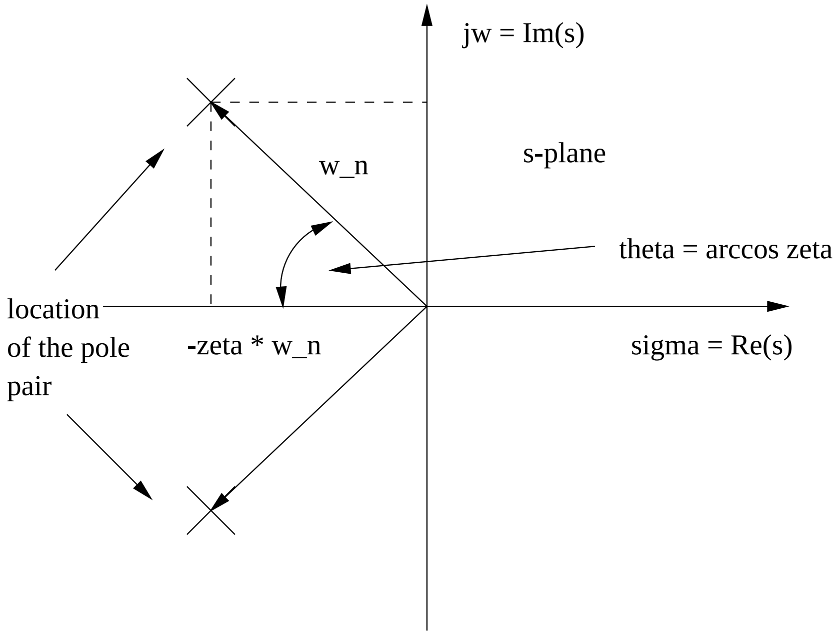


Figure 3-4: Complex pole pair on an s-plane plot.

$$t_r \simeq \frac{2.2}{\omega_h} \tag{3.7}$$

where ω_h is defined as the "half-power" or the $3dB$ frequency. It corresponds to the frequency, where the magnitude of the closed-loop system response is $\frac{1}{\sqrt{2}}$ (0.707) of its DC value.

4. The settling time is the time required for the step response to settle within some specified percentage of the final value. A commonly used bound is $+/-2\%$.

$$t_s \simeq \frac{4}{\zeta\omega_n} \tag{3.8}$$

Figure 3-5 shows these parameters.

3.2.3 Steady-state error & DC gain

Steady-state error is a time-domain measure of how well a feedback system's output tracks its command. In general, in order to track the input exactly, the number of integrators in the forward path must equal the order of the input function. The scope of my project is the first-order and the second-order systems, as shown in Figure 3-6. Thus, to achieve zero steady-state error, $E_{ss} = 0$,

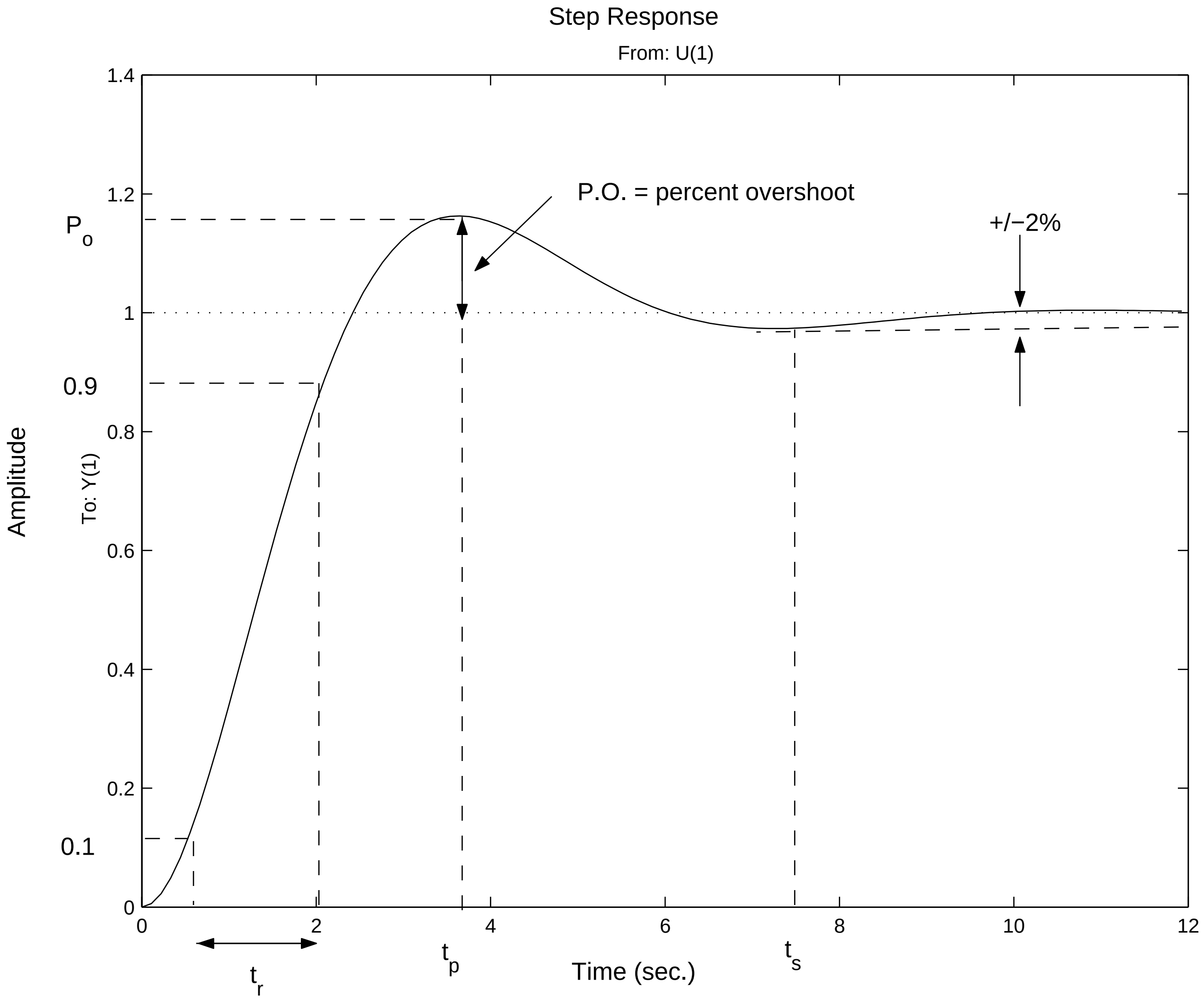


Figure 3-5: Parameters describing the step response of a system.

for a step input which is a first-order function, we need one integrator in the forward path, for a ramp we need two integrators, and so on. If the number of integrators is one less than the order of the input function we can expect a finite error, if it is two less we expect our steady-state error to grow to infinity.

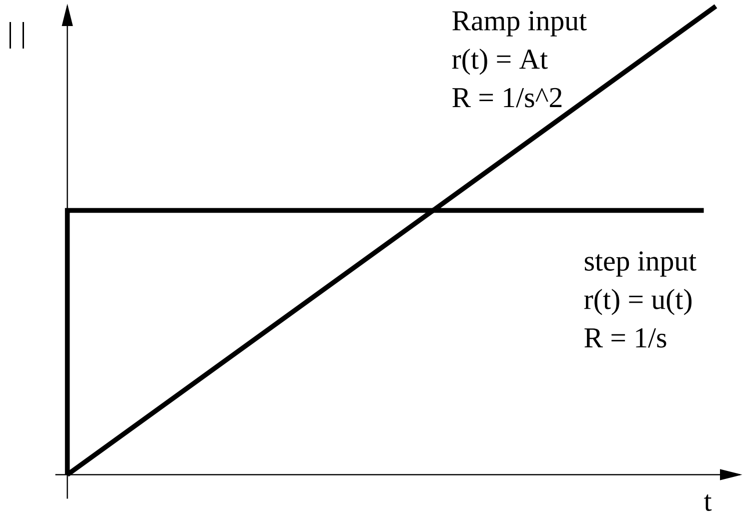


Figure 3-6: Step and ramp input functions.

For example, for the feedback system shown in Figure 3-1, the magnitude of the error in the finite error case can be calculated with the following formulas:

$$\lim_{s\to 0} s\frac{E}{R} * (R = step = \frac{1}{s}) = \lim_{s\to 0} s\frac{1}{s}\frac{s}{K+s} = 0 \tag{3.9}$$

$$\lim_{s\to 0} s\frac{E}{R} * (R = ramp = \frac{1}{s^2}) = \lim_{s\to 0} s\frac{1}{s^2}\frac{s}{K+s} = \frac{1}{K} \tag{3.10}$$

K is defined as the *gain* of the function or the *loop gain*. Thus, if we know the gain of the function and the number of poles at zero, we can immediately determine the steady-state error magnitudes.

3.3 Frequency Domain Characteristics

System performance can also be evaluated in the frequency domain. As in the time domain, first and second-order systems are frequently used to approximate and assess the performance of a given design. In the models below, I will assume that the gain equals one ($K = 1$) and that there are no zeros in the unity feedback closed-loop response, $\frac{C(s)}{R(s)}$.

Before I venture into the equations, I will briefly describe the parameters. Some of them are shown in Figure 3-7.

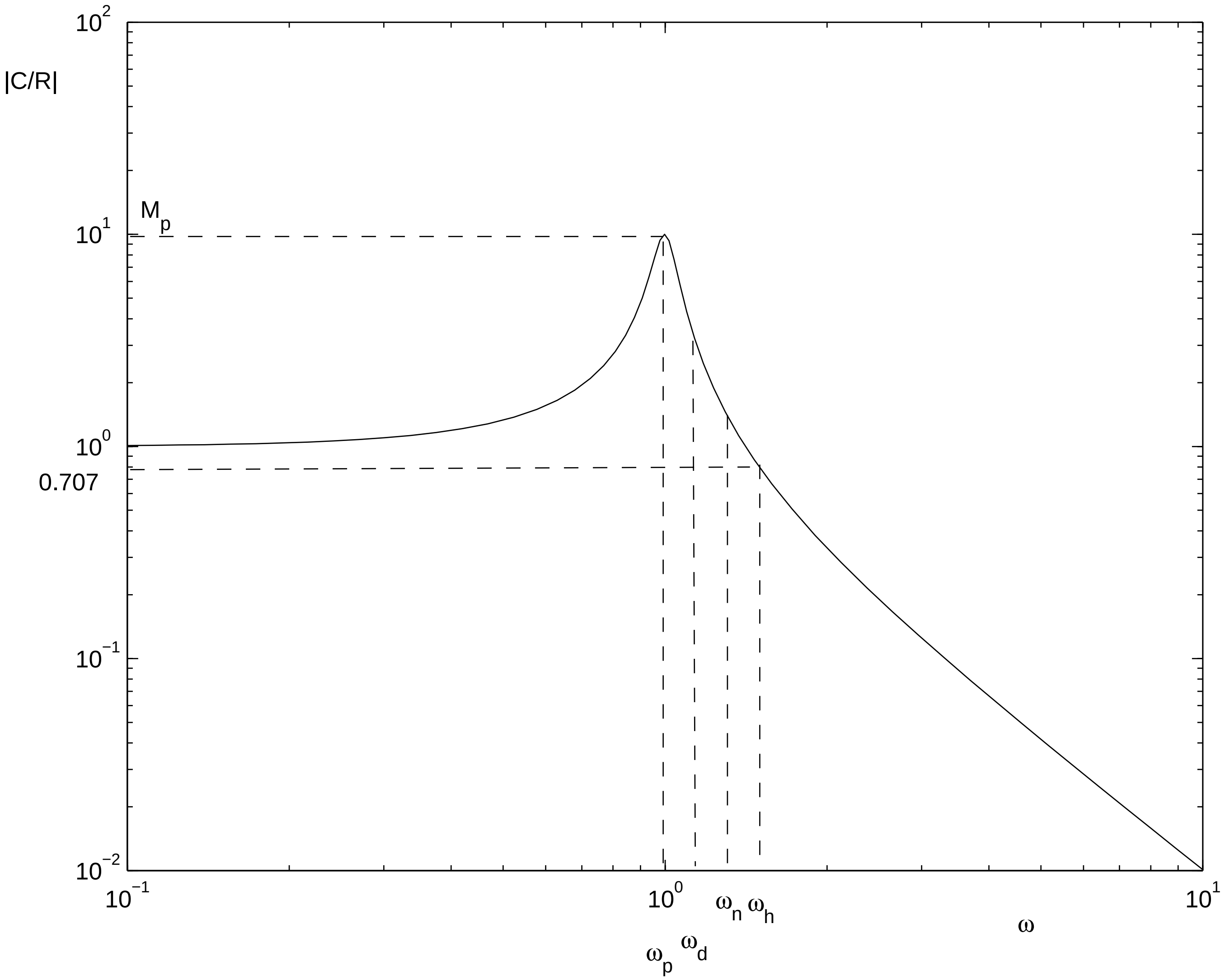


Figure 3-7: Parameters describing the frequency response of a system.

1. The magnitude ($|\frac{C(j\omega)}{R(j\omega)}|$) is the magnitude of the system response at frequency ω.

2. The phase angle (ϕ) is a function of frequency and relates the phase of a system.

3. The peak magnitude (M_p) is the maximum magnitude of the frequency response.
4. The peak or resonant frequency (ω_p or ω_R (rad/s)) is the frequency at which M_p occurs.
5. The natural frequency (ω_n) is the undamped natural frequency of the system.
6. The bandwidth or 3dB frequency (ω_h) is when the magnitude falls to $\frac{1}{\sqrt{2}}$ (0.707) of its DC value.

3.3.1 First & second order parameters

Below is a table listing some of the parameters for the first and second-order systems:

Parameter	First-order system	Second-order system
$\frac{C(s)}{R(s)}$	$\frac{K}{\tau s+1}$	$\frac{K\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$
Poles	$\frac{1}{\tau}$	$\omega_n(-\zeta \pm \sqrt{\zeta^2-1})$
Zeros	-	-
Gain	K	K
$\vert\vert$	$\frac{K}{\sqrt{\omega^2\tau^2+1}}$	$\frac{K\omega_n^2}{\sqrt{(\omega_n^2-\omega^2)^2+4\zeta^2\omega_n^2\omega^2}}$
ϕ	$-tan^{-1}\tau\omega$	$-tan^{-1}\frac{2\zeta\omega\omega_n}{\omega_n^2-\omega^2}$
M_p	-	$\frac{1}{2\zeta\sqrt{1-\zeta^2}}$
ω_p	-	$\omega_n\sqrt{1-2\zeta^2}$
ω_h	$\frac{1}{\tau}$	$\omega_n[1-2\zeta^2+\sqrt{2-2\zeta^2+4\zeta^4}]^{\frac{1}{2}}$
t_r	2.2τ	$\frac{2.2}{\omega_h}$
P_o	-	$e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}}+1$
$t_s(2\%)$	4τ	$\frac{4}{\zeta\omega_n}$
t_p	-	$\frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$

Figure 3-8: Properties of first and second-order systems

3.4 Methods of Analysis for Higher Order Systems

When a system can be approximated as first or second-order, checking whether the system meets a set of specifications is not difficult. However, if the system exhibits higher order behavior, a designer cannot easily determine whether it meets specifications. Deriving and solving the differential equation gives an accurate solution. However, following this approach is very tedious and difficult when a system is more complex and the modeling equations are of a higher order.

The goal of being able to predict a system's performance without having to solve the differential equations drove the development of such methods as the the *Routh criterion*, the *Root locus method*, the *Nyquist criterion*, and the *Nichols plot*. *Bode plots* are also very useful at graphing the system's response and can be used to determine stability. I will only provide a brief description of each one of these methods, as there is an extended amount of technical literature on this topic [25], [26], [13], [34].

3.4.1 The Routh criterion

The first thing that a designer would like to know about any system is whether or not it is stable. There is a relatively simple mathematical method called the *Routh criterion* to determine stability. The Routh test works with the characteristic equation, $1 + L(s) = 0$, and determines the number of roots with positive real parts. If this number equals zero, then the system is stable. For a more detailed description of this technique refer to [25], [34], [29], [7].

Although the Routh criterion can inform as to whether the system is stable or not, it does not indicate the degree of stability. Thus, we cannot predict the amount of *overshoot*, or the *settling time* of the controlled variable. Since the performance specifications usually indicate the prescribed limits on overshoot and the maximum time for the transients to die out, the Routh criterion is not sufficient for design purposes.

3.4.2 Bode

A *Bode plot* is used to graph the magnitude and phase of the *loop transmission*, $L(s)$. The asymptotic magnitude and phase are usually graphed on a log-log and a log-radian scale, respectively. The slope of the magnitude plot at a given frequency equals to the number of zeros minus the number of poles that appear below this frequency. Going from the lower frequency to the higher frequency, the slope is increased by one at every zero encountered and decreased by one at every pole encountered.

To find the exact magnitude and phase of the *loop transmission* at a given frequency ω the following formulae can be used:

$$L(s) = K * \frac{(a_0 s + 1)(a_1 s + 1)(a_2 s + 1)...}{(b_0 s + 1)(b_1 s + 1)(b_2 s + 1)\cdots} \tag{3.11}$$

$$|L(j\omega)| = K * \frac{\sqrt{(a_0\omega)^2+1}\sqrt{(a_1\omega)^2+1}\sqrt{(a_2\omega)^2+1}\cdots}{\sqrt{(b_0\omega)^2+1}\sqrt{(b_1\omega)^2+1}\sqrt{(b_2\omega)^2+1}\cdots} \tag{3.12}$$

$$\phi = tan^{-1}(a_0\omega) + tan^{-1}(a_1\omega) + tan^{-1}(a_2\omega) - tan^{-1}(b_0\omega) - tan^{-1}(b_1\omega) - \cdots \tag{3.13}$$

An example of a Bode magnitude and phase plot is shown in Figure 3-9.

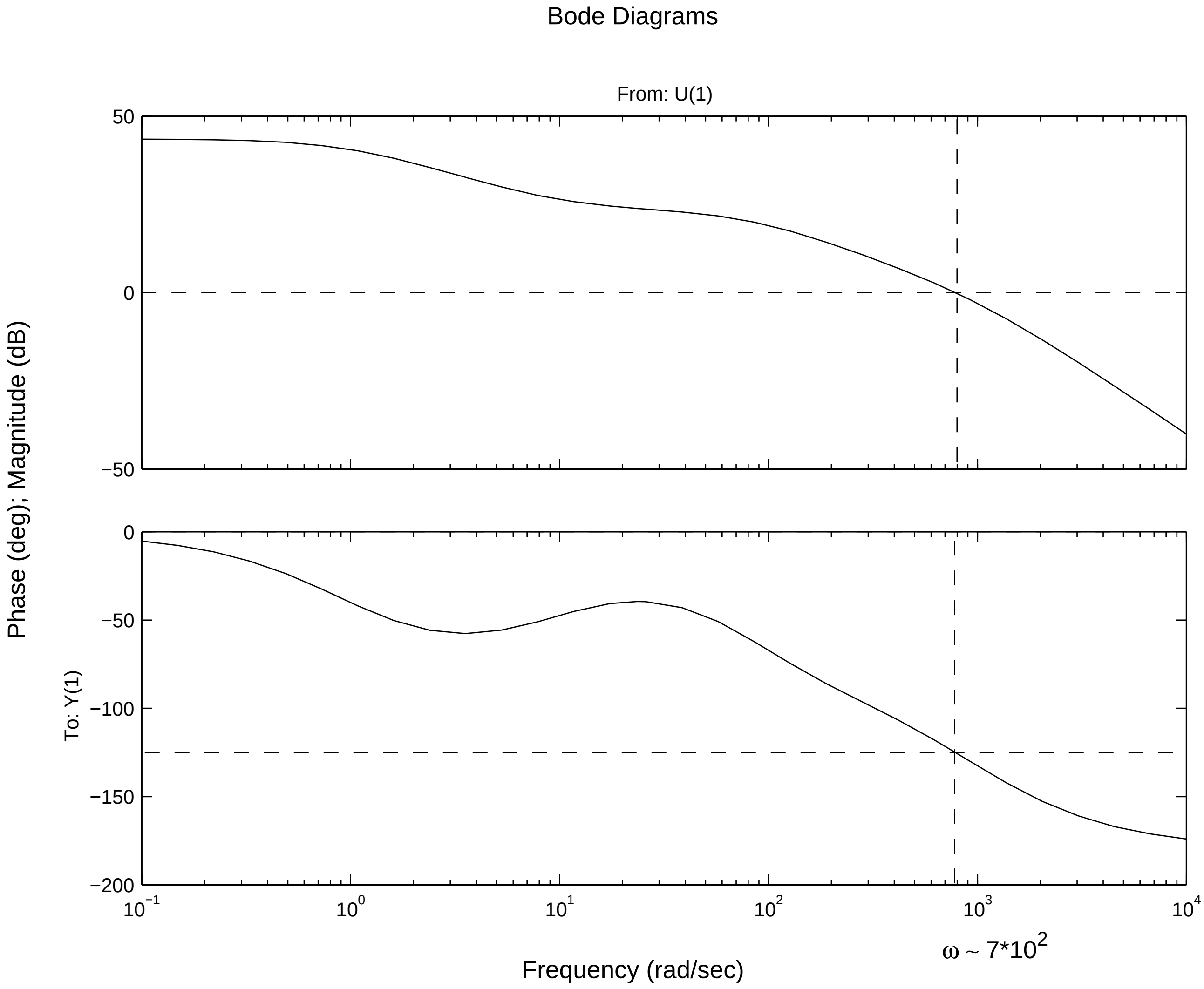


Figure 3-9: Example of a Bode magnitude and phase plot for $L(s) = \frac{10^6(s+10)(s+150)}{(s+1)(s+100)^2(s+1000)}$

3.4.3 More on analysis methods

More powerful analysis methods have been developed that can evaluate different aspects of the system performance. Three methods most commonly used by the feedback compensation designers are the Root Locus method, the Nyquist criterion, and the Nichols plot.

The **Root Locus** is a plot of the roots of the characteristic equation of the closed-loop system as a function of the gain as shown in Figure 3-10. For more information refer to [25], [12], [7], [34], [23].

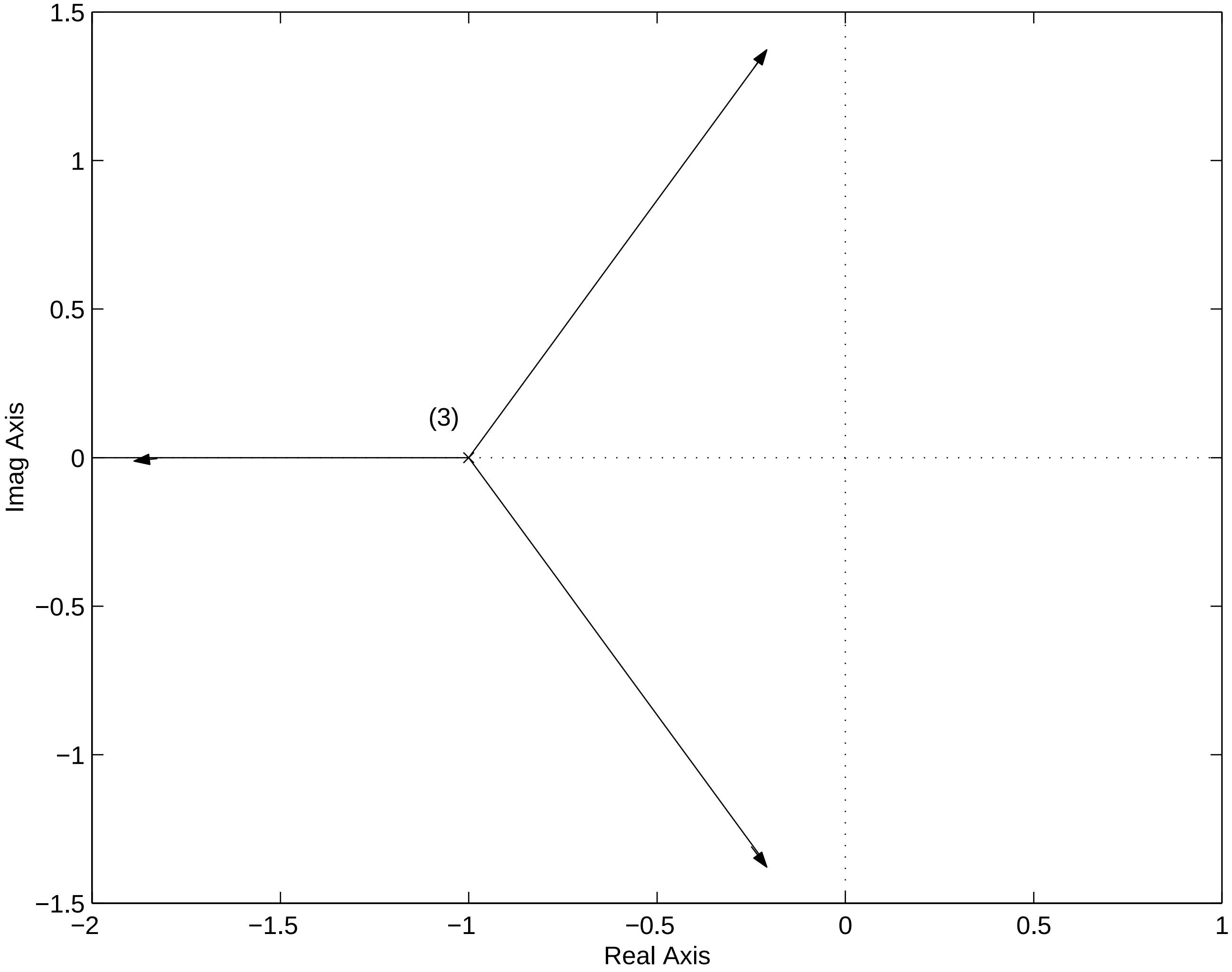


Figure 3-10: Example of a root locus plot for $L(s) = \frac{K}{(s+1)^3}$

The **Nyquist Criterion** is used to determined the stability of a system based on the mapping from the real-imaginary plane using *Nyquist D-contour* to a $\frac{L(s)}{K}$ plane. The details on this process can be found in [25], [23]. An example of a Nyquist plot is shown in Figure 3-11.

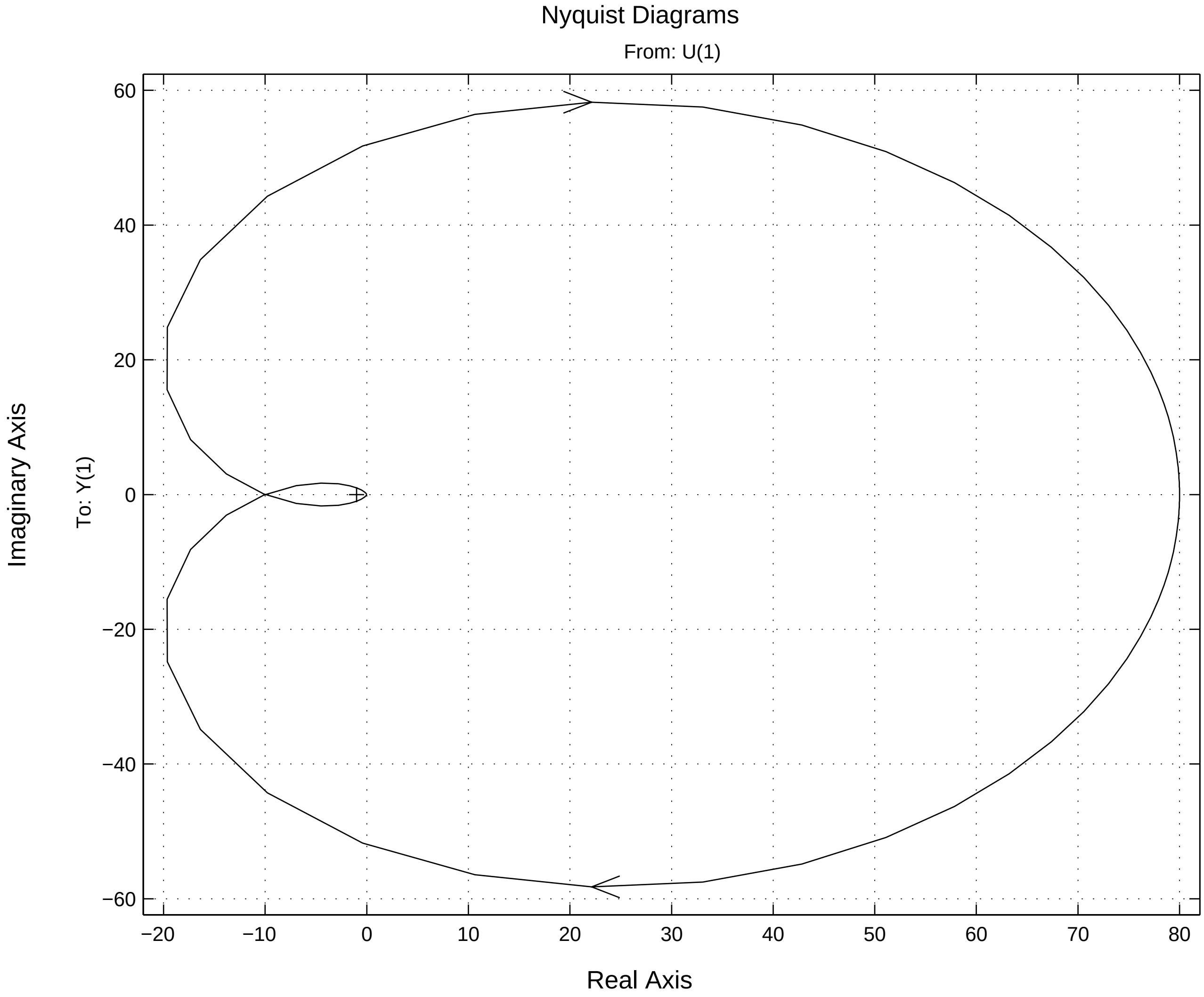


Figure 3-11: Example of a Nyquist diagram for $L(s) = \frac{K}{(s+1)^3}$

The **Nichols chart** is used to plot the magnitude and phase of the loop transmission and determine the value of *peak magnitude*, M_p. An example of a Nichols plot is shown in Figure 3-12. M_p determines the degree of the system stability. M_p can often be approximated as:

$$M_p \simeq \frac{1}{sin(p.m.)} \qquad (3.14)$$

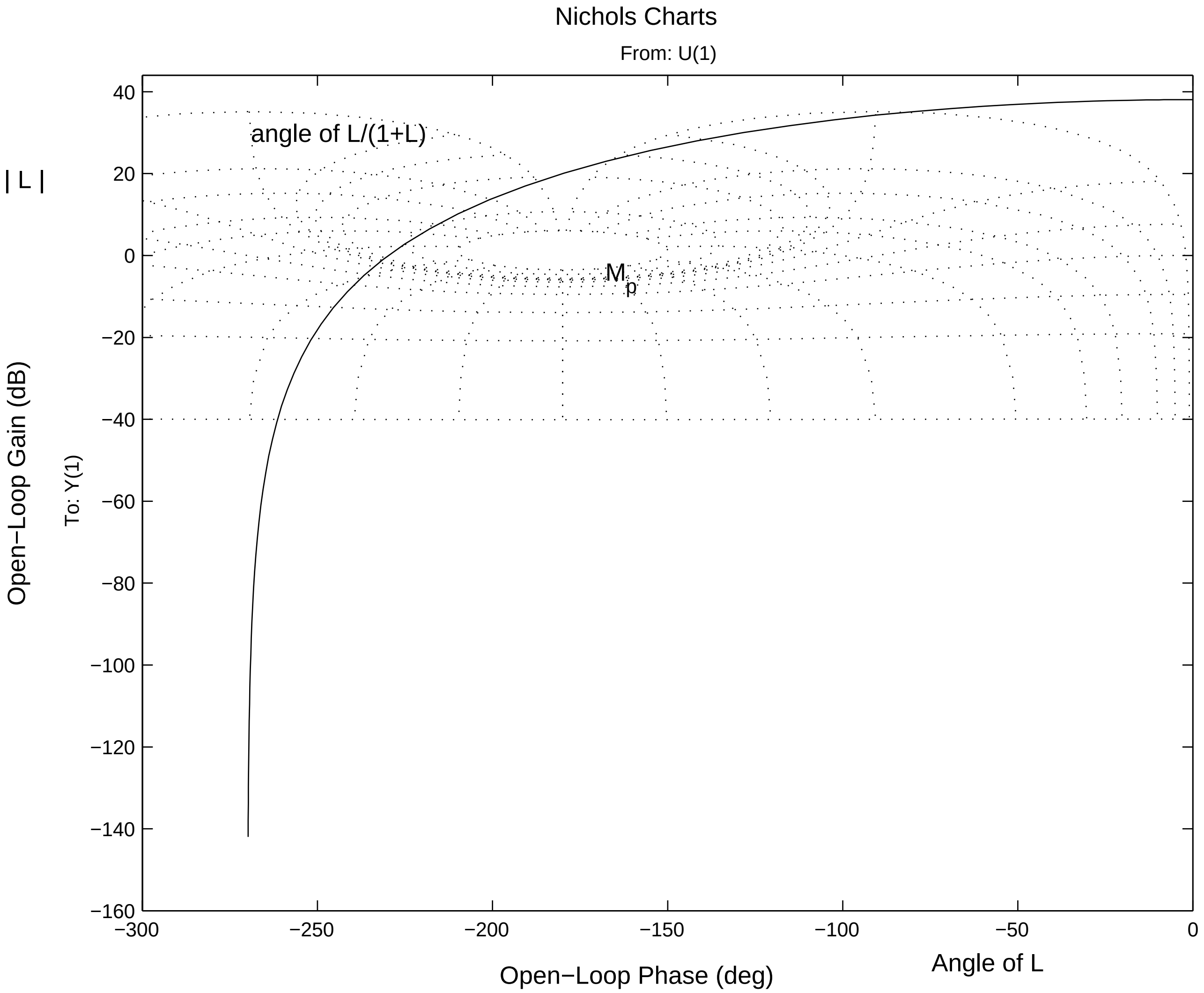


Figure 3-12: Example of a Nichols plot for $L(s) = \frac{80}{(s+1)^3}$

3.5 Stability

The most important requirement that must be met by any system is achieving stability. Thus, the stability measures are described in a separate section.

The performance of a feedback system is evaluated based on its accuracy, speed, and stability. Most of the time the design of a feedback system involves a tradeoff. Improving one of the system characteristics comes at a price of worsening another. To determine whether the system is stable or not, we examine the roots of the denominator of the closed-loop transfer function $\frac{C(s)}{R(s)}$, which are the factors of $1 + L(s)$, the denominator function of the closed loop system. All of these poles must be negative and lie in the left-half of the real-imaginary plane, as shown in Figure 3-13.

Stability of a feedback system is usually assessed by evaluating one or more of the *stability margins* listed below and shown in Figure 3.5.

1. The phase margin ($p.m. = 180^o - \phi$) is the phase difference between -180^o and the phase at the frequency when the magnitude of $L(j\omega)$ is unity. Phase margin is a positive quantity. Theoretically, if $p.m.$ is equal to zero, the system becomes unstable; in real life, in order to have a well-behaved system, the phase margin must be much larger than zero, depending on the application. For example, an industrial regulator needs 30^o of phase margin, a servo needs about 45^o of phase margin, and a feedback amplifier needs as much as $50 - 60^o$.

2. The gain margin ($g.m. = \frac{1}{|L(j\omega)|_{\phi(j\omega)=-180^o}}$) is one over the magnitude of the loop transmission when the phase of $L(j\omega)$ equals -180^o. If the magnitude of $L(j\omega)$ was increased by this factor, it would result in an unstable system.

3. The magnitude crossover frequency (ω_c) is the frequency when $|L(j\omega)| = 1$.

4. The phase crossover frequency (ω_ϕ) is when $\phi = -180^o$.

5. The peak magnitude of the closed-loop response can be used to estimate phase margin or vice-versa, $M_p \simeq \frac{1}{sin(p.m.)}$

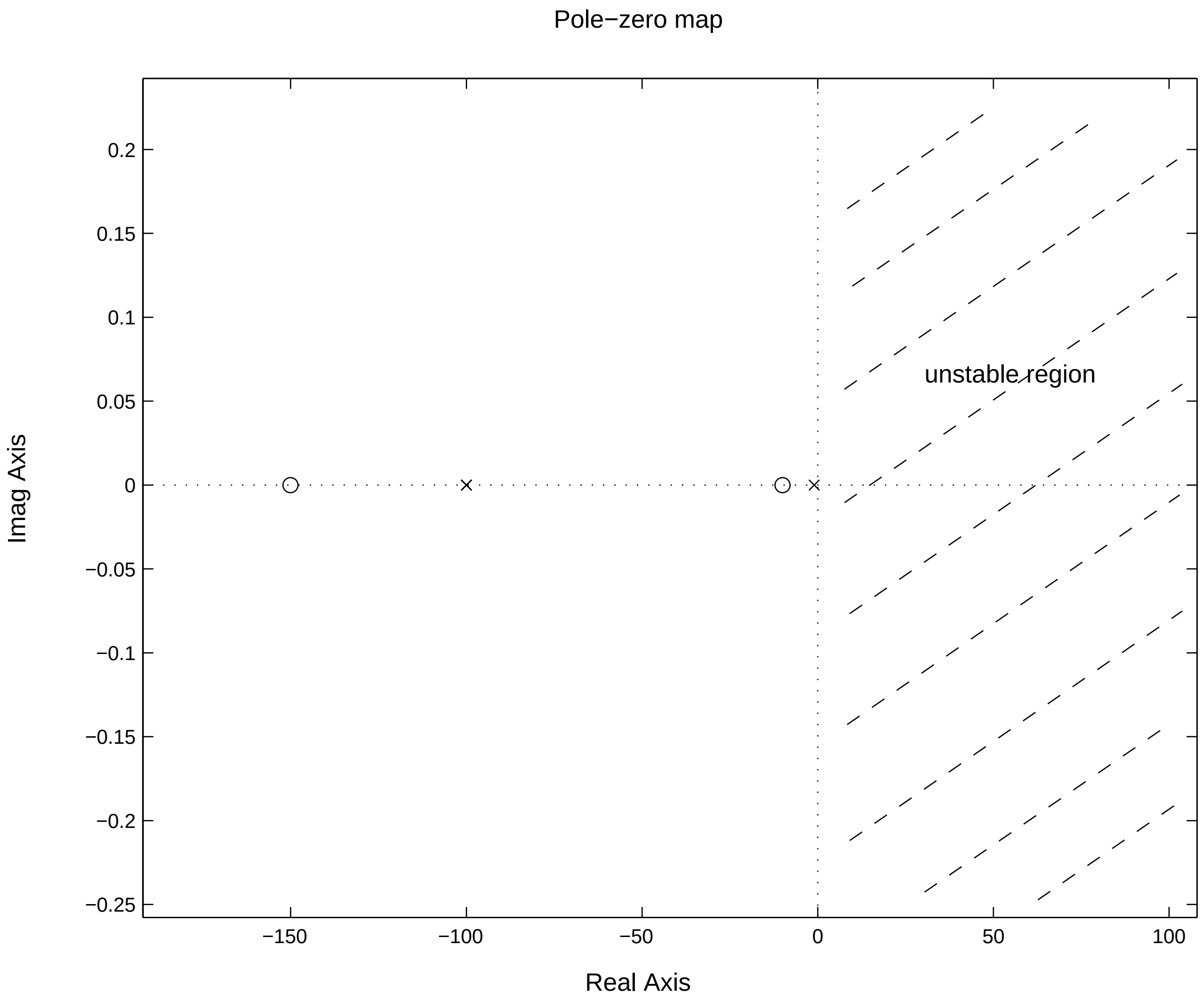


Figure 3-13: Example of a pole-zero plot for $L(s) = \frac{10^6(s+10)(s+150)}{(s+1)(s+100)^2}$

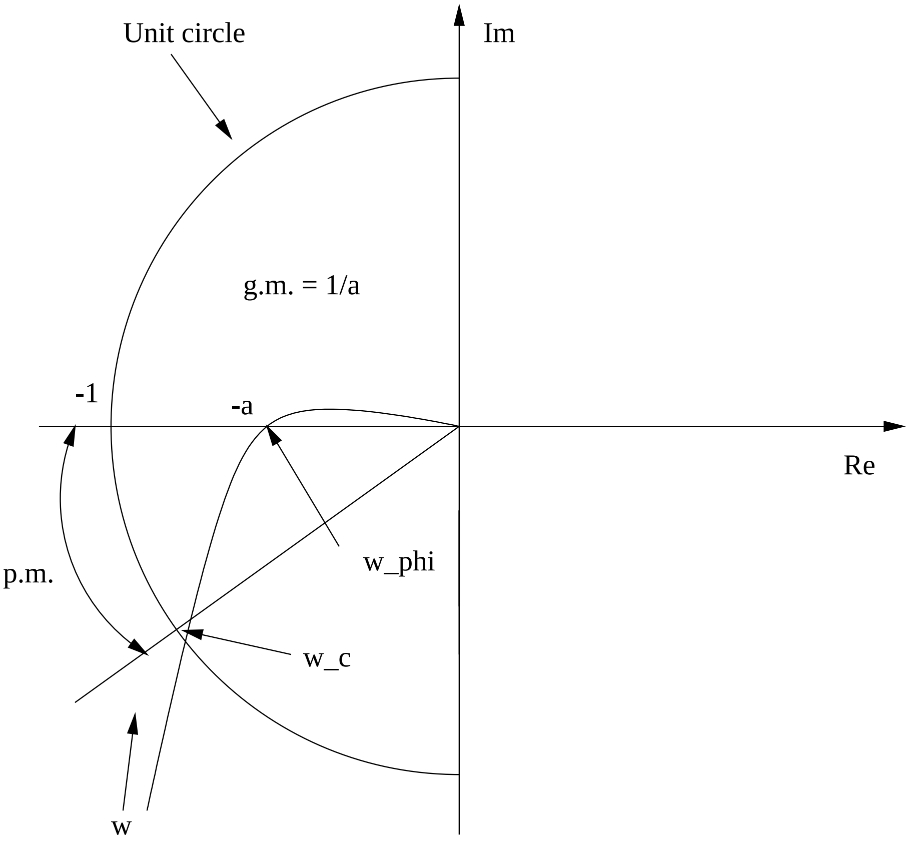


Figure 3-14: Phase and gain stability margins shown on a Nyquist plot.

3.6 User Specifications

Many of the system properties described in the preceding sections are specifications that a potential user of a system provides to the designer. These specifications are provided in the time, or in the frequency domain, or in both. Some of the most often encountered specifications are **crossover frequency, bandwidth, phase margin, steady-state errors (step or ramp input), time-to-rise, and peak magnitude**. Many of the specifications from the time domain can be translated into the ones in the frequency domain and vice versa.

3.6.1 The *Bode Obstacle Course*

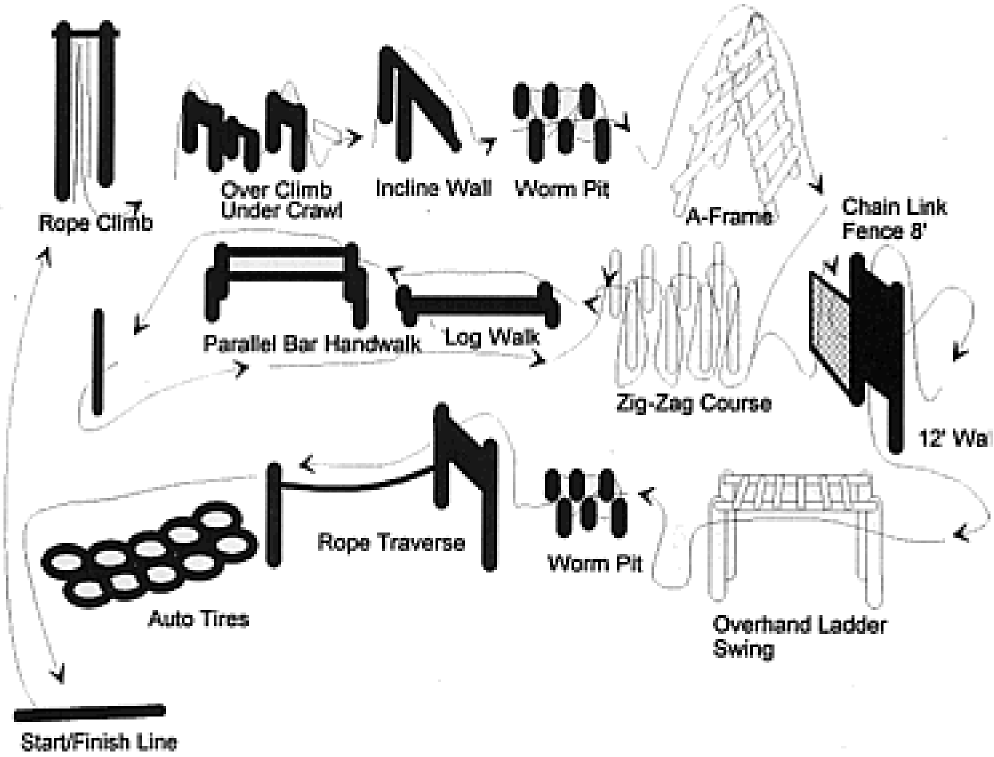


Figure 3-15: Bode Obstacle Course...?

The Bode Obstacle Course is a method that allows us to come up with an asymptotic model of the system's magnitude based on a set of frequency domain specifications. We assume unity feedback and limit ourselves to monotonically decreasing magnitudes of $L(s)$. For every constraint, a possible behavior of the loop transmission is determined by coming up with a line than spans some range of frequencies suggested by the constraint. After interpreting all specifications, a piecewise linear model for the magnitude is determined by combining linear solutions across all frequencies. The points where the lines meet are the locations of the poles and of the zeros of the loop transmission. Once the system that meets the specifications is found, it is compared to the original one in order to determine the compensator required. Assuming that $G_f(s)$ is minimum phase, the function of the compensator is determined from the following relation:

$$G_c(s) = \frac{L(s)}{G_f(s)} \tag{3.15}$$

The following is a set of specifications interpreted with the *Bode Obstacle Course* method:

Crossover behavior Given a desired ω_c, L(s) can be approximated as $\frac{\omega_c}{s}$, $\frac{\omega_c^2}{s^2}$, or $\frac{\sqrt{2}}{\frac{s}{\omega_c}(1+\frac{s}{\omega_c})}$

Steady-state error When the steady-state error limit for a given input signal is specified, it implies the number of integrators in the forward path and the magnitude of the DC gain. Section 3.2.3 covered the relationship between the order of the input signal and the number of integrators in the forward path.

If the steady-state error for an input of order n is specified to be a constant γ, then at low frequencies:

$L(s) \simeq \frac{K}{s^n}$ where $K \geq \frac{1}{\gamma}$

Low frequency obstacles In order to track commands well and reject disturbances, the error for some low frequency ranges, specified by $0 < \omega < \omega_l$, should remain small, less than ϵ. Since at low frequencies we expect the magnitude of L(s) to be big, thus $E/R \simeq 1/L(s)$. This implies that $|L(0 < \omega < \omega_l)| \geq \frac{1}{\epsilon}$.

High frequency obstacles For good noise rejection, the magnitude of loop transmission should be low enough as to filter out most of the high frequency signals. Thus, the performance specification might require that $|C/R(\omega_k)| \leq \kappa$. Since at high frequencies the magnitude of loop transmission is expected to be small thus $C/R \simeq L(s)$. This implies that the magnitude of $L(\omega_k) \leq \kappa$.

The graphical realization of the Bode performance specs is shown in figure 3.6.1.

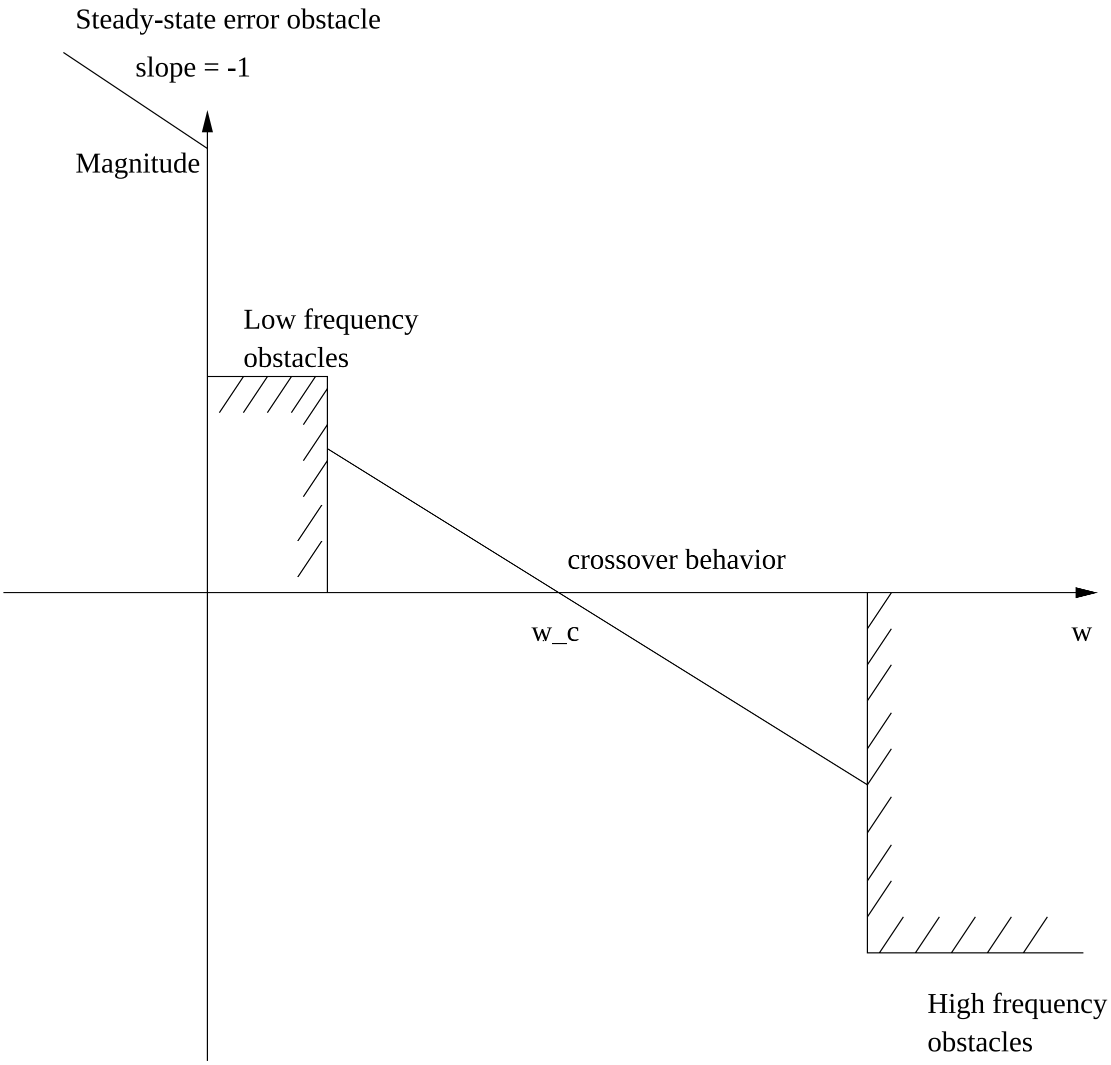


Figure 3-16: *Bode Obstacle Course*

Chapter 4

Compensation Theory

Do, or do not. There is no 'try'.

Yoda, "The Empire Strikes Back"

The action of walking from an initial location to a destination point along a predetermined path satisfies the definition of a feedback control system. In Figure 4-1, the prescribed path is the *reference input* signal. The eyes of the person walking act as a *comparator*, monitoring the difference between the actual and the prescribed path. The prescribed path is the desired output. If the person moves in a direction which takes him or her off the prescribed path, the eyes transmit a signal to the brain. The brain amplifies the input signal and transmits its output signal to the legs, in order to correct the actual path of movement and to bring it in line with the prescribed path. As we can see based on the definition of a feedback system, feedback control mechanisms have existed since the beginning of time [7], [22].

4.1 Why Compensate?

The preceding chapters focused on methods of characterizing a system by analyzing its performance against a given set of parameters. However, as the world is not always the way we would like it to be, the system does not always behave exactly in a way its user would like it to behave. The

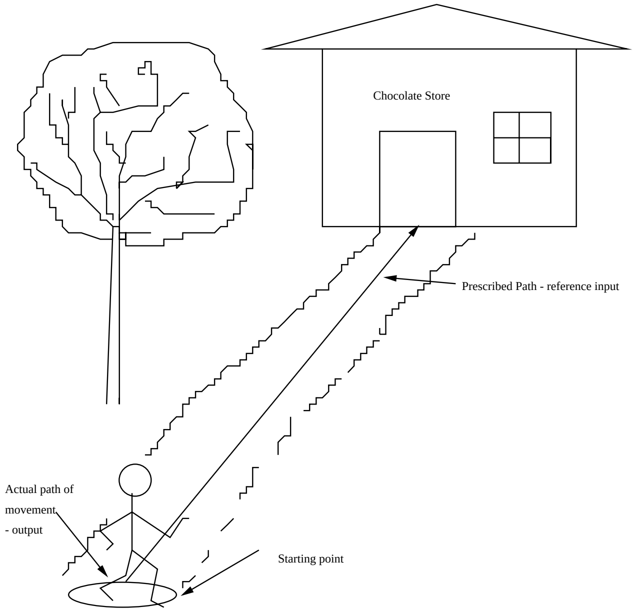


Figure 4-1: Human being acting as a feedback system.

system can have stability problems and have its output oscillate. It can alternately be too slow, or fast with a large steady-state error. A potential user of a system has a set of expectations from it. What if the system as it stands cannot meet these expectations? In such case, the designer would like to change the properties of the system so as to fulfill these expectations. These expectations become specifications in the design process.

Based on a set of performance requirements either in the time or in the frequency domains, the designer comes up with an approximate model of a system. Then the designer tries to reshape the model of the original system such that it resembles the one which meets the requirements. Sometimes it is impossible to come up with a system model meeting all the requirements. Other times it is not possible to change the loop transmission characteristics enough so as to obtain the desired system from the original one. In such cases, the designer must rank the significance of desired characteristics and decide how to most effectively trade off the desired properties.

4.2 Basic Compensation Theory

The process of introducing additional equipment into a system to change its time and frequency characteristics in order to improve system performance is called *compensation*, or *stabilization*. A properly compensated system is stable, has a satisfactory transient behavior and a large enough gain to ensure that the steady-state error does not exceed the specified maximum. The compensation devices might be electrical networks, mechanical equipment, or chemical solutions. The compensator may be placed in any of the following locations:

- In series with the forward transfer function (called *series* or *cascade* compensation).
- In the feedback path (called *parallel* or *feedback* compensation).

The selection of the location for inserting the compensator depends on the control system, the physical modifications that are necessary, and the desired results. In my thesis document, I focus on the series compensation, since the software tool developed is limited to the series compensation of a system. Brief discussion of feedback compensation will follow in section 4.4.

4.3 Series Compensation Methods

Compensation is not a well-defined science consisting of a clearly defined series of steps. It is usually a trial-and-error procedure where the designer plays a major role in the outcome. Finding a good compensator is a puzzle with many solutions or no solution. The trick to finding the most appropriate one is often to know the approximate answer already. For a good designer, it is a pattern recognition problem; the set of specifications implies the compensation method to be used as well as a set of related tricks. Sometimes the specifications do not work together, such as when satisfying one specification excludes the possibility of satisfying another. In such a case, a good designer evaluates the situation and ranks the importance of different specifications to make appropriate tradeoffs.

In the following discussion, I work with the simplified model of a feedback system as shown in Figure 4-2.

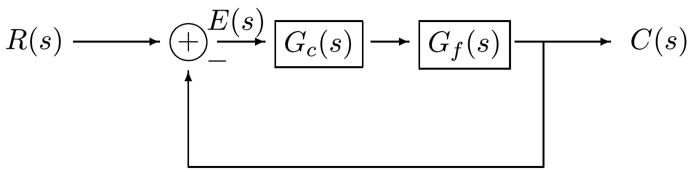


Figure 4-2: Simplified block diagram of a feedback system.

The symbols are:

$G_c(s)$ - Compensator

$G_f(s)$ - Plant or system to be compensated

$R(s)$ - input signal or reference point

$C(s)$ - desired output signal

$E(s)$ - error

4.3.1 Proportional compensation

Analysis of the Bode plots and evaluation of a system's performance based on the magnitude and phase of its loop transmission implies that the degree of stability is strongly correlated with the amount of phase margin at the *crossover*, when $|L(j\omega)| = 1$. If phase is monotonically decreasing, a conceptually straightforward modification to achieve a higher phase margin and thus stability would be to lower the gain of the loop transmission. By lowering the magnitude of $G_c = K$ we can push down the magnitude plot and achieve a crossover at a lower frequency. The phase plot does not change, and since the phase of most systems is monotonically decreasing, we can achieve a higher phase margin and thus stability. Figure 4-3 shows the effects of lowering the gain K on the magnitude and the resulting improvement in the phase margin.

However, a crossover at a lower frequency means a decrease in the bandwidth of the system and thus a slower response. Also, since the gain of the system often directly affects the magnitude of the steady state error, when the DC gain is decreased, the error increases. Thus, improving stability often involves certain tradeoffs such as an increase in the error magnitudes or a slower response.

Proportional compensation might be sufficient for some applications, such as servomechanisms where the dynamics are generally dominated by mechanical components with bandwidths of less than 100 hertz. A portion of the DC loop transmission of a servomechanism is often provided by an electronic amplifier, and these amplifiers can provide frequency independent gain into the megahertz range. Changing the amplifier gain leaves the dynamics of the loop transmission virtually unaltered [25, page 166].

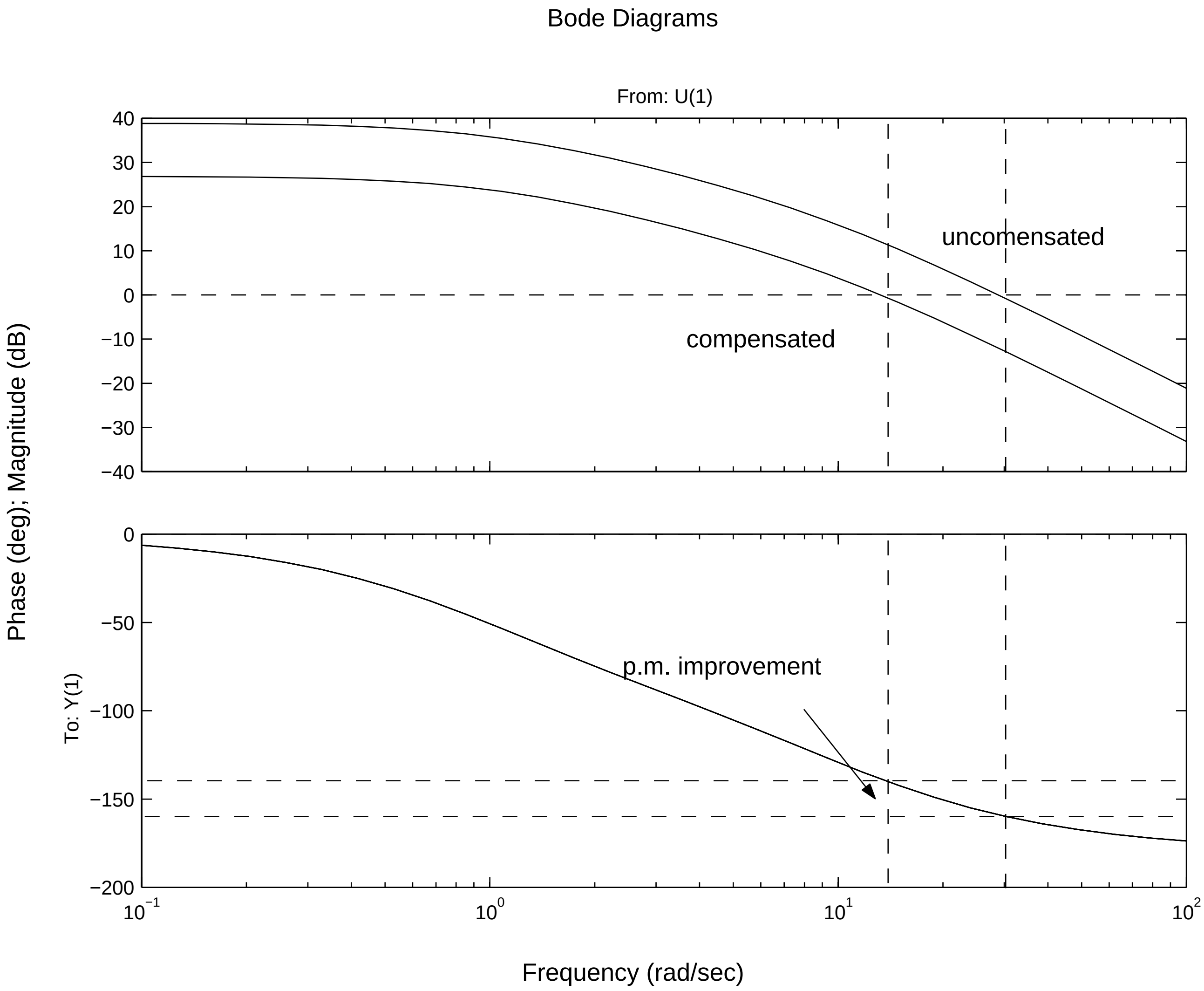


Figure 4-3: Proportional compensation can increase the phase margin but decreases bandwidth. Original system, $L(s) = \frac{88}{(s+1)(0.1s+1)}$, is compensated by decreasing the DC-magnitude by 4. Thus the compensated loop transmission is $L(s) = \frac{22}{(s+1)(0.1s+1)}$

4.3.2 Dominant pole compensation

The use of *proportional* control often severely limits a system's performance. *Dominant pole compensation* is an attempt to retain the DC characteristics of a system while keeping it stable. It is accomplished by making $G_c(s)G_f(s)$ a single low-frequency pole combined with appropriate gain. Now the open-loop transfer function looks like a single pole in the vicinity of crossover. A single-pole loop transmission results in a phase margin of 90^o, ensuring that a system is stable for any amount of negative feedback. Because retaining stability in such a system is very straightforward, many types of compensation reduce to make one pole dominate the loop transmission. However, once again, such compensation comes at the price of reduced bandwidth, since the dominating pole is at a lower frequency than the other poles in the system. Figure 4-4 shows the effects of applying a single pole in order to compensate a system.

Regulators There is an important class of systems for which introducing a dominant pole is the ideal thing to do. Such systems, called *regulators*, are designed to hold the value of their output constant independent of disturbances. The output of a regulator is not meant to, and thus does not need to, track a rapidly changing and fluctuating input. Having a dominant pole at the output allows for successful disturbance rejection. Since the dominant pole appears at a low frequency, the system is slower and slowly adjusts its output value when necessary.

4.3.3 Lag compensation

Another compensation method is making $G_c(s)$ a *lag* type transfer function which introduces a zero and a pole in the function of the compensator. The name "lag" comes from the fact that when looking at the frequency plot, starting at the lower frequencies on the left and going to the higher frequencies on the right, the compensator's zero comes after the pole. Thus the zero "lags" after the pole.

$$G_c(s) = a_o \frac{\tau s + 1}{\alpha \tau s + 1} \tag{4.1}$$

Benefits of lag compensation

One important feature of this network is that it allows us to increase the low-frequency open-loop gain while maintaining the phase margin. Another useful characteristic of this compensating net-

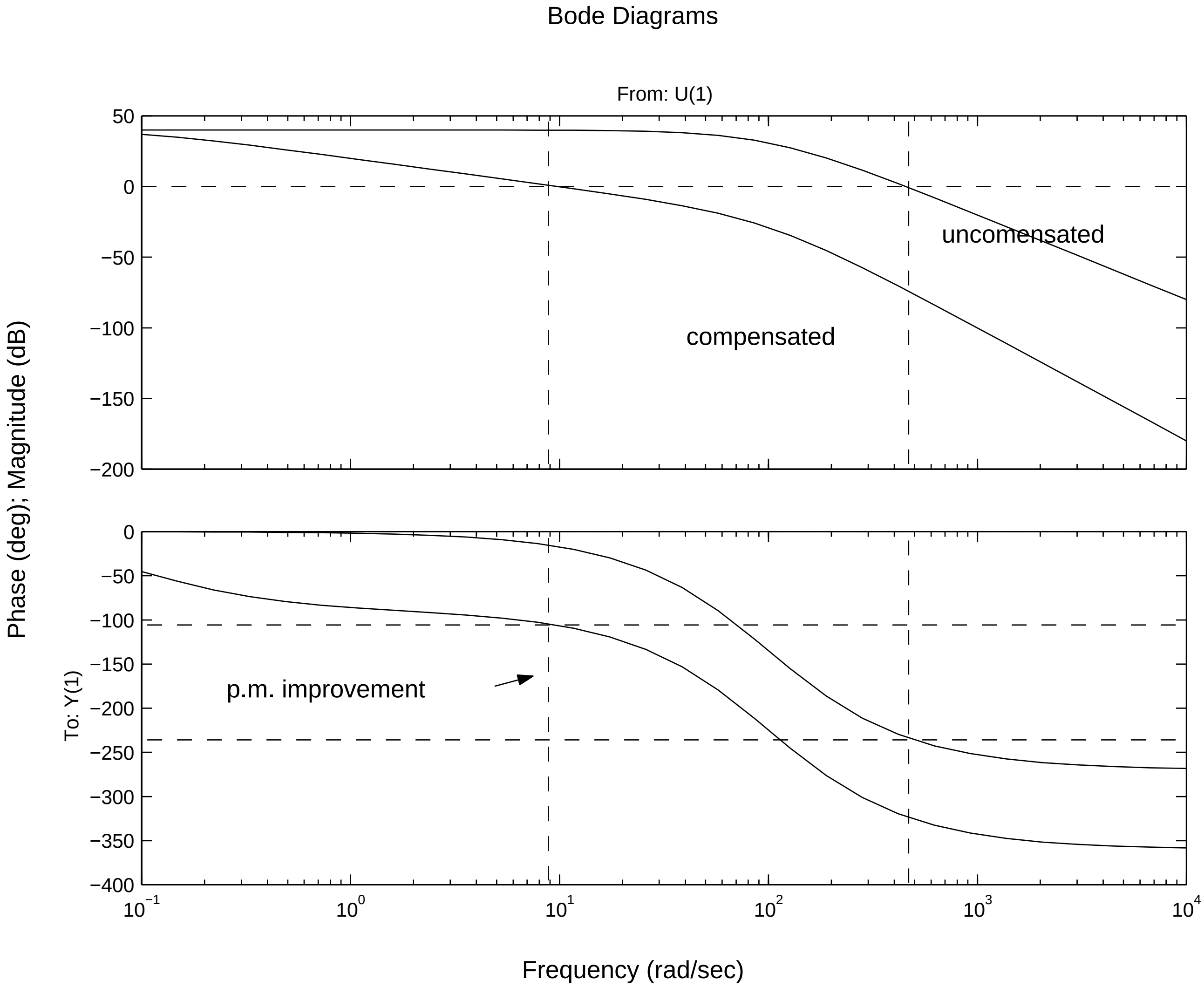


Figure 4-4: Dominant pole compensation improves stability with a single pole placed at a frequency well-below the other poles. Original system, $L(s) = \frac{100}{(0.01s+1)(0.01s+1)(0.01s+1)}$, is compensated by a dominant pole. Thus the compensated loop transmission is $L(s) = \frac{100}{(0.01s+1)(0.01s+1)(0.01s+1)(10s+1)}$

work is that the pole occurs before the zero. If combined with another system, the *lag* compensator causes the magnitude to drop faster in the region between the pole and the zero. Thus, the net magnitude of the loop transmission at the frequencies beyond the pole-zero pair is lowered. This means that the high frequency noise is rejected more successfully. Good noise rejection implies a more robust behavior.

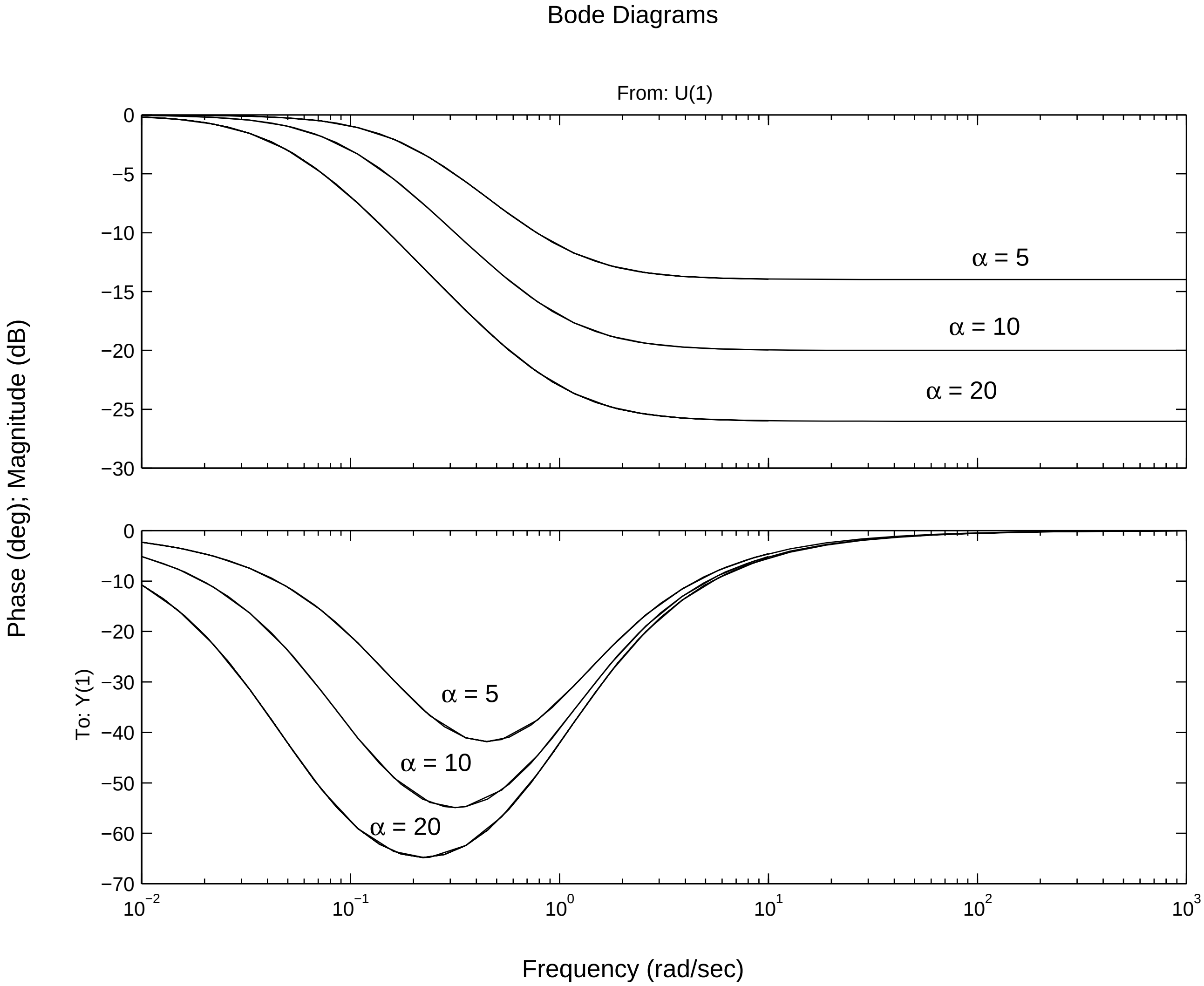


Figure 4-5: Bode phase and magnitude plots for a lag network where $\frac{V_{out}}{V_{in}} = \frac{\tau s+1}{\alpha\tau s+1}, \quad \tau = 1$

The phase plot of a *lag* network as a function of α is shown in Figure 4-5. The inverted bump in the phase of the lag network is an undesired side effect. Placing the *lag* network well below the crossover is intended to reduce the residual negative phase shift from the lag function at crossover to almost zero, thus retaining the same stability level. The net negative phase shift is due to the pole appearing before the zero. The maximum phase shift magnitude due to this network is dependent on α, which relates the geometric distance between the singularities, as shown in equation 4.2.

$$\theta_{max} = \sin^{-1}(\frac{\alpha - 1}{\alpha + 1}) \tag{4.2}$$

$$G_c(s) = a_o \frac{\tau s + 1}{\alpha \tau s + 1} \tag{4.3}$$

The lag network needs to fit somewhere between the crossover and the low-frequency region of operation. The fact that it should be as much below the crossover as possible is easy to conclude from the phase plots. Why then not place the pole-zero network at the lowest frequency possible? A more detailed observation of the time response of such a system indicates that a pole-zero doublet at a low frequency causes an undesirable *long tail transient* as pictured in Figure 4-6. Placing the pole-zero doublet at a low frequency is equivalent to a large time constant, τ, associated with the pole. This type of behavior can deteriorate the step response of a system. Thus lag compensators should be designed with caution, keeping in mind the tradeoff between achieving desensitivity and having to deal with a long settling time.

The lag compensator does nothing to improve the speed of response of a system since it cannot push the crossover out. If anything, it may lower the crossover. However, it allows us to keep the gain high at low frequencies, which decreases the error magnitudes while keeping the system insensitive to high frequency noise. In some cases, it can help us keep the gain high at some higher frequencies as well. One extreme example is placing the pole at the origin. In this case, we have eliminated the steady-state error for a step input. Now the error equals zero. This type of compensator is frequently realized via an operational amplifier and is called *proportional plus integral*, in short PI, compensator.

4.3.4 Lead compensation

Another approach to compensation is adding some positive phase in the vicinity of the crossover. Positive phase can be obtained with a *lead* compensator, where the zero "leads" before the pole, resulting in a net positive phase. The general form of a lead compensator is:

$$G_c(s) = \frac{1}{\alpha} \frac{\alpha \tau s + 1}{\tau s + 1} \tag{4.4}$$

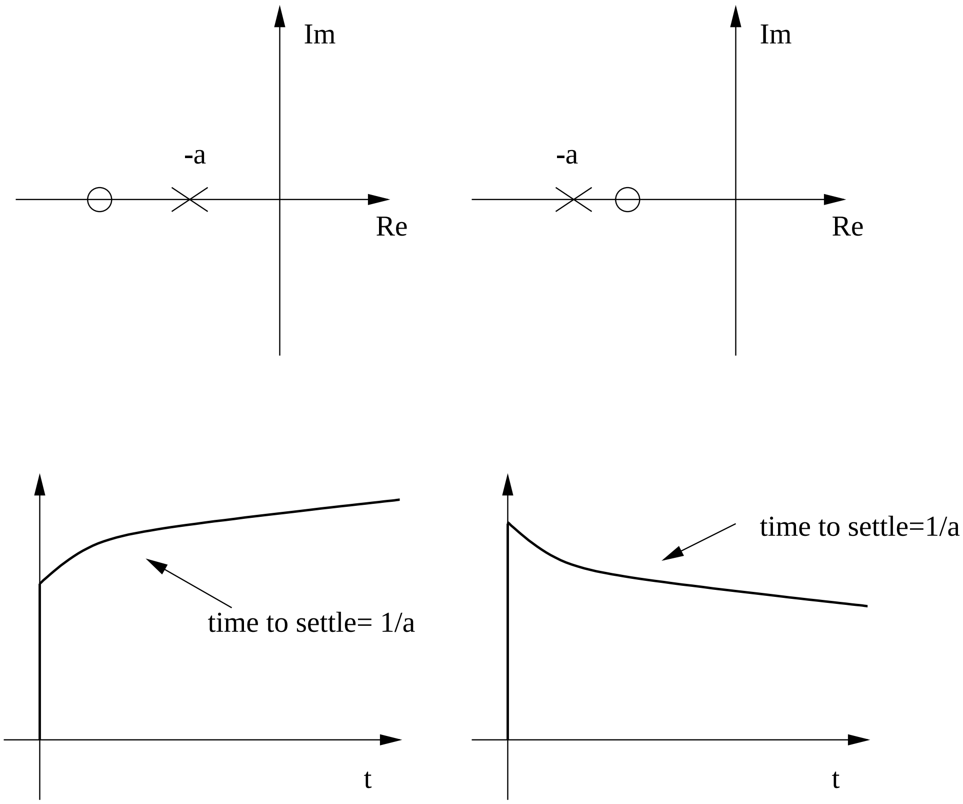


Figure 4-6: Pole-zero doublets and their “long-tail” step responses.

The ability to add some positive phase can be advantageous in two ways:

1. The phase margin at the crossover is increased, leading to a more stable system.
2. The crossover frequency can be pushed further out without a decrease in the phase margin, thus speeding up the time response of a system.

An example of using lead compensation is shown in the graph below, figure 4-7.

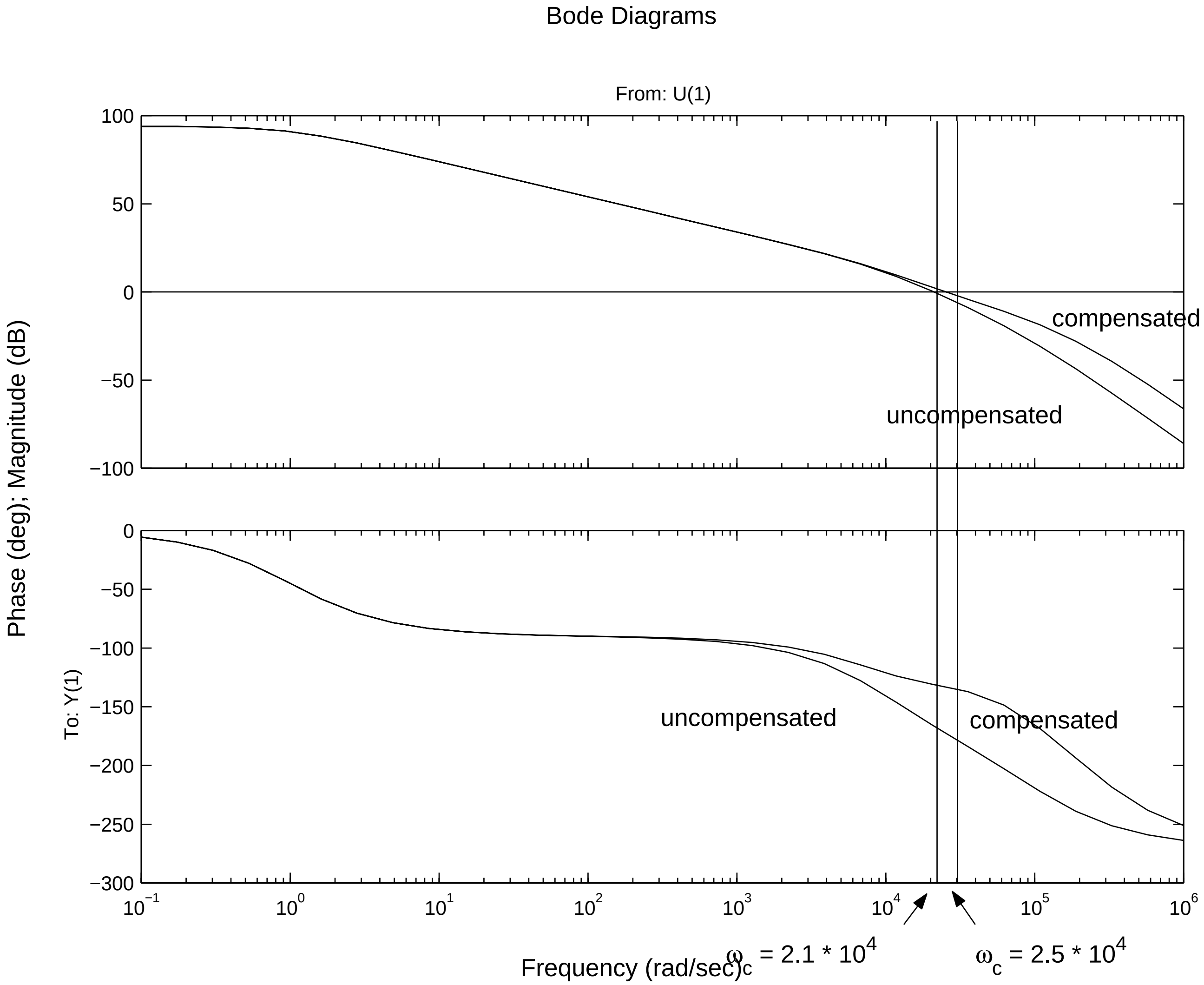


Figure 4-7: Original system: $L(s) = \frac{5*10^4}{(s+1)(10^{-4}s+1)(10^{-5}s+1)}$, system compensated with lead: $L(s) = \frac{5*10^4(4*10^{-5}s+1)}{(s+1)(10^{-4}s+1)(10^{-5}s+1)(4*10^{-6}s+1)}$

The maximum allowed phase shift at the crossover for the lead compensated system to remain stable is the same as in the lag compensator case, greater than -180^o, and is calculated with equation 3.13.

The singularities of the lead compensator are usually placed around the intended crossover frequency. The highest positive phase from the *lead* compensator is obtained at the geometric mean of its zero and its pole locations. In order to take a full advantage of this feature, the geometric mean is where the crossover often ends up. Another place for the crossover is the location of the zero from the lead network. If the pole is far enough from the zero, approximately 45^o of positive phase cab be obtained without any significant magnitude increase.

The lead network phase plot is commonly referred to as a *phase bump.* The relative size of the phase bump is shown as a function of α in Figure 4-8. In theory, the height of the phase bump is limited to $+90^o$. In practice, the phase bump tends to be no higher than $+60^o$ degrees.

Why not take a full advantage of all the 90^o available? Since the phase is entirely dependent on α, if we make α a large number, we can get almost 90^o." However, as Freud once said, "There is no such thing as a free lunch," and we certainly would not get the positive phase without any side effects. The value of α determines the relative ratio between the zero and the pole. The Bode plots of the magnitude and the phase as a function of α of the lead network in Figure 4-8 show that while the phase behavior is very desirable, the magnitude does not behave ideally. Instead of decreasing, it increases monotonically in the region between the two singularities. So the farther they are from each other, the higher the net magnitude. This particular "feature" is not desirable since it means greater noise sensitivity. Instead of attenuating the high frequency noise, the lead network allows more noise to pass through. In practice, designers choose α to be around 10.

4.3.5 Lead-lag compensation

Once a designer has mastered the tricks of the trade and understands which compensator does the best job in a given situation, he realizes that he can try to combine the best of both worlds. Sometimes a design requires both a gain increase and a bandwidth increase. The compensators I have discussed so far do not individually provide the designer with a wide range of improvements. Thus, a more complicated compensator is needed.

Since *lead* and *lag* compensation are usually implemented within the different regions of the frequency domain, they can be made to work together. Lag compensation occurs well below the crossover, while lead networks surrounds the unity-gain frequency. *Lead-lag* compensation, which

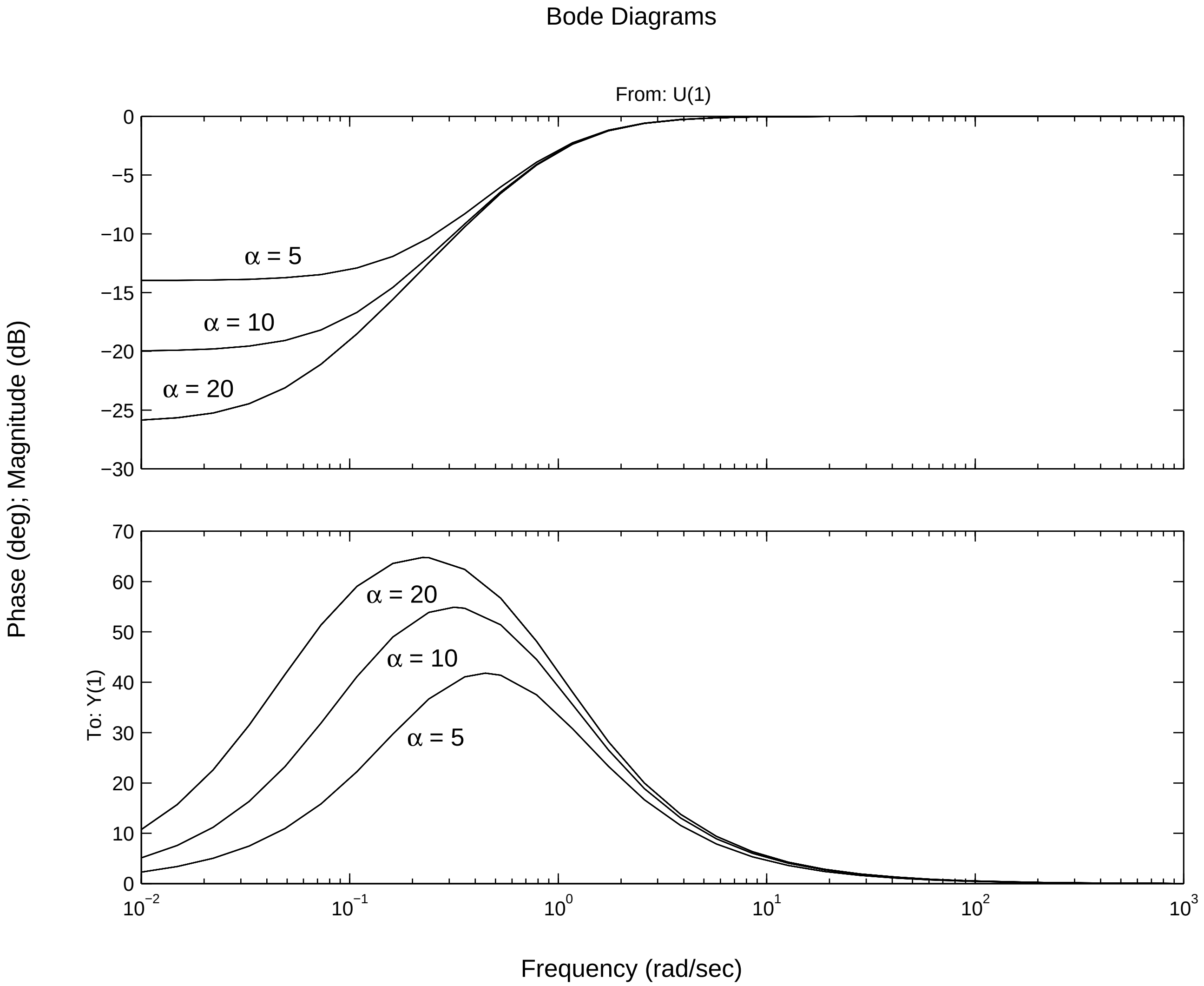


Figure 4-8: Bode phase and magnitude plots for a lead network where $\frac{V_{out}}{V_{in}} = \frac{1}{\tau}\frac{\alpha\tau s+1}{\tau s+1}, \quad \tau = 1$

is a combination of both, allows the designer to simultaneously improve the D-C behavior of a system, increase bandwidth, and improve stability.

So why not use lead-lag all the time? The reason why designers do not always go for the lead-lag compensator is that much of the time there is no need to. The simpler the design which meets the specifications, the better. A fancier compensator is more difficult to analyze, and complexity often comes at a price of unexpected behavior. In addition, if we design a fancy compensator, it tends to need more parts to build it. It is fine to build a couple of these. However when it comes to production on a large scale, the savings associated with a simpler compensator can be very substantial.

4.3.6 Summary of series compensation methods

There are no generalized rules concerning compensation, since the right approach is dependent on many factors, such as the behavior of fixed elements in the loop, anticipated inputs, desired performance, and even the economic aspects of implementing a given compensator. After reading numerous works describing various approaches to compensation, table 4-9 below is an attempt to give a general summary of the most important features of each compensator as seen in [25].

Type	Special Consideration	Advantages	Disadvantages
Reduced $a_o f_o$		Simplicity.	Lowest desensitivity.
Create dominant pole	Lower the frequency of the existing dominant pole if possible. Locate at the output of a regulator.	Can improve noise immunity of system. Usually the type of choice of a regulator.	Lowers bandwidth.
Lag	Locate well below the crossover frequency.	Better desensitivity than either of the above.	May add undesirable "tail" to transient response.
Lead	Locate zero near crossover frequency.	Greatest desensitivity. Lowest error coefficients. Fastest transient response.	Increases sensitivity to noise. Cannot be used with fixed elements that contribute excessive negative phase shift.

Figure 4-9: Comparison of series-compensating methods

4.4 Minor Loop Compensation

4.4.1 Feedback compensation

The preceding section covered the application of *series* or *cascade* compensation in improving the performance of a given feedback control system. It is often possible to achieve the same or even better results with *feedback* or *parallel* compensation, where the compensator is in the feedback path.

In the preceding discussion, we assumed $H(s) = 1$, which indicates unity feedback. However, $H(s)$ can be of the same form as the compensators used for series compensation, such as *lead, lag,* and other forms. Parallel compensation is implemented by adding an element in the feedback path around a controlled variable. This often results in a two-loop system, as we can see in Figure 4-10. The inner loop is wrapped around the controlled element and is called the *minor loop.* In most cases this loop is a lot faster than the outer loop. The outer loop is commonly referred to as the *major loop.* In section 4.4.2, I provide some mathematical modeling of the effects of parallel compensation.

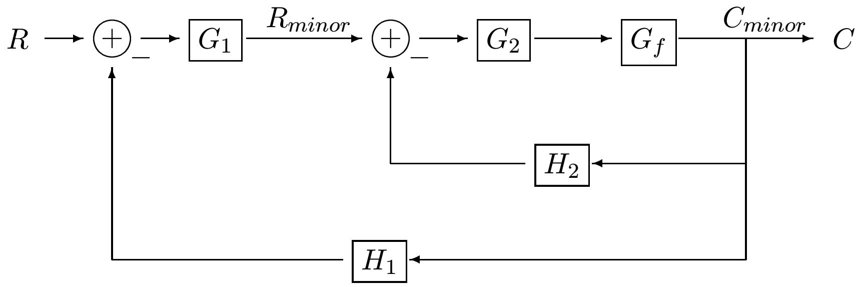


Figure 4-10: Block diagram of minor loop compensation.

Designing an effective feedback compensator is not as straightforward as designing a series compensator. Feedback loop dynamics are confusing to analyze and predict. The traditional methods often do not work. The design procedures change radically. For example, the open-loop poles from the feedback element become the zeros of the closed-loop system. Many designers who feel comfortable designing series compensators do not understand the subtle mechanics of the feedback compensation and thus do not use it. Sometimes because of the physical constraints of the system, there is no practical feedback compensator that can be used.

Often the environmental conditions in which the feedback system resides require it to retain a high degree of accuracy and stability, even when the outside conditions change. One example is an airplane, which can be subjected to rapid changes in altitude and temperature. Regardless of these conditions it must retain the same functional behavior. *Parallel compensation* can stabilize a system across much wider range of environmental conditions than *series compensation.*

Using parallel compensation can alleviate the effects of noise. Sometimes an amplifier in the forward path accentuates the noise, which is an undesirable side effect. Putting a compensator in the feedback path can enable the designer to attenuate this high frequency noise.

Improving the speed of response of a system is often encountered as a design requirement. While series compensation can offer methods to improve the time response, parallel compensation can offer improvement in cases where the series compensation cannot help. Sometimes the dynamics of a feedback system vary widely in the different parts of the system. Then the isolation of loop components is necessary. This isolation can be achieved by introducing an inner feedback loop around the portion to be isolated. For a more detailed description of these practices see [25], [7], [14], citefranklin.

MASCoT is capable of evaluating and designing series compensators, but does not handle minor loop compensation. Adding an additional software engine which implements minor loop compensation would be a very useful improvement for the next *MASCoT* version, see chapter 10 for more information.

4.4.2 Model equations

Some of the effects of the minor loop feedback can be seen by examining the inner and the outer loop equations derived from the block diagram in Figure 4-10. For the minor loop in Figure 4-10 we have:

$$\frac{C_{minor}}{R_{minor}} = \frac{G_2 G_f}{1 + G_2 G_f H_2} \tag{4.5}$$

where the *return ratio* $= G_2 G_f H_2$

If the magnitude of the loop transmission is large, that is, $|L(s)| \gg 1$, then we can approximate the minor closed-loop equation as:

$$\frac{C_{minor}}{R_{minor}} = \frac{1}{H_2} \tag{4.6}$$

For the major loop in Figure 4-10 the closed-loop equation is:

$$\frac{C}{R} = \frac{G_1 \frac{1}{H_2}}{1 + G_1 \frac{1}{H_2} H_1} \tag{4.7}$$

where the *loop transmission* $= G_1 \frac{1}{H_2} H_1$

Thus, with a loop transmission of a significantly large magnitude we can reduce the effect of the changes of the the plant and other elements in the forward path. The closed-loop behavior depends entirely on the feedback element.

4.4.3 Tachometer feedback example

A practical example of minor loop compensation is the use of tachometers in servomechanism systems to track both the velocity as well as the position of an object. A differentiator $K_t s$ is used as the feedback element in the minor loop. The output R is a measure of position. The differentiator, $K_t s$, differentiates R with respect to time and output velocity. The output velocity is then compared with the command velocity. With a sufficiently high gain of the loop transmission, the minor closed loop equation can be approximated as a single pole integrator $\frac{1}{K_t s}$.

Chapter 5

MASCoT Design

I hear and I forget. I see and I remember. I do and I understand.

Confucius

MASCoT is a software toolkit that can evaluate and compensate linear non minimum phase systems using series compensation methods. There are two modes of operation that the toolkit supports. In the *evaluator mode*, the user provides models of both the plant and the compensator. In the *automatic compensator mode*, the user provides the plant model, chooses the desired specifications, and the method of compensation, and lets MASCoT come up with the appropriate compensator. The system model is specified with poles, zeros, and the gain of the system.

The user interacts with *MASCoT* via a graphical user interface (GUI). The GUI communicates user inputs to the other modules of the system. Besides the GUI, there are three other modules: the system and its properties, the specification table and specifications, and the compensating engines. These modules communicate with each other and with the user through the graphical user interface, as pictured in Figure 5-1.

MASCoT is written in Java, which allows for a nice graphical user interface, platform independence, and an object-oriented design.

5.1 High Level Perspective

MASCoT consists of four modules which interact together: the *GUI* module, the *SystemModel and Properties* module, the *Specifications Table and Specifications* module, and the *Engines* module. The user interacts with the toolkit via the GUI module, which then starts the appropriate sequence of events by calling other modules with the appropriate process requests.

The GUI module gets the input from the user and starts the appropriate program sequence. The input from the user includes the systems' models, and what the user wishes the toolkit to do. A system model is defined by the user who specifies poles, zeros, and the gain of each system. Based on what the user requests, the GUI calls other modules in the appropriate sequence.

There are two sequences involving the interaction of the GUI with the other modules: evaluation of user-supplied compensator and auto-compensation. Figure 5-1 shows how the modules are interconnected. Numerous interfaces and abstract classes standardize and thus simplify communication and control between different modules.

In the *user-supplied compensator* case, the GUI calls the SystemModel module to evaluate and store the system properties. When the system model is complete, the GUI requests it from the module and displays it to the user.

In the *auto-compensator* mode, the GUI first calls the system model module to get the system properties. Then it calls the specifications table to configure all the user specification. Lastly, the GUI calls the appropriate compensating engine with the prepared system and specifications. The compensating engine figures out the compensator model and evaluates the compensator's and the compensated system's properties using the system module. The engine compares the resulting system properties to the specifications defined in the specifications table. If the results are satisfactory, the engine passes the compensated system back to the GUI. Then the GUI displays these results to the user.

5.2 Code Overview

Each modeled or computed system property is described via an object, which stores and changes the information if necessary. The state of any system property is automatically kept. The properties pertaining to a given system are congregated together to define a given SystemModel object. All related information is thus stored together. A set of public methods is defined via which other

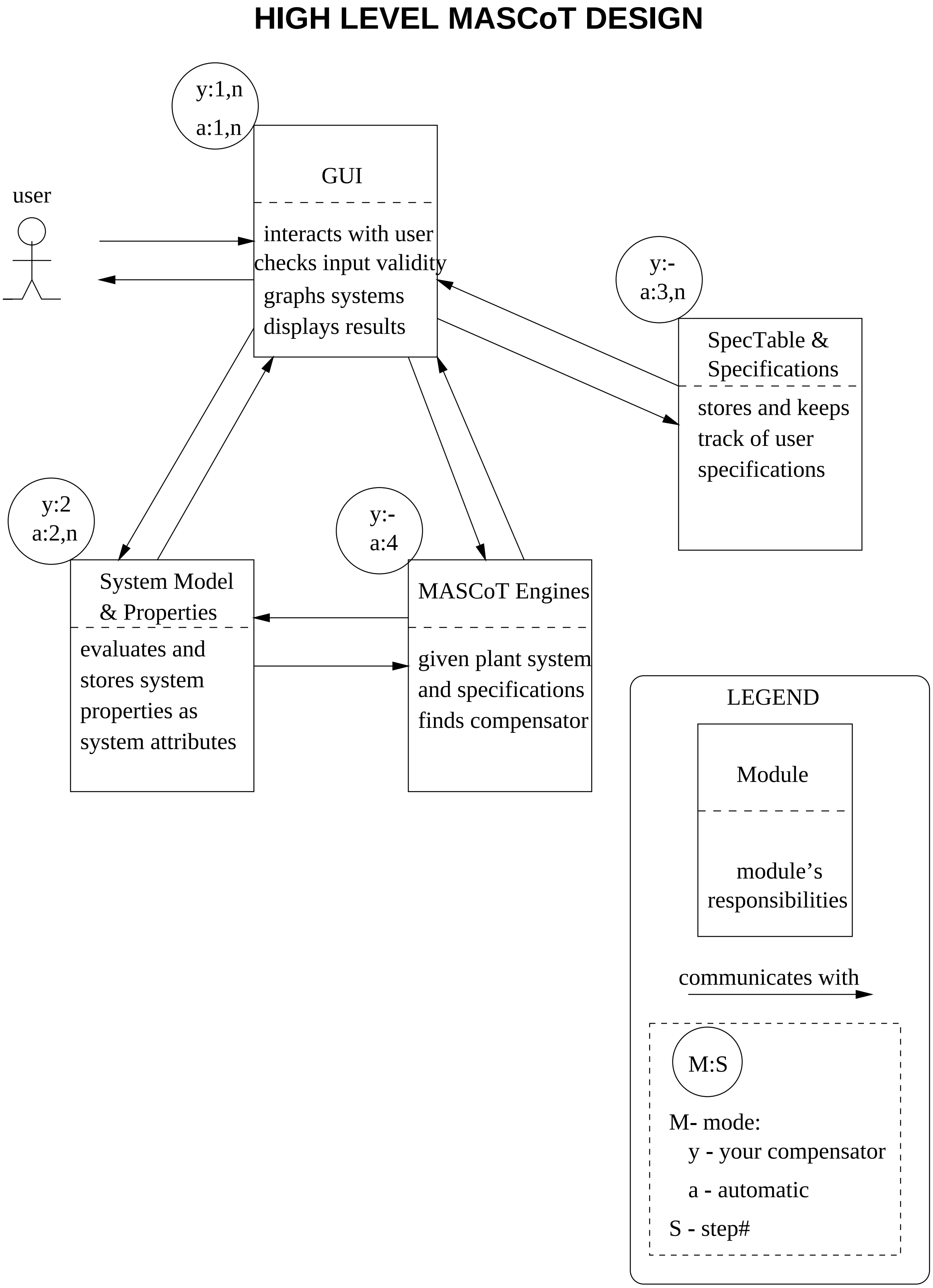


Figure 5-1: Seeing the *Big Picture.*

modules can retrieve and/or modify the system property. All the actual properties are specified as *private* and cannot be accessed directly.

Similarly, each specification provided by the user is stored as a particular type of a specification object. The specifications, along with the desired compensation method, are grouped together in a specification table class.

Finally, the compensating engines all derive from an abstract class *Engine*. However, each engine is implemented as a separate class and pertains to a given compensation method, such as proportional, dominant pole, lead, lag, or undefined.

All the user-defined classes developed for *MASCoT* are grouped together in a single package called *thesis*. There are no separate packages to separate the GUI from the back-end components. Using multiple packages to isolate the graphics, the system evaluators, and the compensating engines would be a good model to follow in the future.

5.3 Flow Control

5.3.1 GUI responsibilities

The GUI modules are responsible for the user interface, which is controlling the input and output of information in a format understood by the user. When the GUI module gets the input from the user, it first checks its validity. If the input is invalid, it promptly displays the problem to the user and ask for correction. If the input is valid, it starts the appropriate program sequence. It passes the user's input to the appropriate SystemModel modules for evaluation and to the appropriate Engine modules for computation. The output from these modules is returned to the GUI modules for display back to the user. The communication scheme is implemented by passing references of objects which need to communicate and via object listeners which await the appropriate user commands to invoke SystemModel and Engine constructors and methods.

The GUI modules also include plotting utilities. The plotting is used to display the graphs of the magnitude and phase characteristics of systems. The MagnitudePlot and the PhasePlot class objects compute the appropriate function values based on the system model supplied by the user. The plotting classes are responsible for drawing the computed coordinates. The user can also zoom in and zoom out on the plots.

SystemModel & Engine responsibilities

There are two primary *MASCoT* modes available:

1. Evaluation of a given plant, the user's own compensator, and the feedback system consisting of the plant series compensated by the user-supplied compensator.

2. Automatic design of a compensator based on a given plant and a set of specifications.

Your Compensator mode

In the **Your compensator** mode the Engine and the Spec modules are not used, since all we are interested in is computing system properties. When a user decides to evaluate his compensator, the Java Event Handler instantiates the appropriate SystemModel object, using the system's singularities as arguments. The SystemModel's method instantiates different methods of the Properties classes which in turn evaluate the relevant system properties to be stored within the system object. The information is then returned to the GUI modules for display.

Automatic Compensator mode

In the **Automatic compensation** mode the user inputs the plant model, selects the *Automatic* mode, the type of compensation to be used, and the requirements that the compensated system should meet. When the user clicks to execute, the event handler processes the request by instantiating the SpecTable object. The SpecTable object instantiates the appropriate Spec objects to hold the specification information. Depending on the type of compensation chosen, the appropriate compensating engine is called. The engine translates the SpecTable information into an appropriate system model which completely or partially meets the requirements. The original system is evaluated just as it was done in user-compensated mode. Two SystemModel objects, the original system and the compensator are combined together in a closed loop to instantiate yet another SystemModel object which is the compensated system.

The properties of the compensated system are compared with the specifications from the SpecTable. If the requirement are met, the result is passed to the GUI module to display to the user. If they are not met, the procedure is repeated with a different compensator. Once there are no more compensators to try and the best possible solution has been found, the control is returned to the GUI modules, which display the appropriate message to the user.

5.4 Design Choices

5.4.1 Event model

The SystemModel and the Specs objects are created when the *MASCoT* program starts, in order to easily pass their references to other objects with which they need to communicate. The same thing is done for the different GUI modules responsible for various panels within the *MASCoT* frame. The communication, in Java jargon the Event Model, between the components is realized by passing references of objects that are listening and responding to events to the objects that are creating the events.

5.4.2 Interfaces and abstract classes

A set of interfaces, such as the *SystemPropInterface* and the *SystemSpecInterface*, define method prototypes that must be implemented by all the system Properties and Spec objects, respectively. This makes manipulating the data within the objects a lot simpler, since the retrieve, store, and compute methods are all standardized. A number of abstract classes, which are shared by different properties and specification classes serve a similar purpose.

The Error class is an abstract class from which different classes of errors inherit, including SteadyStateError and PropertyRangeError. The methods that are common to both errors are predefined in the abstract class; the other methods have function prototypes to be fully implemented by the appropriate subclasses of the Error class.

The Spec class is an abstract class from which all the specification classes inherit methods. All of the methods within the Spec class are predefined for the most commonly seen specifications, and the variations are implemented in the subclasses which override the method with their own definitions.

The Engine class is an abstract class from which all the compensating engines extend. It holds many method prototypes which must be implemented by all the compensating engines. It predefines some method definitions, which are shared by two or more compensating engines. All the engines implement the *EngineInterface* to standardize access to their methods.

5.4.3 Why Java?

The other comparable feedback compensation software currently available, the *Control System Toolbox* from MathWorks, Inc., is written in C/C++. Speed is one main advantage of using C/C++.

However, I chose to implement my toolkit using Java, for several reasons.

1. The graphics libraries such as the *Swing* package available in Java allow for creating impressive user interfaces. A simple graphical interface allows even a novice user to become completely familiar with all the features of *MASCoT* within minutes.

2. Java is platform independent. Thus, from a single version of code, one can have a program which can run on virtually any machine. Moreover, since *MASCoT* is also an applet, the user can run the toolkit as an applet in a web browser.

3. Java is an object-oriented language. I can create class libraries, and if the design is well-done, reuse already written modules.

4. There are no pointers in Java. Pointers are the source of 90 % of software bugs.

The main problem with Java is speed. Every time *MASCoT* runs, its files are first read by the *Java Virtual Machine*. The *Virtual Machine* is platform specific and exists on every platform (Windows, Unix, Solaris). It translates Java class code into machine instructions so that the machine knows what to do. In case of a C/C++ program, when the code is compiled it is written in machine-specific language. When a C/C++ program is run, there is no intermediary "translator." Since a whole step is omitted the program loads faster.

However, today's processors are fast enough so that the wait is not significantly longer. Multi-threaded Java is an attempt to improve the speed of the system run. I did not attempt to write multi-threaded program since that would add to the code complexity. The efficiency and speed of execution were not among the primary goals of the implementation presented in this thesis.

Chapter 6

Graphical User Interface (GUI)

640K ought to be enough for anybody.

Bill Gates, in 1981

The user interacts with the compensation toolkit via a fully functional Graphical User Interface (GUI). The graphical user interface for *MASCoT* is written entirely in Java and supports the most recent versions of Sun's Java release, the Java 2 Platform. This release includes new graphics libraries as part of the *Swing* development environment.

6.1 Application & Applet

MASCoT can run either as an application or as an applet. When run as an application, *MASCoT* acts like a regular program. Examples of regular programs are a word processor program, a spreadsheet program, and a drawing program. These programs are normally stored and executed from the user's local computer. *MASCoT* can also run as an applet, which is meant to be stored on a remote machine that users will connect to via a World Wide Web browser, such as the Internet Explorer or Netscape. Upon a request from the user, the *MASCoT* applet is loaded from a remote computer into the browser, executed at the browser, and discarded when the execution completes.

6.2 High-level View on the GUI Design

A major part of the graphical user interface uses *Swing* components as building blocks for its modules. The execution starts from the class Compensator, whose methods support both the applet and the application mode. All the user interface classes along with all the engine and properties modules belong to the package *thesis*.

In the applet mode, the execution starts from the *init()* function which invokes the applet frame. An instance of the Compensator class is loaded into the applet. The *Compensator* class defines the general layout of the main *MASCoT* interface. It contains other panels which are responsible for the input from the user, launching of other frames, and graphing functions.

In the application mode, the *main()* method is called. A JFrame, Java graphical component, is created and the object of the class Compensator is inserted into the frame. The Compensator class is a subclass of the JApplet class, thus it inherits the applet's methods. However, since we are in the application mode, the applet's *init()* method must be called explicitly. Unlike in the applet mode where this method is called implicitly.

The Compensator instantiates the next level of panels contained within it, the top panel and the BottomPanel. The BottomPanel is implemented as a separate class. It has mainly the input fields and buttons for user interaction and launching other frames. The top portion of the Compensator holds two graphs, each one of them an instance of the GraphPanel class. The Compensator frame is the primary *MASCoT* user interface. From here, all of the other frames are launched and the compensator program is run.

There are three other frames that can be launched separate from the main *MASCoT* frame. The SpecJFrame is where the user specifies the system constraints in the *Automatic* mode. The PropDisplayFrame contains all the relevant information about the original system, the compensator, and the compensated system. The SpecDisplayFrame has information on the specifications and how well the compensated system meets these specifications. The overall GUI design is shown in Figure 6-1. A more detailed description of the graphical elements is provided in the sections that follow.

6.3 Primary *MASCoT* Interface

The *MASCoT* Compensator frame is the primary frame launched when the program starts. It is an applet frame. If it is run as an application, this applet is inserted into a generic application

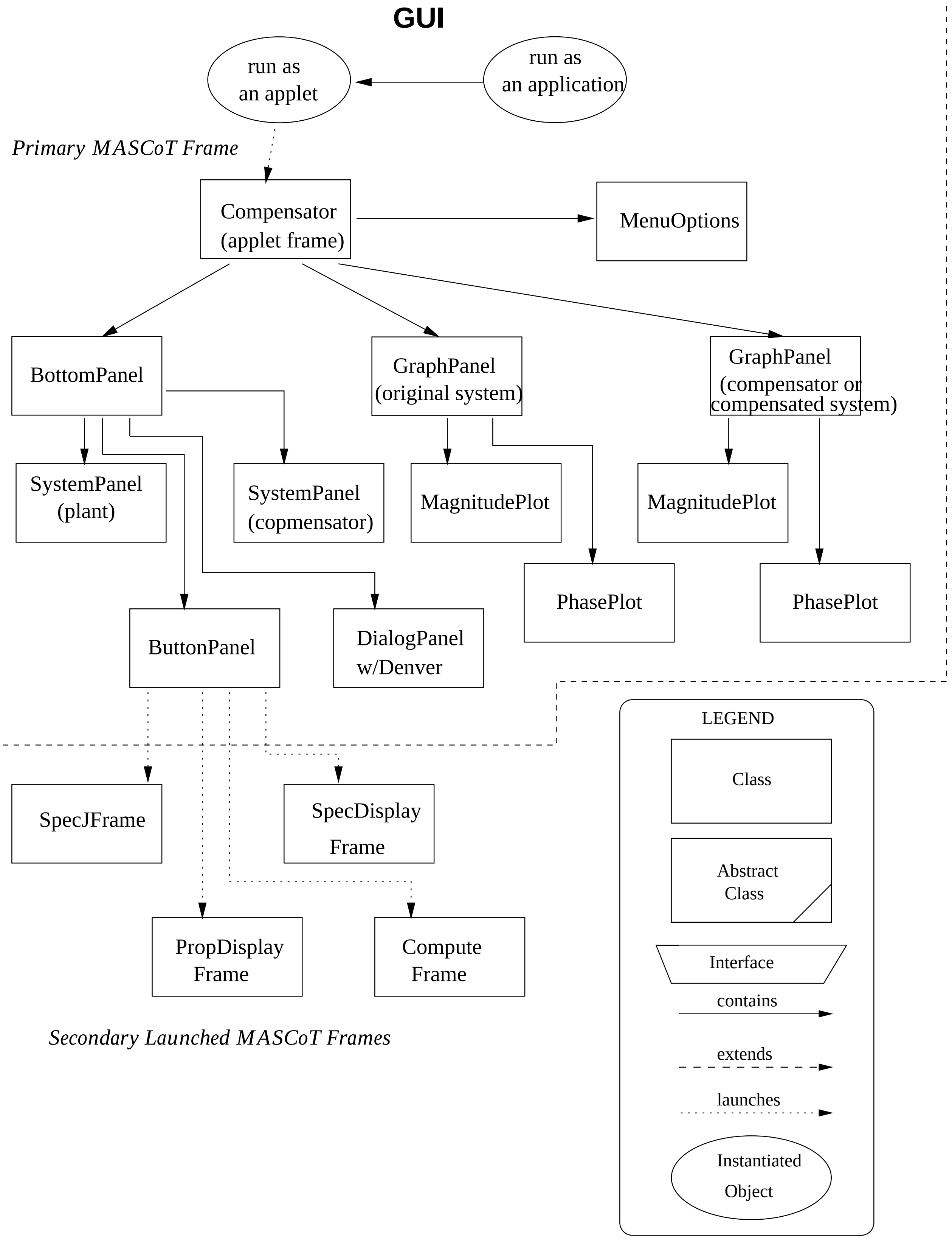


Figure 6-1: Diagram of the overall GUI design.

frame. If it is run as an applet, it is inserted into a specified location in the hypertext document. It can also be launched via an *appletviewer*. An appletviewer is a browser specifically designed to launch applets.

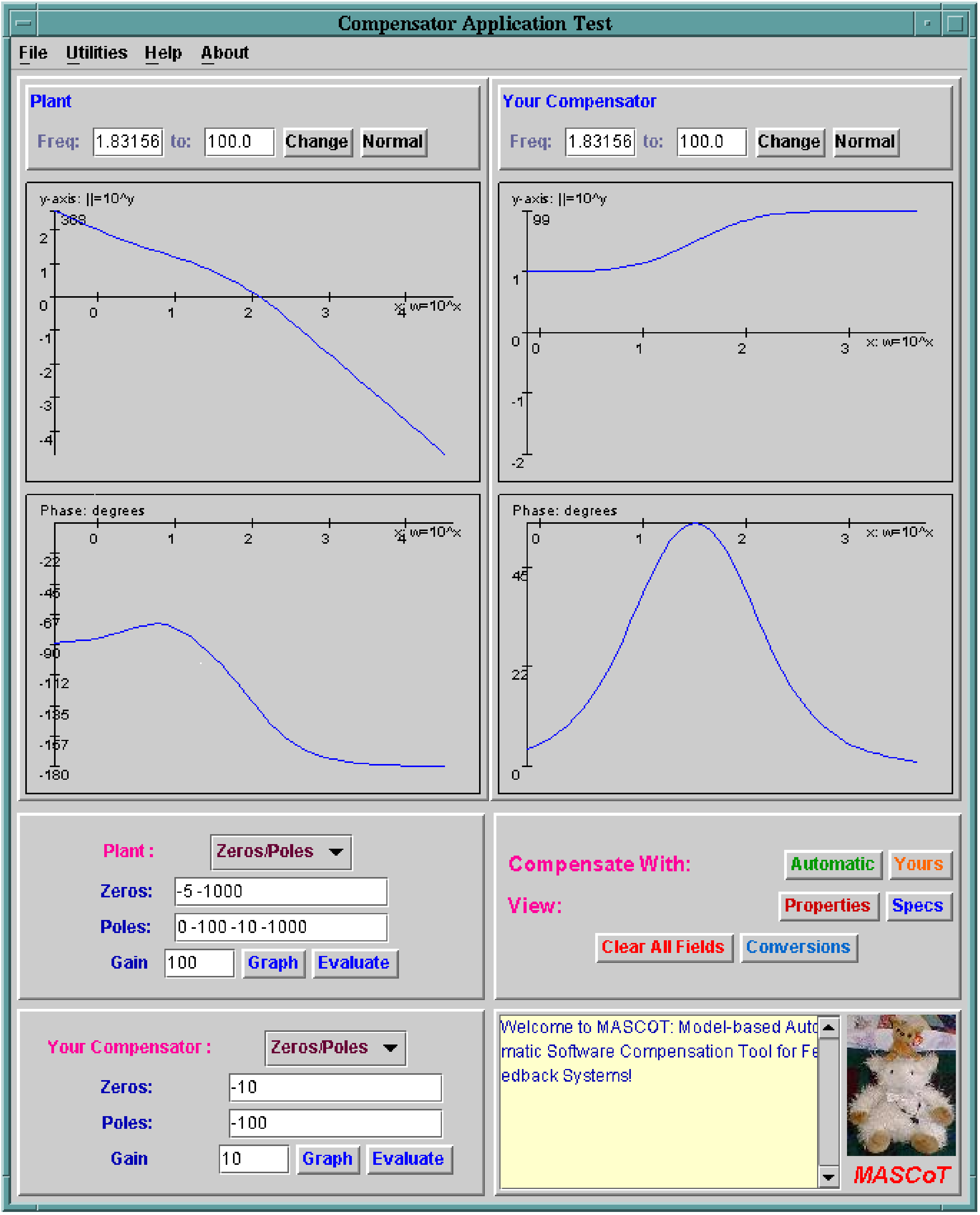
The *MASCoT* primary frame contains three panels implemented as separate classes. The top portion contains two GraphPanel objects, and the BottomPanel object holds system parameter panels. The picture of the *MASCoT* frame is shown in Figure 6-2.

The GraphPanel holds objects of the panel responsible for zooming in and out, the magnitude plot, and the phase plot. Given a system, the GraphPanel invokes the MagnitudePlot and the PhasePlot class to plot the magnitude and phase of a system. The left side panel is used to plot the original system, before compensation. The right side panel is used to plot the compensator or the compensated system. Zooming capability is achieved by specifying the range of frequencies to plot and pressing the *Change* button. The user can always return to the default with the *Normal* button. A more detailed description of the graphing capability and procedures is in section 6.4.

The BottomPanel holds four other panels. Two on the left are instances of the SystemPanel class, one used to input the model parameters of the original system, the other used to input the model parameters of the compensator. The third panel contains buttons to invoke automatic compensation or analysis of the compensator provided by the user. The *Properties* button launches a frame containing information about the properties of the original system, the compensator, or the compensated system, see section 6.7. The *Specs* button launches a frame comparing the desired and the achieved performance based on the specifications.

The system can be compensated automatically or with a compensator provided by the user. To run the automatic compensation tool, the *Automatic* button is pressed. This launches a frame which asks the user for desired properties to be met by the system, as described in section 6.5. To evaluate the performance of the system when compensated by the user, the button labeled *Yours* should be pressed.

The two remaining buttons on the BottomPanel are *Clear All Fields* and *Conversion*. Pressing the first button clears all of the data currently being displayed, effectively reinitializing the toolkit. The *Conversion* button is an additional feature added later on in the development process. When pressed, the ComputeFrame is launched, which allows the user to quickly compute the relationships between variables that characterize a system in the time and in the frequency domains. The ComputeFrame is able to calculate these relationships for the first and for the second-order



Plant Freq: 1.83156 to: 100.0

Your Compensator Freq: 1.83156 to: 100.0

Plant : Zeros/Poles

Zeros: -5 -1000

Poles: 0 -100 -10 -1000

Gain 100

Your Compensator : Zeros/Poles

Zeros: -10

Poles: -100

Gain 10

Welcome to MASCOT: Model-based Automatic Software Compensation Tool for Feedback Systems!

Figure 6-2: The primary MASCoT frame.

parameters.

The fourth panel contains a dialog box providing feedback to the user about the currently performed activity. It also contains a snapshot of a *MASCoT*'s mascot, a cashmere sheep called *Denver*.

6.4 Graphing System Function

6.4.1 General information

The MagnitudePlot class is responsible for the Bode magnitude plot of a system. This particular class extends the Canvas class, which is an AWT component. The Canvas class allows for a reasonably simple implementation of graphics. The magnitude is plotted on a log-log scale. In order to graph a system, the MagnitudePlot class is provided with the system model in the form of the system poles, zeros, and gain. The ticks on the axes are shown for every decade, and the scale numbers indicate powers of ten.

The PhasePlot class is very similar to the MagnitudePlot class. It is also a Bode plot. It uses the same graphical components, however, this time it is a phase plot, so the scale is log-linear. In a manner similar to the MagnitudePlot class, it draws the system graph from the user-supplied poles, zeros, and gain.

6.4.2 Plotting details

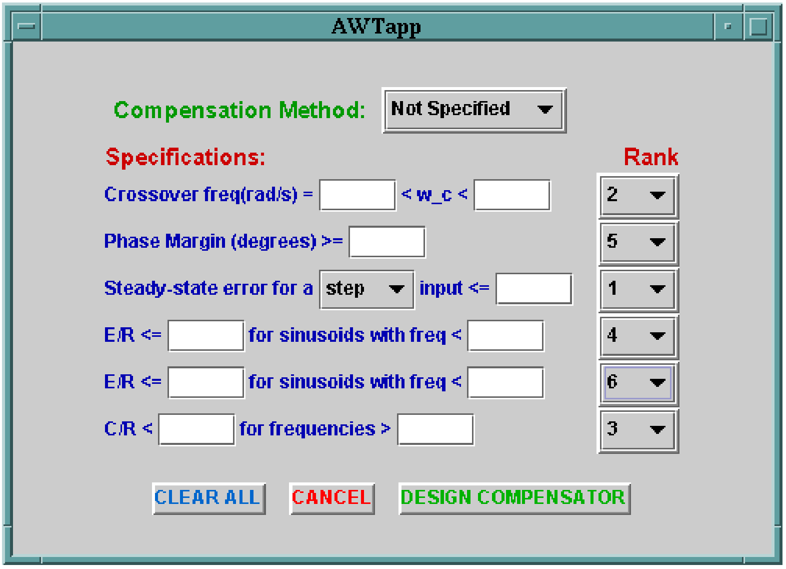
The default range of frequency values to plot is specified depending on the system being plotted. The minimum value is located a decade below the lowest-frequency singularity. The maximum value is a decade above the highest-frequency singularity.

After determining the extreme values for ω, the *step size* in logarithmic scale is determined by dividing the range by 50. The maximum and minimum values of the function are plotted, either in magnitude or in phase, as determined by evaluating the function at each one of these points. From the maximum and minimum values in the region, the vertical scale is determined. This allows stretching the plot vertically in order to fit the whole available canvas.

When the user chooses to readjust the frequency range in order to zoom in and out, after checking for invalid inputs such as negative numbers or characters, the minimum and maximum frequency (ω) to plot is set. The process of plotting the function is repeated with the adjusted step size.

6.5 SpecJFrame

A SpecJFrame is launched in the auto-compensation mode, after the user presses the *Automatic* button. The choice box on the top of the SpecJFrame allows the user to specify the type of compensation. Depending on the compensation method picked by the user, the specification options below become disabled or enabled. The reason the available specifications vary is because different compensation methods allow improvement in different dynamics of the system. Thus specifying properties which cannot be improved is not necessary. Figures 6-3,6-4, 6-5, 6-6, 6-7 show snapshots of different modes of the SpecJFrame.



Compensation Method: Not Specified

Specifications: Rank

Crossover freq(rad/s) = < w_c < 2

Phase Margin (degrees) >= 5

Steady-state error for a step input <= 1

E/R <= for sinusoids with freq < 4

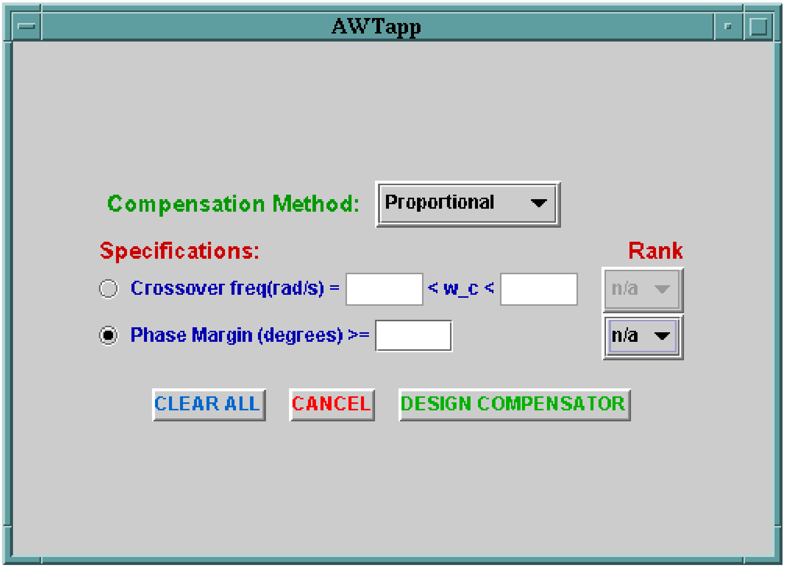
E/R <= for sinusoids with freq < 6

C/R < for frequencies > 3

Figure 6-3: Specification frame when the compensator type is not specified.

6.6 SpecDisplayFrame

This frame is launched via the *Specs* button. This button is enabled after running the automatic compensator. In this mode the user provides specifications to be met by the system. The SpecDisplayFrame lists the requested specifications and their respective ranks and the method of compensation used. Figure 6-8 shows a snapshot of the frame.

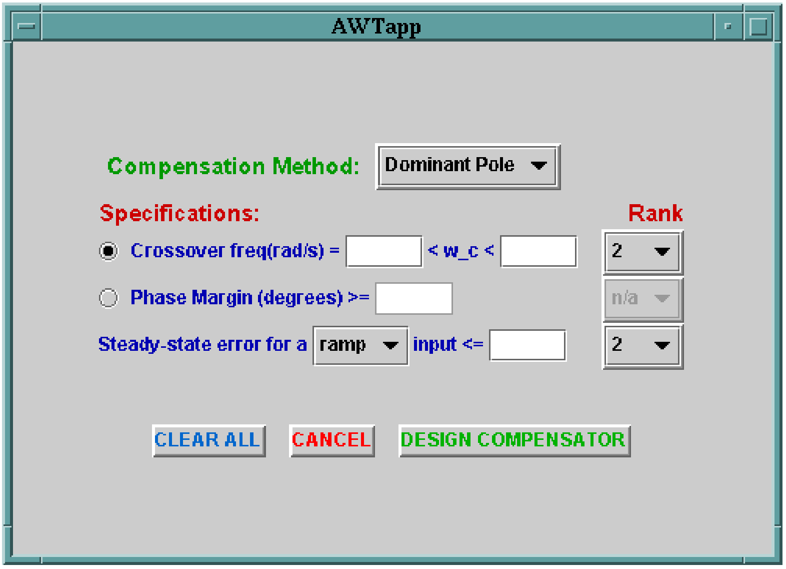


Compensation Method: Proportional

○ Crossover freq(rad/s) = < w_c < n/a

● Phase Margin (degrees) >= n/a

Figure 6-4: Proportional compensation mode.



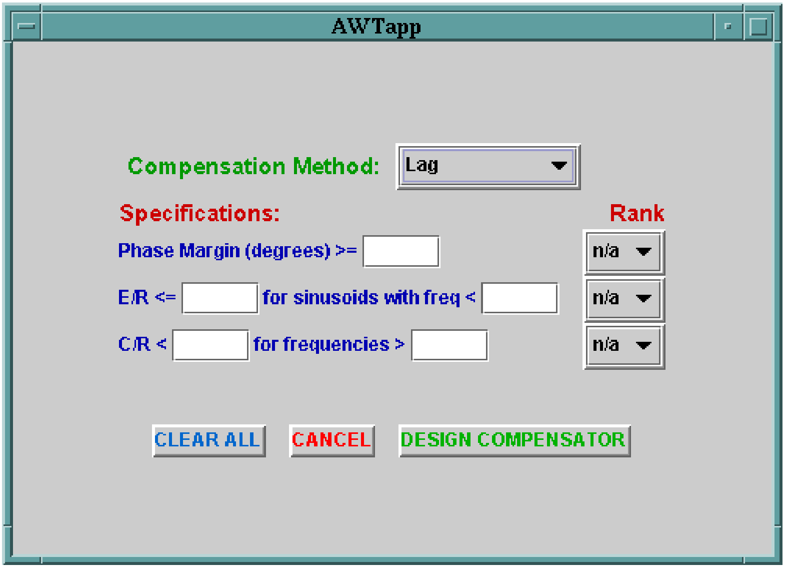
Compensation Method: Dominant Pole

● Crossover freq(rad/s) = < w_c < 2

○ Phase Margin (degrees) >= n/a

Steady-state error for a ramp input <= 2

Figure 6-5: Dominant pole mode specifications.



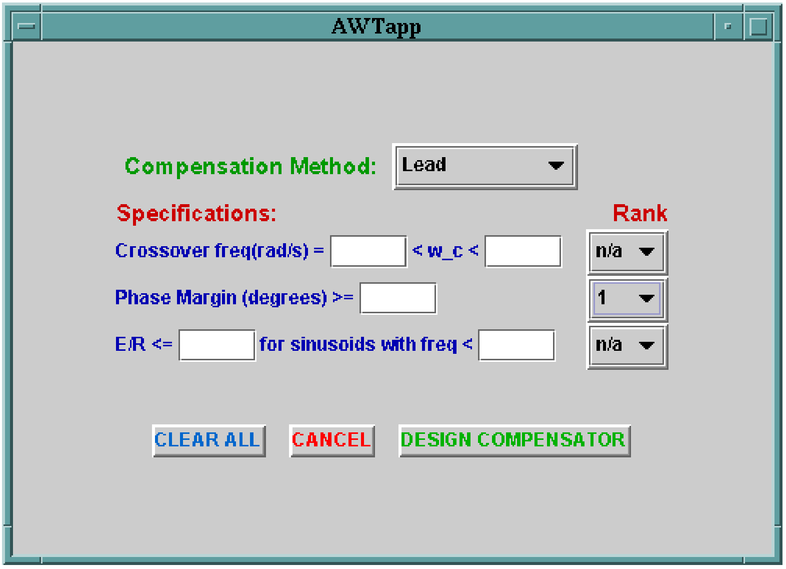
Compensation Method: Lag

Phase Margin (degrees) >= n/a

E/R <= for sinusoids with freq < n/a

C/R < for frequencies > n/a

Figure 6-6: Lag compensation mode specifications.



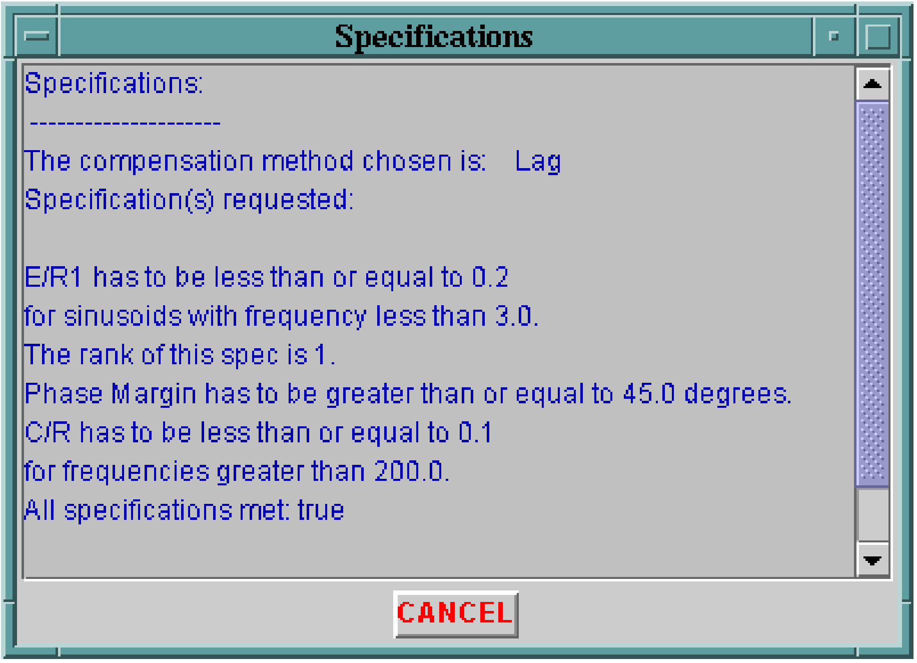
Compensation Method: Lead

Crossover freq(rad/s) = < w_c < n/a

Phase Margin (degrees) >= 1

E/R <= for sinusoids with freq < n/a

Figure 6-7: Lead compensation mode specifications.



```
Specifications:
--------------------
The compensation method chosen is:   Lag
Specification(s) requested:

E/R1 has to be less than or equal to 0.2
for sinusoids with frequency less than 3.0.
The rank of this spec is 1.
Phase Margin has to be greater than or equal to 45.0 degrees.
C/R has to be less than or equal to 0.1
for frequencies greater than 200.0.
All specifications met: true
```

Figure 6-8: Picture of the SpecDisplayFrame

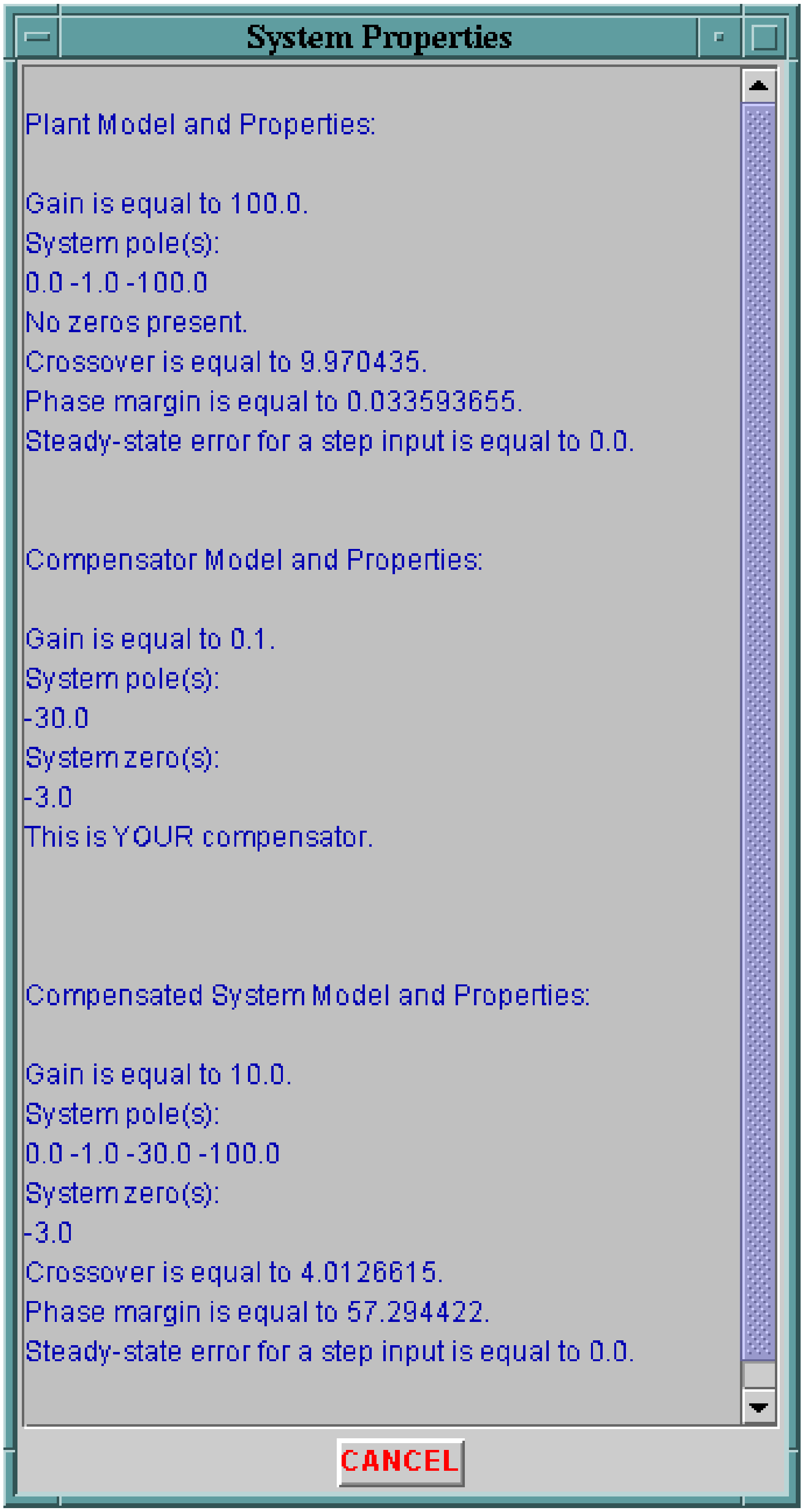
6.7 PropDisplayFrame

Pressing the *Properties* button launches the PropDisplayFrame, which displays information about the original system, the compensator, or the compensated system. This frame is empty until the user analyzes or compensates his system. The information provided is a set of common system properties in the time and frequency domain, as shown on the snapshot in Figure 6-9.

6.8 ComputeFrame

The user can access the ComputeFrame with the *Conversions* button. This frame allows the quick and easy computation of relationships between either first or second-order system variables. The frame switches between the first and the second-order system models when the forward or the backward arrows are pressed. Figures 6-10 and 6-11 show the snapshots of the frame with the variables which model first-order and second-order systems, respectively.

The relationship between the variables is computed using the equations from Table 3-8. Entering one or more of the variables on the left and pressing the right-facing arrow in the middle results in an attempt to compute the values of the variables on the right-hand-side. If none can be computed,



Plant Model and Properties:

Gain is equal to 100.0.
System pole(s):
0.0 -1.0 -100.0
No zeros present.
Crossover is equal to 9.970435.
Phase margin is equal to 0.033593655.
Steady-state error for a step input is equal to 0.0.

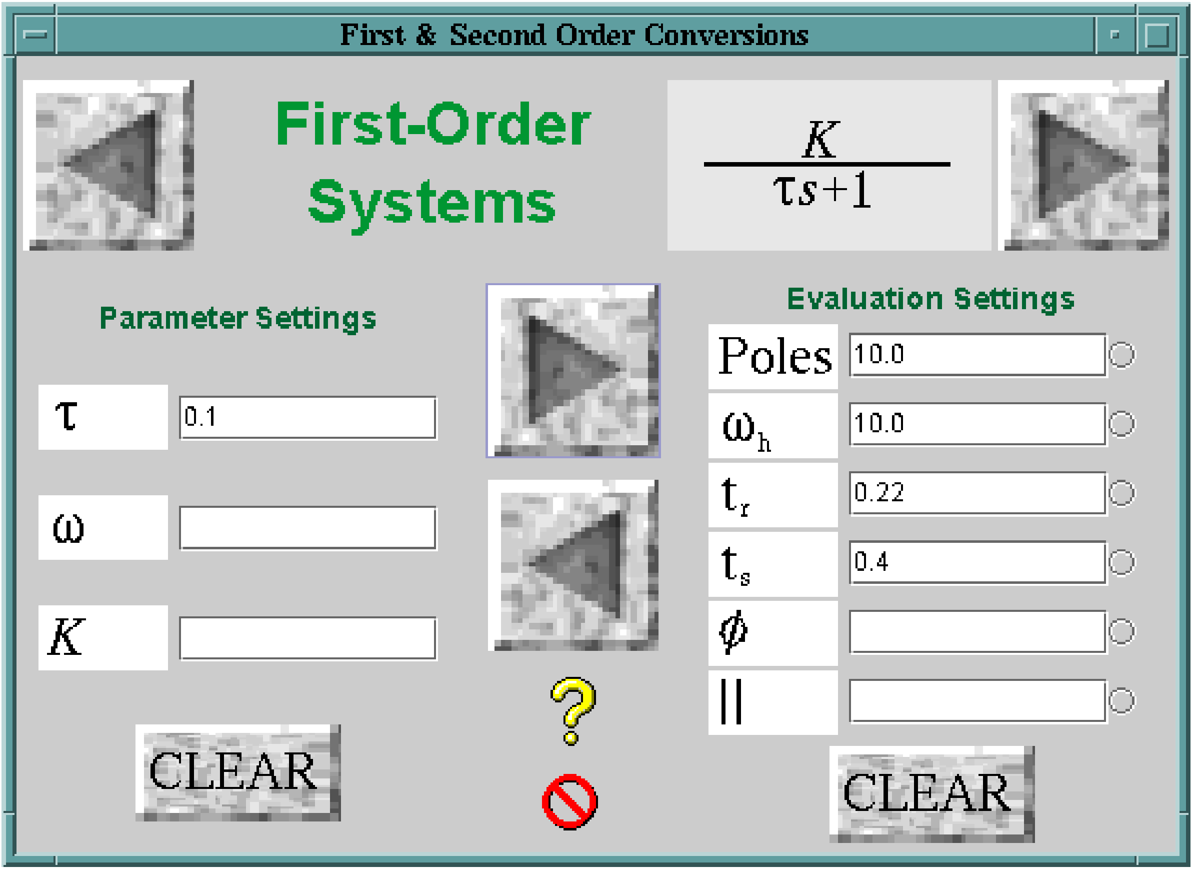
Compensator Model and Properties:

Gain is equal to 0.1.
System pole(s):
-30.0
System zero(s):
-3.0
This is YOUR compensator.

Compensated System Model and Properties:

Gain is equal to 10.0.
System pole(s):
0.0 -1.0 -30.0 -100.0
System zero(s):
-3.0
Crossover is equal to 4.0126615.
Phase margin is equal to 57.294422.
Steady-state error for a step input is equal to 0.0.

Figure 6-9: Picture of the PropertiesDisplayFrame



Parameter Settings

τ 0.1

ω

K

Evaluation Settings

Poles 10.0 ○

ω_h 10.0 ○

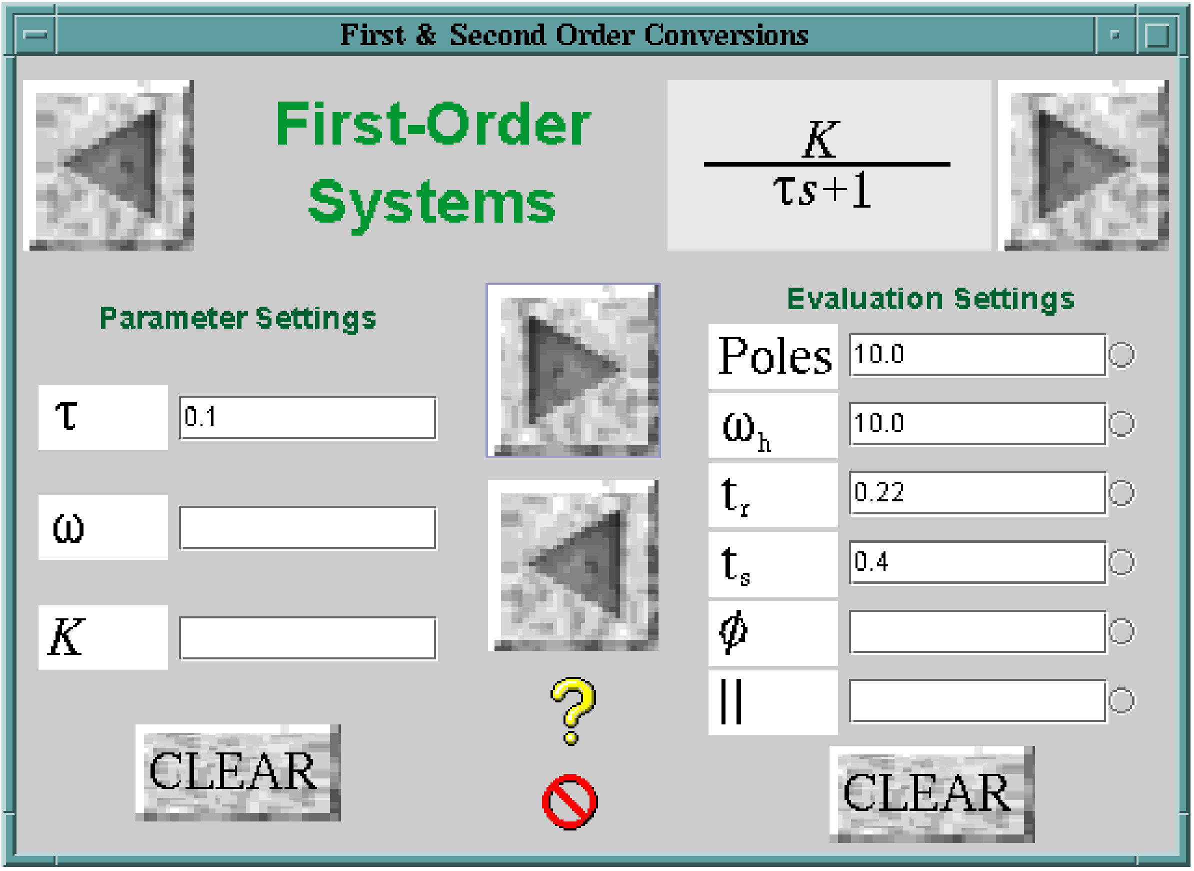
t_r 0.22 ○

t_s 0.4 ○

ϕ ○

$||$ ○

Figure 6-10: Picture of the ComputeFrame with the first-order system model



First & Second Order Conversions

First-Order Systems

$$\frac{K}{\tau s+1}$$

Parameter Settings

τ 0.1

ω

K

Evaluation Settings

Poles 10.0 ○

ω_h 10.0 ○

t_r 0.22 ○

t_s 0.4 ○

ϕ ○

$||$ ○

Figure 6-11: Picture of the ComputeFrame with the second-order system model

nothing is displayed. Otherwise, one or more corresponding values are displayed on the right. Similarly, the reverse direction can also be computed. In this case, however, the user specifies only one of the values on the right-hand-side by selecting the appropriate circle next to it, and then presses the left-facing arrow to convert.

6.9 Menu options

There are four menu options on the top of the main *MASCoT* interface: *File, Utilities, Help,* and *About.* From the *File* menu the user can choose *Open* file to load a modeled system into the toolbox or *Save* to store the output from a *MASCoT* run. From the *Utilities* menu the user can launch *Preferences* to change the look of the GUI or *Print* to print the output from the compensator or the graphs. The *Help* menu has *Documentation* on the input format, specifications, and graphics in the form of a user manual. The *About* menu has the copyright and contact information.

6.10 *MASCoT* User Manual

MASCoT as an application is launched by typing on the command line:

```
java thesis.Compensator
```

To start *MASCoT* in a browser window, the user has to go to the proper html page which has the link to the *MASCoT* applet embedded in it. When the user loads the html page, the embedded applet code is launched and inserted into the proper location on the webpage.

MASCoT operates in two modes:

1. Evaluator of a user-provided compensator.

2. Engine that designs a compensator given a set of specifications.

The user models the plant or the compensator system with zero, pole, and gain values. The locations of the singularities must be separated by spaces or commas and have no imaginary parts.

For the **auto-compensation mode**, no compensator model is necessary, since the *MASCoT* engine will design the compensator. The user inputs a model of the plant with pole, zero, and gain information and presses the *Automatic* button. A frame requesting desired system performance specifications is launched. There the user chooses the method of compensation, fills in any specifications of interest, and runs the compensating engine. After the run, the original system plot is displayed along with the compensated system loop transmission plot. System properties and achieved performance can be accessed via the *Specs* and *Properties* buttons.

To run in the **user-provided compensator** mode, the user types in the singularities and gain of the plant and of the compensator. To run the evaluator, *Your* compensator button is pressed. As a result, the plant and the compensated system loop transmission are graphed. The *Properties* button allows access to the information about the resulting system.

Graphing The user can graph either the compensator or the plant by pressing the *Graph* button next to the appropriate system model. The *Eval* button allows to evaluate plant or compensator properties before compensating the system.

When graphing, the user can choose to see any portion of the plot in more detail by typing in the appropriate frequency ranges and pressing the *Change* button. To get back to the default display, the *Normal* button is pressed. The menu options on the top of the em MASCoT frame are described in section 6.9.

Chapter 7

System & Spec Modules

We must not cease from exploration and the end of all our exploring will be to arrive where we began and to know the place for the first time..

T. S. Elliot

A system model in the form of user-provided poles, zeros, and gain is used to instantiate an object of a class *SystemModel.* The plant, the compensator, and the compensated system are all instances of this class. Within the SystemModel class, different properties are implemented via their own separate classes. Thus, we have a *Crossover* class, a *PhaseMargin* class, a *SteadyStateError* class and more. The diagram in Figure 7-1 shows all the major classes and how they relate to one another.

7.1 The SystemModel Class

The SystemModel class holds all the information about a given system which could be either a plant, a compensator, or a compensated system. The SystemModel constructor evaluates the time and frequency characteristics based on the user-supplied model, in terms of the singularities and the gain of a system. Each one of the SystemModel properties is an instance of a class representing

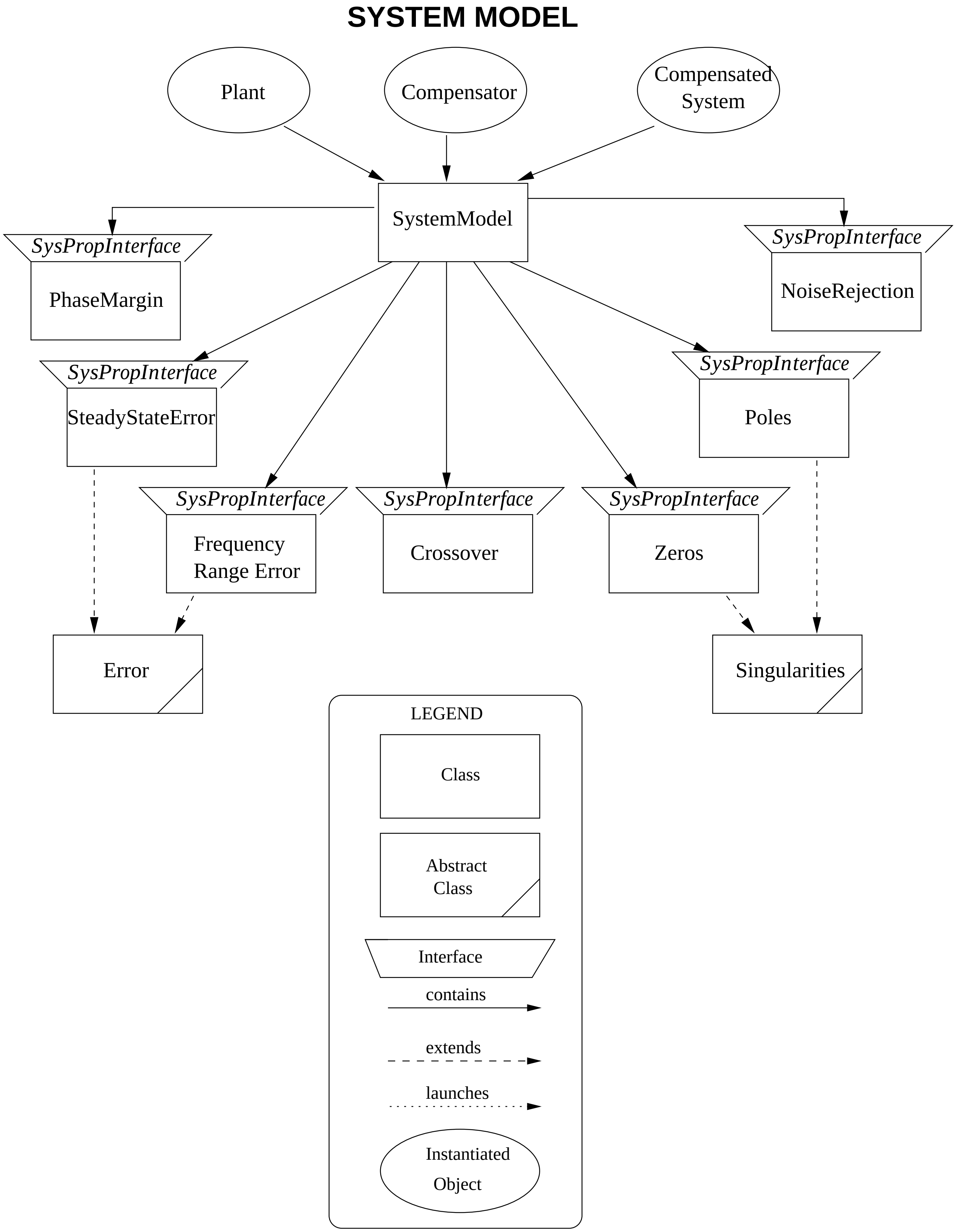


Figure 7-1: Diagram of the SystemModel design.

it. All of the properties implement the same *SystemPropInterface* which allows for simple access to each.

7.1.1 System instantiation

All three systems: plant, compensator, and compensated system, analyzed with *MASCoT* are instances of the same class, SystemModel. There are three different constructors used to instantiate each system. The plant's constructor has as its arguments modeled singularities of the plant. The compensator constructor has all of the above plus the mode of operation and the compensation method as chosen by the user. In case of the compensated system the SystemModel constructor has two SystemModel class arguments, the plant and the compensator, respectively, to be merged into one closed loop system.

7.1.2 Evaluation of a system

A system is evaluated when it is instantiated, as well as anytime its properties change. All of the property classes as described in section 7.2 are evaluated and saved. If the user wishes to recompute any of the properties, the method *setPropertyname()* should be invoked. If the user wishes to see any of the properties of a given system, the method *getPropertyname()* should be invoked. This method returns an object of the appropriate property class. In order to see all the information about a given system, the *toString()* method should be invoked on a SystemModel instance. This method returns a String representations of all of the system properties.

7.2 System Properties Classes

The *SystemPropInterface* class defines the methods supported by each one of the system properties described below. The methods are: *exists(), getValue(), setValue()*, *computeValue()*, and *toString()*. These methods are used to retrieve, store, compute, and relate to the user the value of a given property.

Crossover The Crossover class instantiates a crossover object, which is the frequency when the magnitude of the open loop transmission is 1. The constructor can either set the crossover value or it can compute it from the system model. This property class implements the *SystemPropInterface* along with all of the other property classes. The crossover frequency is in radians.

Phase margin The PhaseMargin class computes and stores the available phase margin of a system in degrees. If the phase margin is less than 0, the return value is zero. The PhaseMargin class implements the *SystemPropInterface*.

Singularities The Singularities class is an *abstract* class which is meant to be a superclass for the Poles and the Zeros classes described below. An *abstract* class cannot have any objects instantiated from it.

Poles The Poles class extends the Singularities class and implements the *SystemPropInterface*. It stores the pole values from the one occurring at the lowest frequency, to the one occurring at the highest frequency.

Zeros The Zeros class extends the Singularities class and implements the *SystemPropInterface*. It stores the values of the zeros from the one occurring at the lowest frequency to the one at the highest frequency.

Gain The Gain class is responsible for storing and retrieving the gain value. The default value of the gain is 1. The gain must be a positive number greater than 0. The Gain class implements the *SystemPropInterface*.

Error The Error class is implemented as an *abstract* class in Java, meaning there are no objects that can be instantiated with this type. It is, however, a superclass from which the SteadyStateError and the FrequencyRangeError subclasses derive.

Steady-state error The SteadyStateError class calculates and instantiates error values for zero frequency (DC) errors. Two types of inputs are handled: the step input and the ramp input. This class extends the Error class and implements the *SystemPropInterface*.

Frequency range error The FrequencyRangeError class calculates and stores the error magnitude for sinusoids in user-defined frequency ranges. This class derives from the Error class and implements the *SystemPropInterface*.

NoiseRejection The NoiseRejection class calculates and stores the information about the ratio of output to input, C/R, above some specified frequency range. This ratio informs us how much of the high frequency input makes its way to the output. This system property class implements the SystemPropInterface.

7.3 The SpecTable Class

When *MASCoT* is run in the auto-compensation mode, the user chooses the method of compensation and a set of performance specifications that the compensated system should meet. When the auto-compensation process is enabled, these specifications are saved by instantiating a *SpecTable* class which holds all this information. Each specification is saved in its own class object. Thus, we have *CrossoverSpec, PhaseMarginSpec, SteadyStateErrorSpec,* and other classes. All the specification classes extend the abstract *Spec* class. The Spec class implements methods common to all specifications. If the subclass needs to implement its own variation on the method, it simply redefines it. All specifications implement the *SpecInterface*, which prototypes some common methods, such as *exists(), getValue(), setValue(), meetsSpec().* The diagram in Figure 7-2 below shows how the SpecTable class is related to the Spec classes.

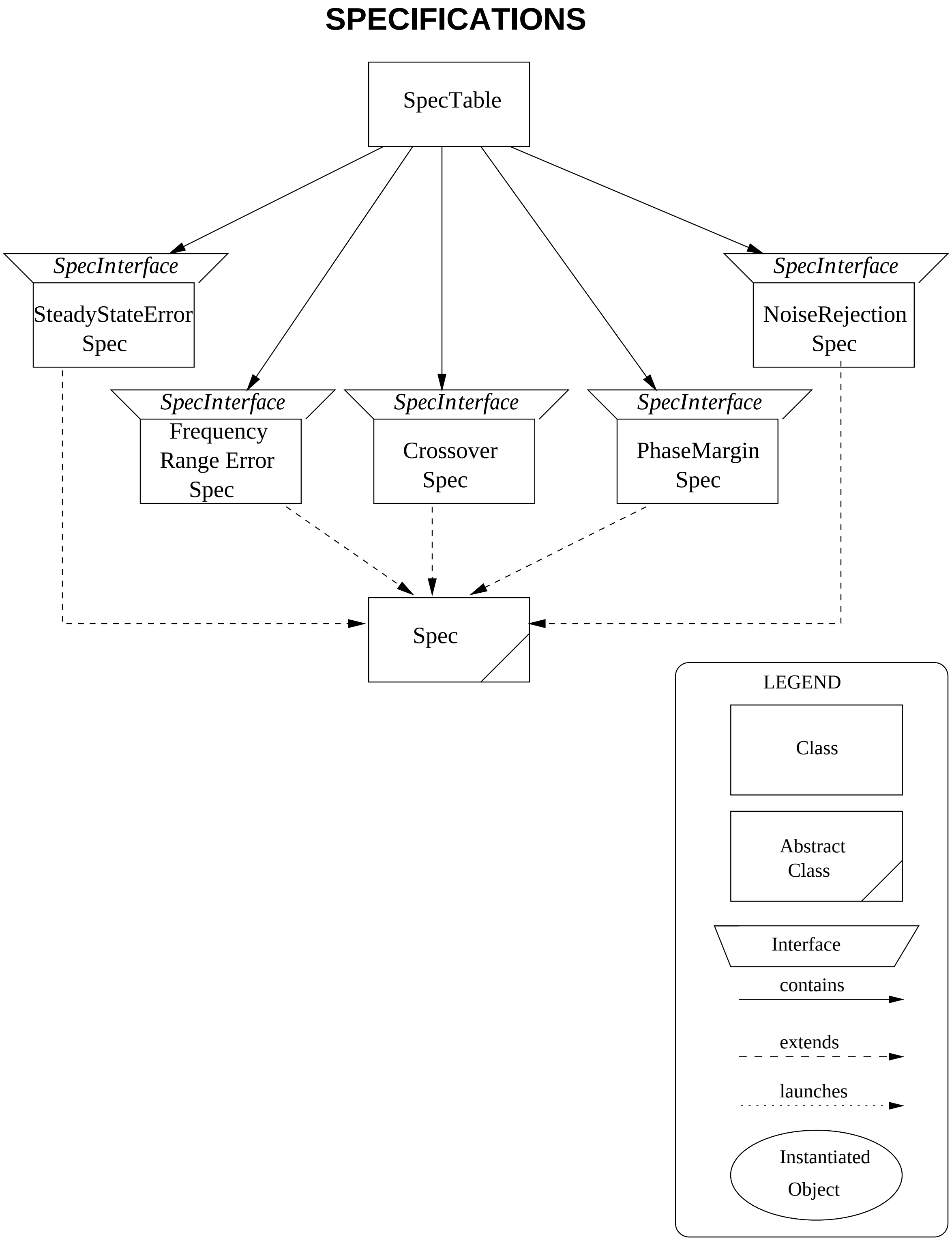


Figure 7-2: Diagram of the specification classes.

Chapter 8

MASCoT Compensation Engines (MEng)

Black holes are where God divided by zero.

Steven Wright

MASCoT compensation engines are a set of separate classes, each responsible for finding a given type of a compensator. The user provides the plant model, specifications, and the type of compensation to use. The type of compensation chosen determines which compensating engine is used. There are five compensating engines, each one implementing one of the five methods of series compensation: proportional, dominant pole, lag, lead, and 'not specified.'

8.1 Nature of a Control Problem

Before venturing into the implementation details of the different engines, let's formulate the general approach to the problems of system control followed by human designers. This will highlight approaches that "intelligent" software agents, such as *MASCoT* might follow.

In general, a control problem can be divided into the following steps:

1. A set of performance specifications must be established.

2. As a result of the performance specifications a control problem is defined.

3. A model of the physical system to be controlled is formulated.

4. The performance of the basic physical system is determined by different analysis methods, evaluating both the time and the frequency domain behavior.

5. If the performance of the original system does not meet the required specifications, equipment in the form of a control loop must be added to improve the response.

6. It might turn out to be impossible to meet any of the specifications. In such a case, an optimal solution in view of the specifications given should be defined.

8.2 MEng Modules Overview

8.2.1 MEng structure

MASCoT Engines (MEngs) are all independent compensating tools. Therefore, any engine can be easily separated and used alone. This is not to say that they have no properties in common. Since all of the engines are compensating tools and use similar techniques to evaluate systems, they all derive from the abstract superclass Engine and implement the *EngineInterface*. The ultimate goal of each engine is to stabilize and to improve the performance of a system. The Engine class defines a series of method prototypes and implements methods shared by the engines. The *EngineInterface* standardizes methods used by other modules to interact with the compensating engines. Figure 8-1 shows a high level diagram of the engines' dependencies.

8.2.2 MEng general approach description

Similarly to a feedback designer, MEngs follow a series of steps before coming up with the final solution. Although the details of the implementation of each step vary depending on the nature of each given compensator, the high level goal remains the same. Therefore, I divide the MEng feedback compensation approach into the following steps:

1. The plant model to be compensated provided by the user is tested for validity and compatibility with the given type of compensation. If the plant is not valid, the control is returned back to the GUI, which displays the appropriate error message to the user. Otherwise, the MEng proceeds to the next step.

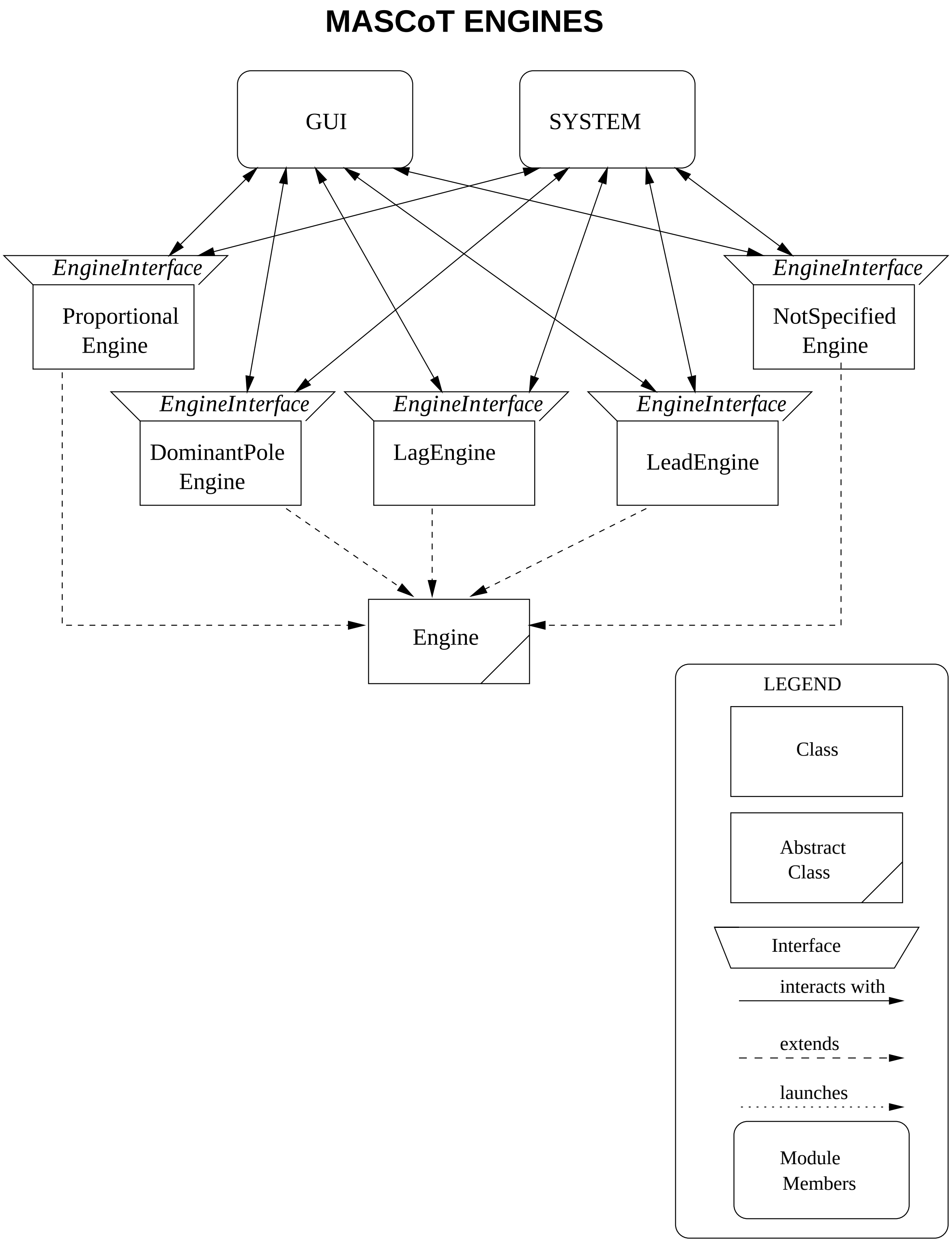


Figure 8-1: MEng design diagram.

2. The set of specifications provided by the user is tested for compatibility with the plant and the compensator type to be used. This step analyzes whether it is possible to meet a given specification with a particular compensator. As a result, some of the specifications are revised to more appropriate values, while others are ignored.

3. For each valid and revised specification, a compensator is found, which allows the compensated system to minimally meet this particular requirement. The solution is also restricted by the requirement that the resulting system must remain stable.

4. At this point, the number of potential compensators is equal to the number of valid specifications. The compensator solutions from each specification are merged two at a time, starting with the two specifications of the lowest priority. The higher-ranked one of the two specifications is used in the next pair-up. The converged solution from the previous iteration is used as the specification's compensator.

5. Finally, a single 'best' compensator is determined. Combining the plant and the compensator produces the compensated system's loop transmission. Iterating through the valid specifications, a set of relevant performance characteristics is evaluated both for the plant and for the compensated system.

8.3 Details on the Shared Properties of the MEngs

8.3.1 Plant validity testing

All compensating engines examine the plant's model to determine whether it represents a real system as well as whether it fits within the constraints of the $MASCoT$ system. $MASCoT$ is limited to compensating systems with real-valued left-hand-plane poles and zeros. The systems are assumed to be linear minimum phase and to have monotonically decreasing magnitudes.

Each engine tests a similar basic set of plant properties with a few minor additions dependent on its particular requirements. The validity tests include:

1. The plant is tested for any zeros at the zero frequency. Having zeros at zero implies that the magnitude of the plant is not monotonically decreasing since it starts off with a positive slope. Thus zero(s) at zero frequency make(s) the plant invalid.

2. The number of the plant's poles must be at least one or two more than the number of zeros since the plant magnitude should be monotonically decreasing. The requirement that there should be two more poles than zeros is implemented in a few of the engines since otherwise their compensation efforts would not make much sense.
3. No more than one pole at zero is allowed. Otherwise, the phase of the system would start off less than -180 degrees.

8.3.2 Specifications' validity and revisions

There are a total of five different types of specifications that can be provided. Each engine except 'Not Specified' allows the user to specify some subset of these requirements. The choice of available specifications varies across the different engines. It is based on whether the particular engine has the ability to make any changes in favor of the particular requirement.

Before the engine attempts to find a compensator that fits the requirements, it tests whether meeting this requirement is at all feasible in terms of retaining system stability. If it is determined that the specification is valid, the program proceeds with the compensation process. If not, the specification is revised or marked as invalid, depending on whether there is an unbiased way to adjust it.

Some basic validity tests, such as for negative or invalid inputs, are already tested for by the GUI and do not need to be performed again. However, tests that require a comparison with the plant or an examination of the compensator are performed by the compensating engine. Validity tests for the available specifications along with potential revisions are listed below.

The phase margin specification The phase margin specification is tested for being in the appropriate range, which is between 0 and some maximum value, usually around 90^o, appropriate for the particular engine used. If the specification is not in the range, it is revised to 45^o which is an average value guaranteeing a reasonable level of stability.

The crossover specification The crossover specification is tested by examining the compensator's ability to meet the specification while still retaining stability. This is tested by evaluating the plant's phase and the average phase of the compensator at the specified frequency. If the sum of the two is below the phase margin required for stability, the crossover specification is revised so as to guarantee approximately 45^o of phase margin.

The steady-state error specification The steady-state error specification can be given for either a ramp or a step input. The only case that is being tested for is a requirement for an error of zero when the input is a ramp. Since this specification implies two poles at zero, which *MASCoT* does not support, such a requirement is considered invalid.

The low-frequency error magnitude specification The low-frequency error magnitude requires that the gain magnitude of the loop transmission be above a certain value. The testing performed examines the frequency specified and its distance from an approximated value of the crossover. The crossover is approximated by finding the frequency where the compensated system remains stable. If the specification frequency is not sufficiently far below the crossover, the specification is deemed invalid and ignored. No revisions are made, since the choice of frequency is entirely dependent on the user.

The high-frequency closed-loop magnitude specification The closed-loop magnitude specification puts a limit on the magnitude of the loop transmission. The user specifies the closed-loop magnitude of the resulting system and the frequency and the minimum frequency at which this specification must be met. The closed-loop response of the system must remain below the user-provided magnitude at all frequencies above the frequency specified by the user. The validation of this specification determines whether the specification frequency is above the crossover value that ensures stability. If the distance from the crossover is too small, the specification is invalidated and ignored.

8.3.3 Usage of the specification's rank

When providing a specification, the user has the choice of ranking the specifications in order of significance. If no rank is selected, all the specifications are weighted equally, however, a predetermined sequence of handling is followed. The predetermined sequence considers phase margin to be the most important specification, followed by the crossover, the steady-state error, the low-frequency error, and the high-frequency closed-loop magnitude specification.

The specification's rank is used in two parts of the compensation process. One use of the rank is to adjust the size of the steps taken when two different compensator solutions are converged into one. The specification with the higher rank adjusts its solution slower toward the solution of the specification with the lower rank, which makes larger steps. This results in the degree of the tradeoff being proportional to the ranking of the specification.

Another use of the rank is the placement of the specification in the appropriate place in the convergence sequence. As mentioned in section 8.2.2, at any one time solutions from only two specifications are converged. The convergence process starts with the specifications ranked to be of the lowest importance. This results in the most important specification having the most weight in the final solution, since its compensator model is directly converged with the solutions resulting from the previous steps.

8.4 The Proportional Engine

The Proportional Engine is the simplest of all of the engines because of the limited number of degrees of freedom possible in the solution. Proportional compensation implies changing only one variable, which is the gain of the system. See section 4.3.1 for more information.

When the user selects the proportional compensation mode, the allowed specifications are either the phase margin or the crossover. The engine is provided with one of the specifications and the plant model. The plant model is tested for validity as described in section 8.3.1. If the plant is deemed invalid, the engine returns the error message for the GUI to display back to the user.

If the plant model is valid, the engine proceeds with the rest of the compensation routine. If the phase margin specification is selected, the engine calculates the frequency at which the plant has the specified phase margin. The compensator is equal to the inverse of the magnitude of the plant's loop transmission at this frequency. When the user chooses to specify the crossover instead, the compensator's magnitude is the inverse of the magnitude of the plant at the specified crossover frequency.

8.5 The Dominant Pole Engine

The Dominant Pole Engine implies inserting a low-frequency dominant pole, which dominates the time response of the closed loop system, as described in section 4.3.2. There are two degrees of freedom available in this mode, since both the gain and the location of the dominant pole can be adjusted. When designing the compensator, I choose to always put a pole at zero. This practice is the most common in most of the dominant pole compensation designs, and it simplifies the problem by reducing the number of degrees of freedom.

The user inputs the plant model and the appropriate specifications. In this mode, the user can specify either the phase margin or the crossover as well as the desired magnitude of the steady-state error. The user can rank the two specifications which determines how much they influence the properties of the final compensator.

The sequence of steps followed by the engine is described in section 8.2.2. The plant and the specification are evaluated for validity. If the plant model is invalid, the control is returned back to the GUI with an error message. If the specifications are incorrect, they are either revised appropriately or invalidated.

The steady-state error specification dictates the magnitude of the system at DC. If the steady-state error specification for a step input is zero, the resulting system must have a pole at DC. Plants that already have a zero-frequency pole are recognized and not compensated since they are not meant to be compensated with a dominant pole. The resulting system would have more than one pole at zero.

Since the location of the dominant pole is fixed at DC, only one degree of freedom remains, which is the gain. Depending on the specification, the appropriate gain is computed. If the specification is the phase margin, the engine finds the location where the combination of the phase of the plant and the dominant pole compensator equals the specified phase margin. The phase of the dominant pole compensator is -90^o everywhere since it only has one pole at zero frequency. The gain of the compensator is the inverse of the product of the magnitude of the plant and the dominant pole.

The crossover specification is even simpler. After determining that the desired crossover results in a stable system, the plant's magnitude combined with the pole at zero is evaluated. The gain is the inverse of this magnitude. The steady-state error specification for a step is automatically met since the error is always zero in case of a step input. If the input is specified to be a ramp, the magnitude of the compensator is the inverse of the error magnitude.

As a result of satisfying each specification, the engine comes up with three or fewer different solutions. These solutions are then converged using the ranking of each specification as the determining factor of how much its compensator will be altered to converge with the other compensator. In some cases, both specifications are improved with one of the choices of the compensator. In such cases, the better compensator is chosen. The sequence of converging solutions is followed.

The usage of the specification's rank is described in more detail in section 8.3.3. If no rank is specified, the default sequence of specifications' fulfillment is followed. In the case of a dominant pole compensator, the first specification is either the crossover or the phase margin, and the second is the steady-state error. If no specifications are present, the dominant engine sets the default specification, which is obtaining 45^o of the phase margin.

8.6 The Lag Engine

The Lag Engine has three degrees of freedom in the locations of the zero, of the pole and the gain of the compensator. The degrees of freedom are restricted in that the network has to be below the crossover frequency yet high enough above the low frequencies, as described in section 4.3.3.

Since lag compensation is used primarily to retain high frequency magnitudes and obtain good noise rejection, only some specifications are relevant and can be provided by the user. The lag compensation does not improve the crossover of a system, thus this specification is omitted. The user can specify the phase margin, the low-frequency error magnitude, or the closed-loop magnitudes at high frequencies. The specifications may be ranked by the user. Otherwise, the standard ordering is followed, with the phase margin first, the low-frequency error second, and the closed-loop high frequency magnitude third. As before, the validity of the plant model is an obvious requirement to proceed.

When implementing the lag engine, I reduce the number of degrees of freedom and use the most commonly used values for some of the parameters of the lag compensator. There are three degrees of freedom to begin with: the ratio of the location of the zero to the pole, α, the ratio of the location of the crossover to the zero, and the gain of the compensator. I fix the zero location to always be a factor of 10 below the crossover, which is a common practice among feedback system designers.

All the provided specifications are examined to see whether they are valid, and revised if necessary as described in section 8.3.2. The approach here is similar to the one in the case of the dominant pole compensation where each specification comes up with a compensator model, which allows the compensated system to minimally meet the specification.

For the phase margin specification, I assume that the phase due to the lag network contributes -6^o at the crossover, which is approximately the value if the lag zero is a factor of 10 below the crossover. For this specification, I fix α at 7, since this is a commonly used value for α. The gain is computed such that the compensated system's loop transmission must be unity at the required phase margin.

The low-frequency error specification manipulates both the gain and the value of α. First, the engine comes up with a standard model of a lag compensator that ensures a phase margin dictated by the phase margin specification is met, or, if not provided, 45^o. The value of the error is computed for that particular model. If the error is higher, the value of α is adjusted to its maximum value of 10 and the new system is tested for the error. If the error is met the value of α is lowered until the resulting error closely matches the specification. If the specification error is not met by adjusting α, the maximum value of alpha is retained and the crossover is pushed into the higher frequencies. The limit on the crossover location is that the compensated system's phase margin must be greater than 30^o.

With crossover pushed into the higher frequencies, the error magnitude is evaluated again. If the system meets the requirements, the crossover is pushed back until the computed error closely matches the specification. Otherwise, the compensator model for this specification has α of 10 and phase margin of 30^o.

The high-frequency magnitude specification is met by adjusting the crossover frequency until the specification is met. The limit on the adjustment is that the new crossover cannot be more than a factor of 10 below the crossover for the specified or default value of the phase margin.

After all of the specifications have their respective compensator models, the models are converged in a similar fashion to the one described in section 8.5. The minor difference is that the value of alpha is adjusted to the one determined by the low-frequency error specification.

8.7 The Lead Engine

The Lead Engine is also equipped with three degrees of freedom, as described in section 4.3.4. A lead network is based on having the zero "lead" before the pole, thus allowing for some positive phase margin in the vicinity of the network. The lead network is usually placed near the crossover frequency. Since it is not anywhere close to the low frequency region, it cannot increase the DC gain. The specifications allowed for this compensation method reflect that.

The allowed specifications include the phase margin, the crossover frequency, and the magnitude of the error for a given range of frequencies which are higher than zero. Applying lead compensation can improve these system properties.

To reduce the complexity of the computations, I limit the number of degrees of freedom available. A feedback designer can vary the relative location of the lead network with respect to the crossover, the ratio between the location of the pole and the zero, which is called α, and the gain of the system. I standardize the compensator to place its singularities so that the system's crossover is the geometric mean between the lead singularities. I also fix the value of α to 10.

The user specifies the plant model and none or all of the specifications. The plant is tested for validity as described in section 8.3.1. Similarly, the specifications are examined. If deemed invalid, they are revised or marked to be ignored.

The approach is similar to the one followed earlier, with each specification coming up with the respective model of the system. The models are then merged in the same manner as described in the previous sections.

8.8 The 'Not Specified' Engine

The 'Not Specified' Engine has the most degrees of freedom, since any reasonable combination of poles and zeros is allowed in the compensator. The user can provide any of the specifications listed above. First, the engine validates the system and the specifications. In this case, the specifications are evaluated not only in view of the plant's model but also against each other. Based on the resulting set of valid specifications, the engine shapes the compensated system's gain plot so as to meet all of the specifications. The compensator is the result of the comparison of the gain plot of the original plant and of the compensated system.

8.9 The 'Intelligent' Engine (IntelliEng)

The 'Intelligent' Engine is an alternative approach I am currently working on and have had some success with. IntelliEng is not a compensating tool by itself, it uses one of the other engines to compensate. Based on the user specifications, IntelliEng picks the most appropriate engine of the other four compensators to handle a given compensation problem.

The judgment as to which engine is the most appropriate is based on the plant's model as well as the type of specifications provided. For example, if the plant has a series of closely spaced poles at high frequencies, the dominant-pole compensation is the right approach. If the specification calls for an increase in the phase margin and crossover, the lead engine works the best. The approach seems to work reasonably well in many typical cases, but fails when the system exhibits a large diversity in its behavior.

Chapter 9

Sample Runs of MASCoT

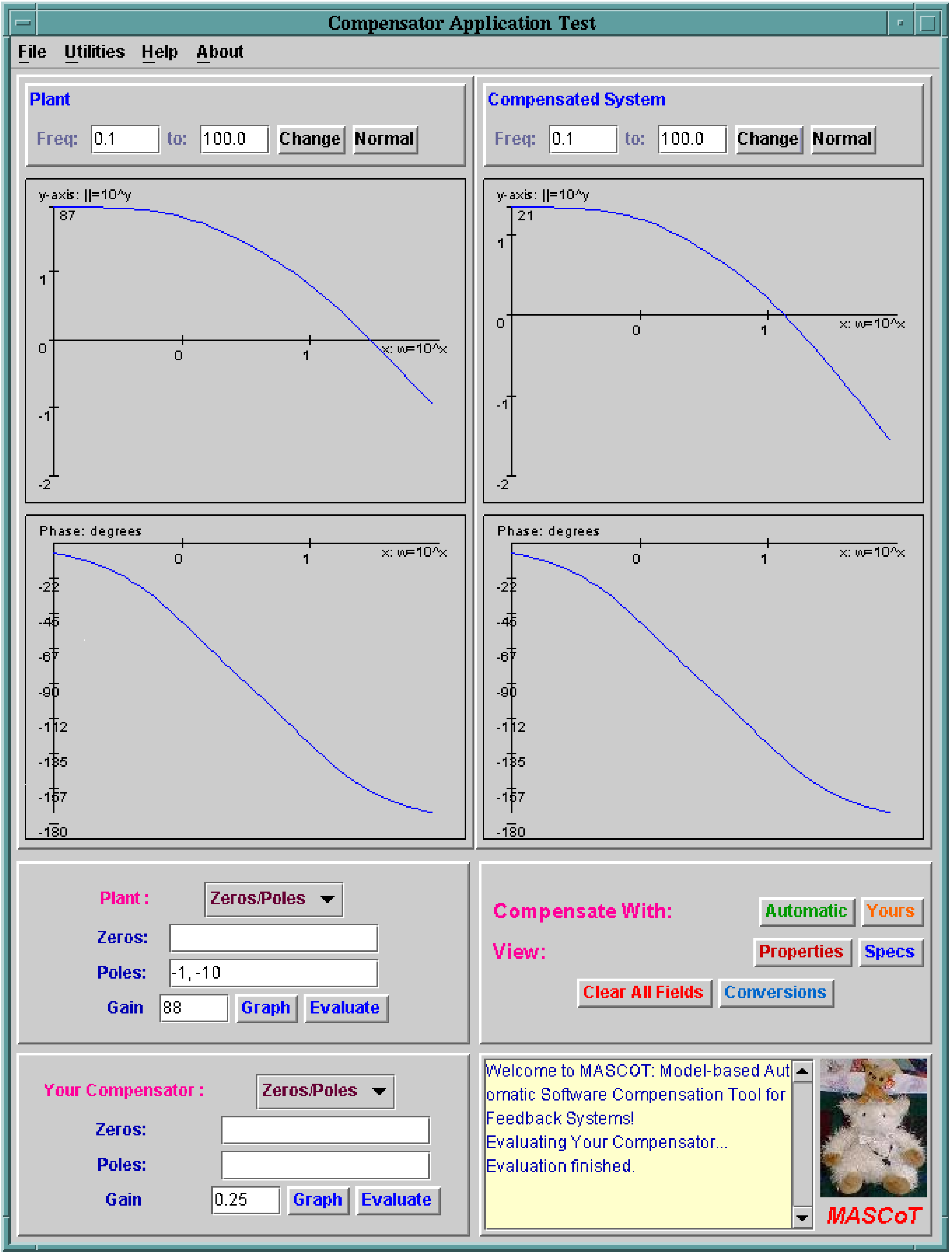
Few things are harder to put up with than a good example.

Mark Twain

Sample results of running *MASCoT* are presented in this chapter. The following examples are for the evaluator and for the automatic compensator mode. Because the toolkit handles linear minimum phase systems with real-valued poles and zeros, only this type of inputs is shown. *MASCoT* can run on any platform, as well as in a browser window. However, in order to streamline the process, all the examples in this chapter are run as applications on a Sun/Solaris platform.

9.1 Evaluating Your Compensator

In the "evaluate your own compensator" mode, the user specifies the plant and the compensator models. After entering the pole, zero, and gain information, the user presses the *Yours* button on the primary *MASCoT* interface and the system graphs the original and the compensated system, and launches a frame containing the plant, the compensator, and the compensated system properties. Figures 9-1, 9-2, 9-3, 9-4 show frames corresponding to two different runs in the evaluator mode. The first two figures are an example of proportional compensation, the latter two of lead compensation.



Plant — Freq: 0.1 to: 100.0

Compensated System — Freq: 0.1 to: 100.0

Plant : Zeros/Poles

Zeros:

Poles: -1, -10

Gain 88

Your Compensator : Zeros/Poles

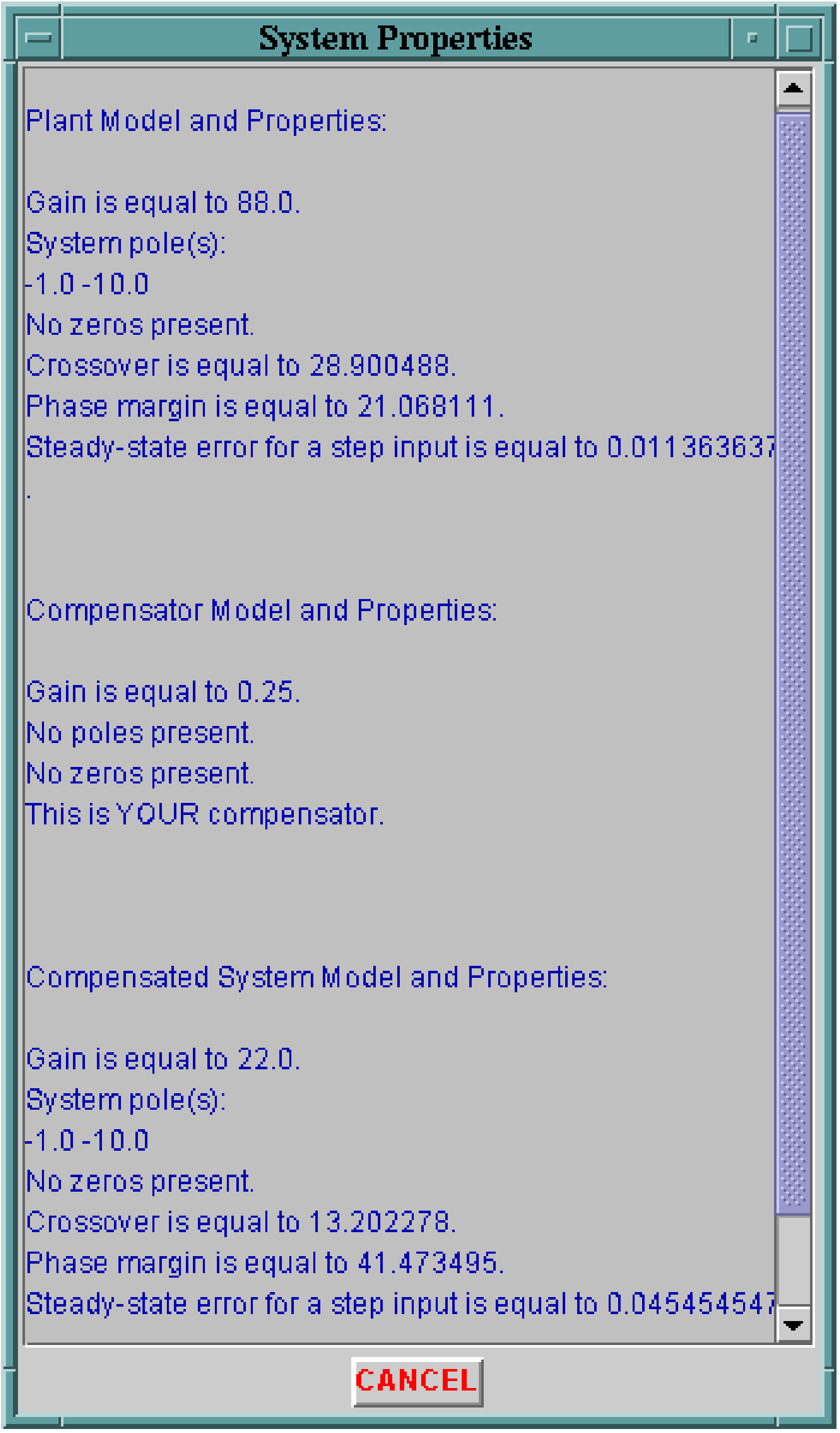
Zeros:

Poles:

Gain 0.25

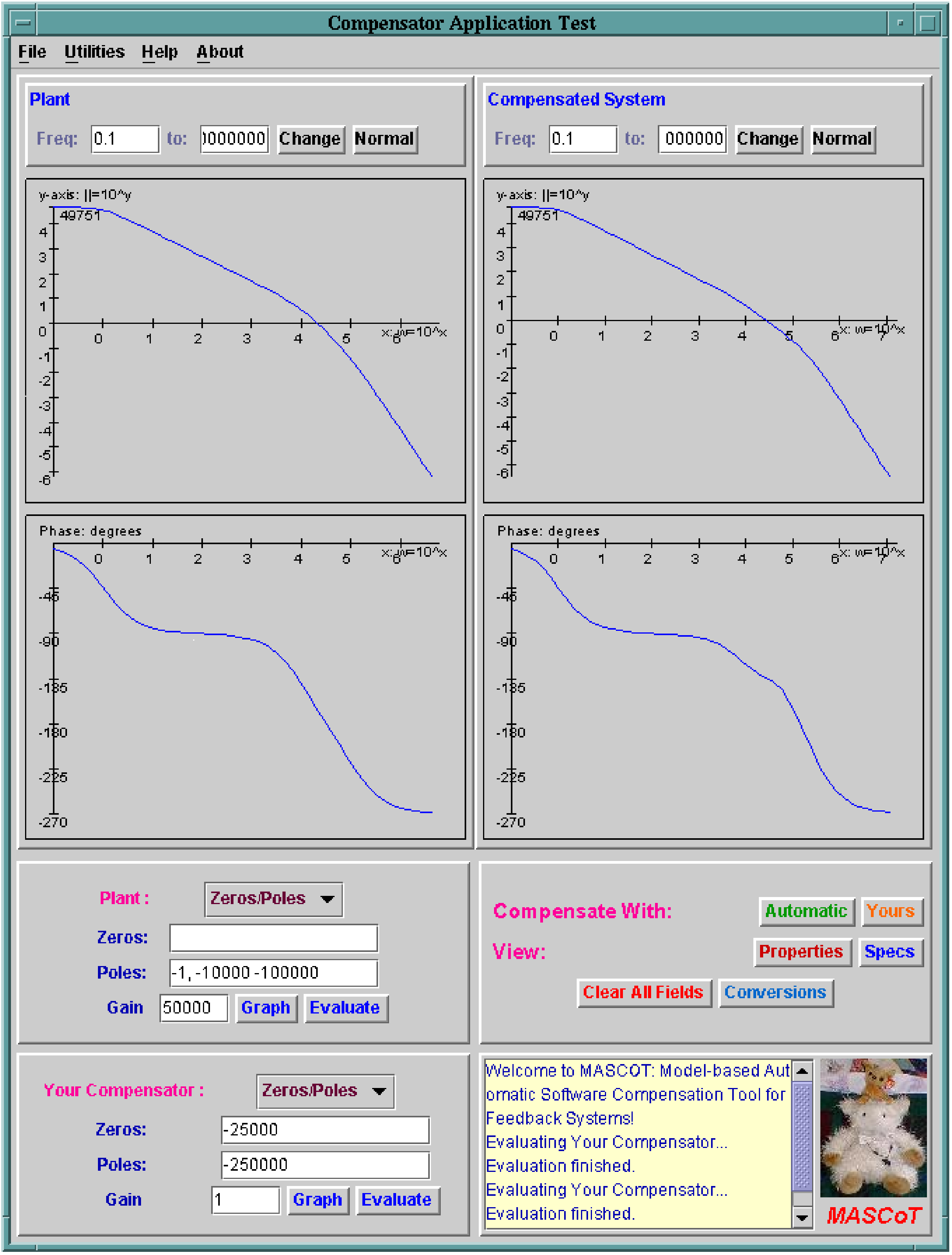
Welcome to MASCOT: Model-based Automatic Software Compensation Tool for Feedback Systems!
Evaluating Your Compensator...
Evaluation finished.

Figure 9-1: The primary *MASCoT* frame after running "your compensator" with the plant, $G_f(s) = \frac{88}{(s+1)(0.1s+1)}$, and the compensator, $G_c(s) = 0.25$.



```
Plant Model and Properties:

Gain is equal to 88.0.
System pole(s):
-1.0 -10.0
No zeros present.
Crossover is equal to 28.900488.
Phase margin is equal to 21.068111.
Steady-state error for a step input is equal to 0.011363637
.


Compensator Model and Properties:

Gain is equal to 0.25.
No poles present.
No zeros present.
This is YOUR compensator.



Compensated System Model and Properties:

Gain is equal to 22.0.
System pole(s):
-1.0 -10.0
No zeros present.
Crossover is equal to 13.202278.
Phase margin is equal to 41.473495.
Steady-state error for a step input is equal to 0.045454547
```

Figure 9-2: The system properties frame accompanying the primary frame from the run in Figure 9-1



Plant — Freq: 0.1 to:)000000

Compensated System — Freq: 0.1 to: 000000

Plant : Zeros/Poles

Zeros:

Poles: -1, -10000 -100000

Gain 50000

Your Compensator : Zeros/Poles

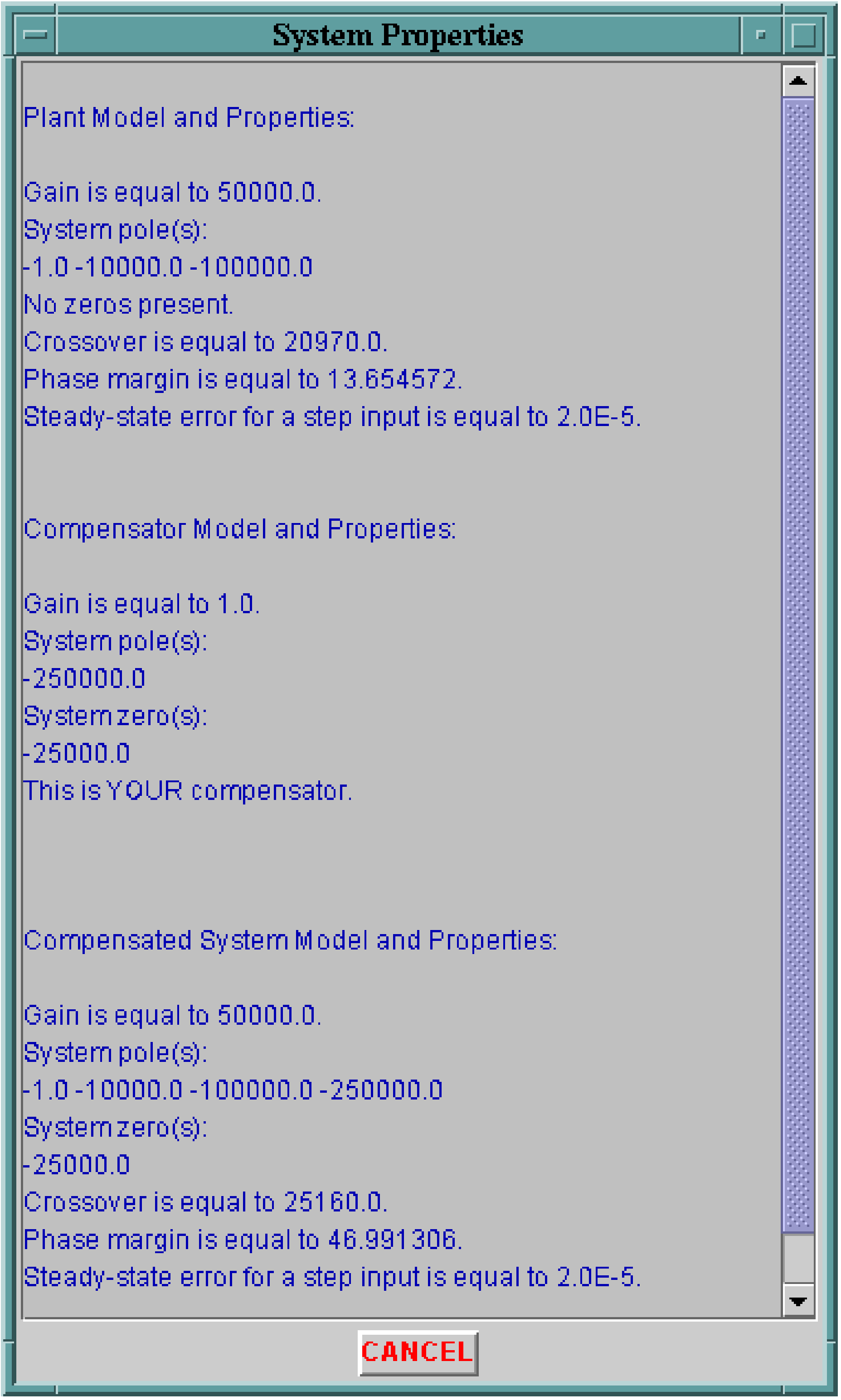
Zeros: -25000

Poles: -250000

Gain 1

Welcome to MASCOT: Model-based Automatic Software Compensation Tool for Feedback Systems!
Evaluating Your Compensator...
Evaluation finished.
Evaluating Your Compensator...
Evaluation finished.

Figure 9-3: The primary *MASCoT* frame after running in "your compensator mode" with the plant, $G_f(s) = \frac{5*10^4}{(s+1)(10^{-4}s+1)(10^{-5}s+1)}$, and the compensator, $G_c(s) = \frac{4*10-5s+1}{4*10^{-6}s+1}$



```
Plant Model and Properties:

Gain is equal to 50000.0.
System pole(s):
-1.0 -10000.0 -100000.0
No zeros present.
Crossover is equal to 20970.0.
Phase margin is equal to 13.654572.
Steady-state error for a step input is equal to 2.0E-5.


Compensator Model and Properties:

Gain is equal to 1.0.
System pole(s):
-250000.0
System zero(s):
-25000.0
This is YOUR compensator.



Compensated System Model and Properties:

Gain is equal to 50000.0.
System pole(s):
-1.0 -10000.0 -100000.0 -250000.0
System zero(s):
-25000.0
Crossover is equal to 25160.0.
Phase margin is equal to 46.991306.
Steady-state error for a step input is equal to 2.0E-5.
```

Figure 9-4: The system properties frame accompanying the primary frame from the run in Figure 9-3

9.2 Auto-compensating MEngines

This section presents a series of runs of the automatic compensator. Each run corresponds to a different engine, as selected by the user. In this mode, the user enters the plant model, the specifications to be met, and the type of compensation to be applied. The auto-compensator does the rest.

Proportional engine

Figures 9-5, 9-6, and 9-7 are on the following pages.

Dominant pole engine

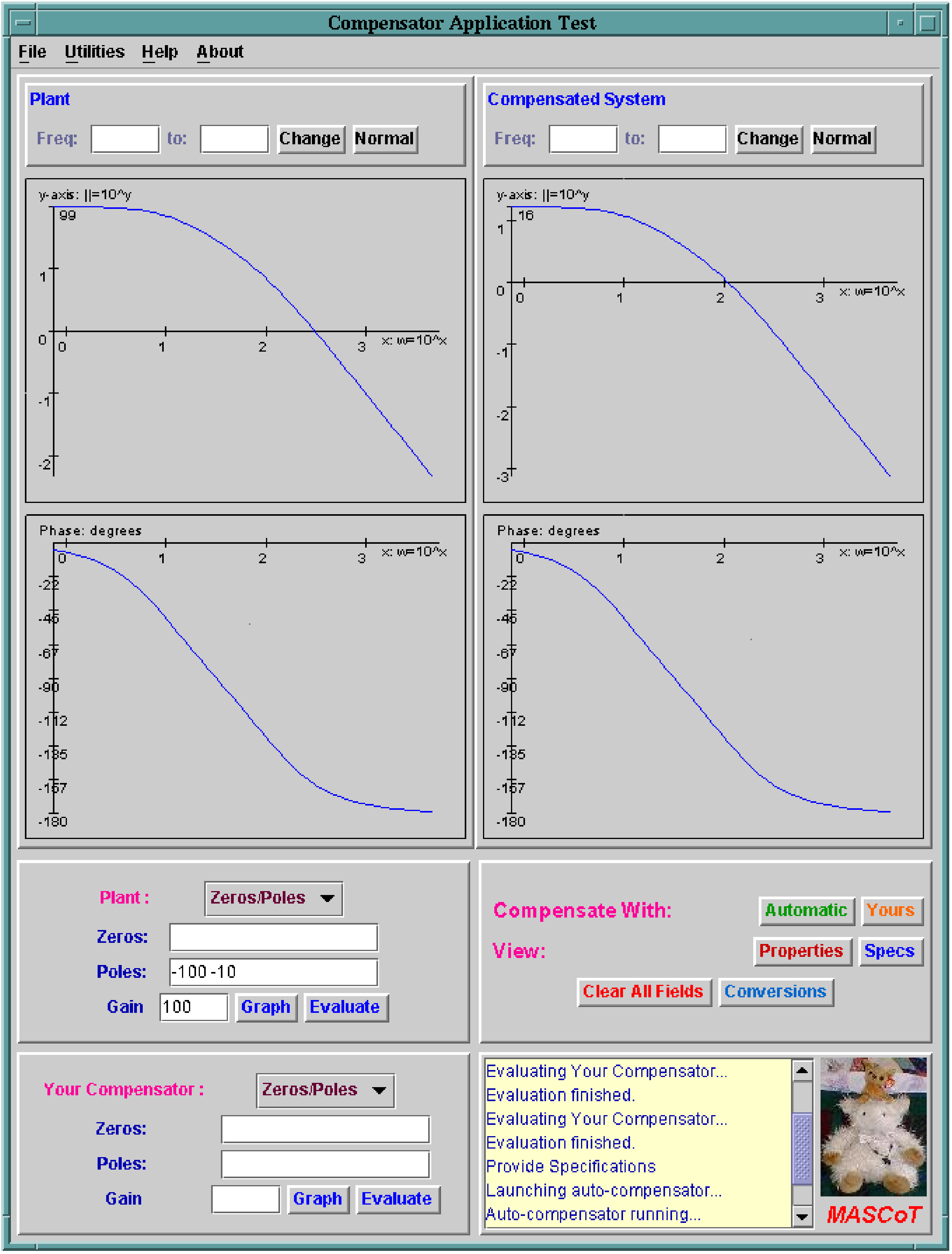
Figures 9-8, 9-9, and 9-10 are on the following pages.

Lag engine

Figures 9-11, 9-12, and 9-13 are on the following pages.

Lead engine

Figures 9-11, 9-12, and 9-13 are on the following pages.

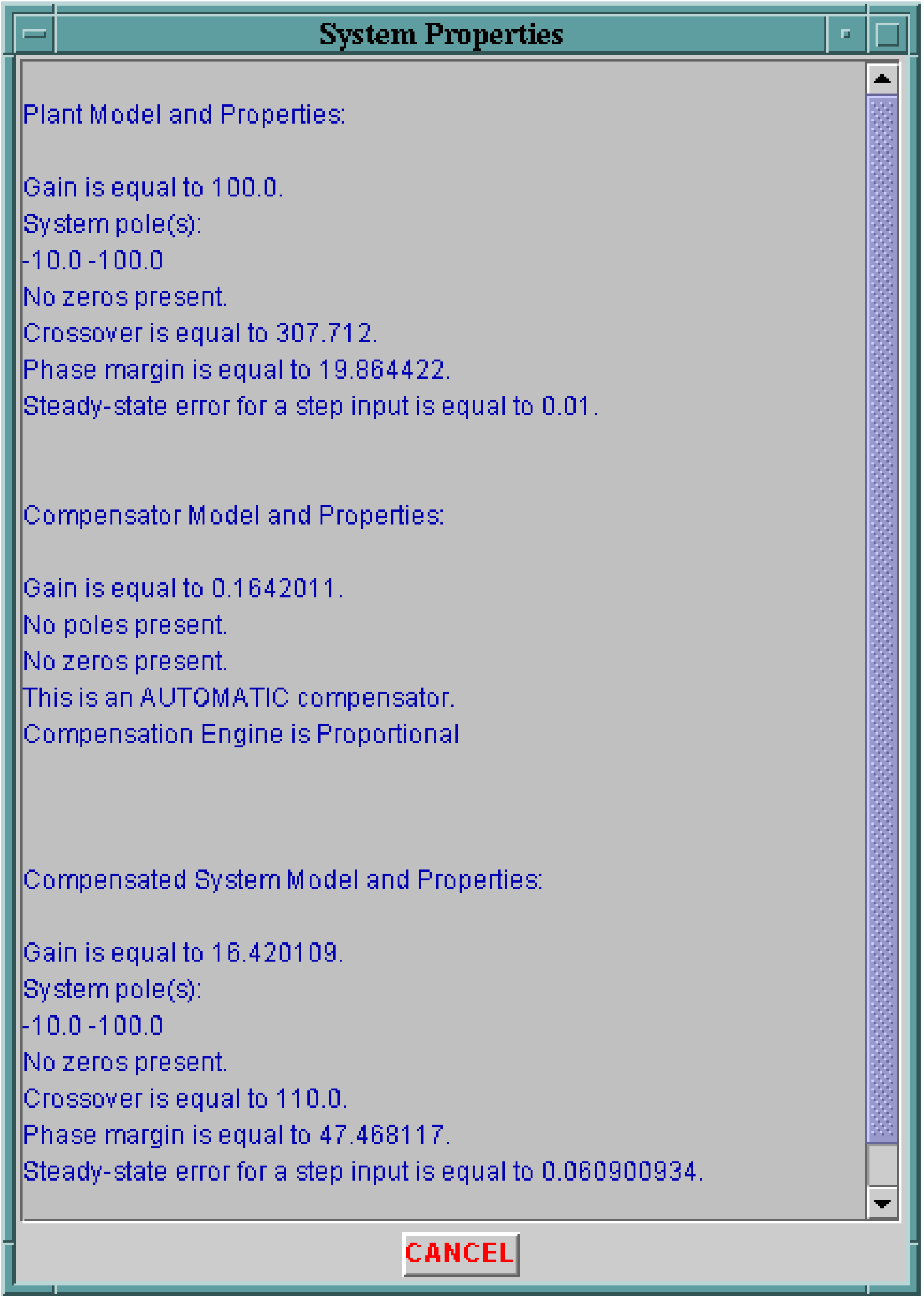


Poles: -100 -10

Gain 100

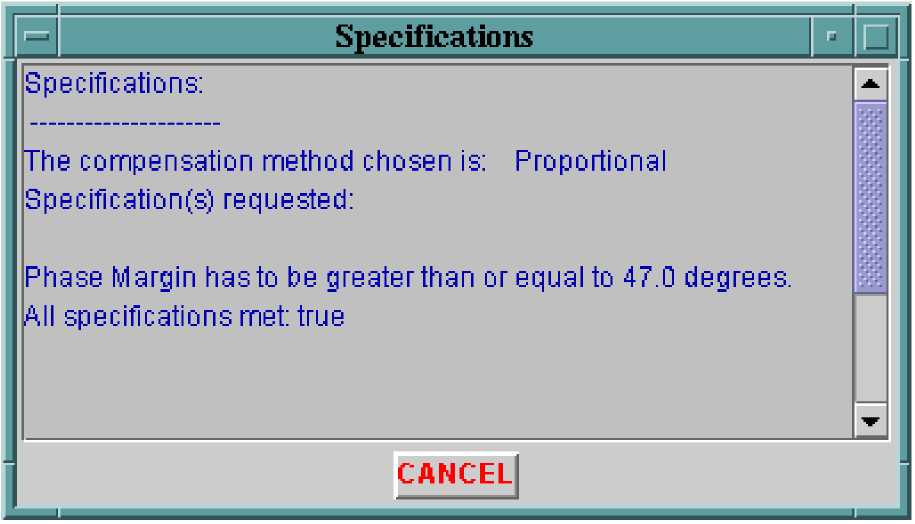
Evaluating Your Compensator...
Evaluation finished.
Evaluating Your Compensator...
Evaluation finished.
Provide Specifications
Launching auto-compensator...
Auto-compensator running...

Figure 9-5: The primary *MASCoT* frame after "automatic compensation" using the Proportional Engine. The plant model is $G_f(s) = \frac{100}{(10^{-1}s+1)(10^{-2}s+1)}$.



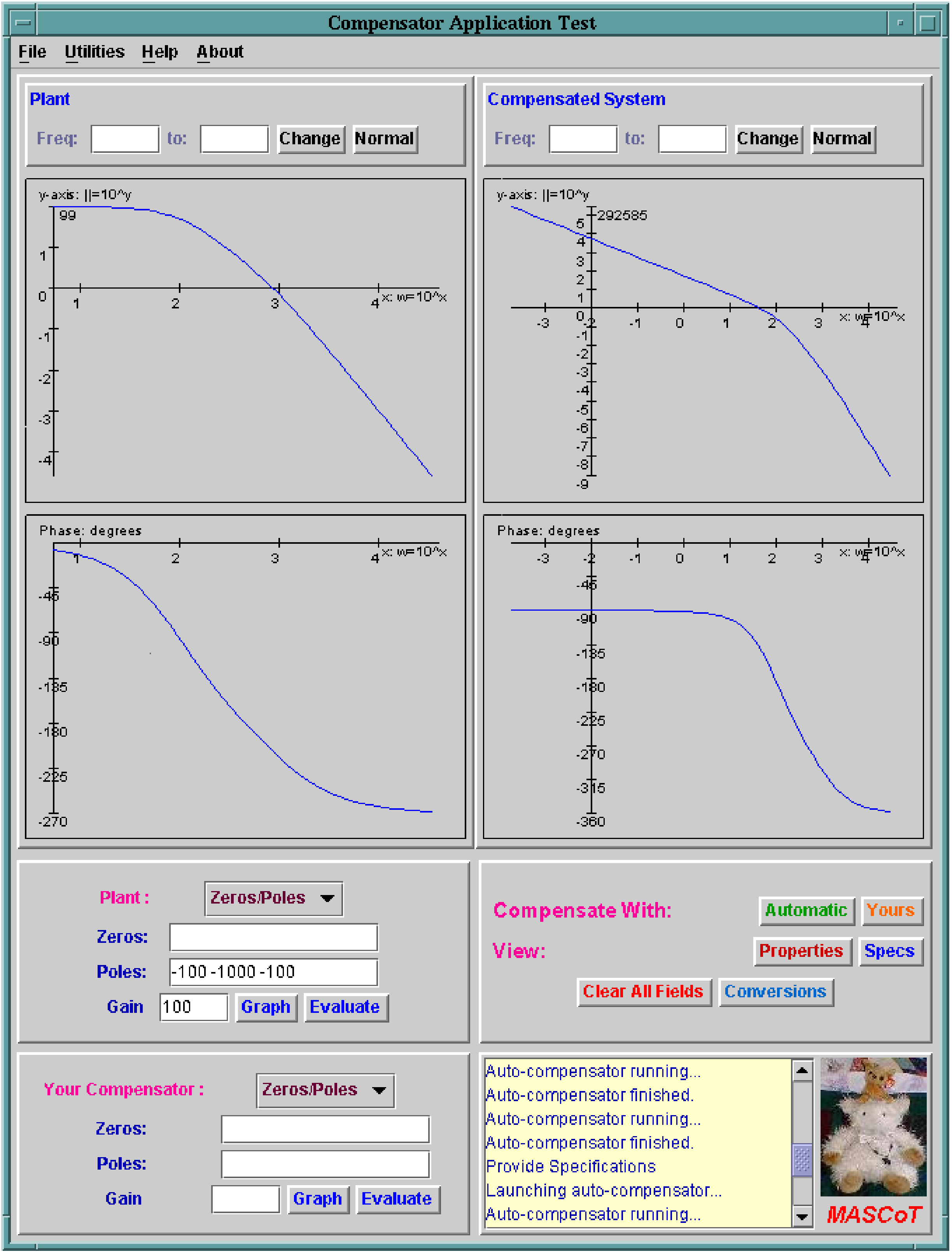
```
Plant Model and Properties:

Gain is equal to 100.0.
System pole(s):
-10.0 -100.0
No zeros present.
Crossover is equal to 307.712.
Phase margin is equal to 19.864422.
Steady-state error for a step input is equal to 0.01.


Compensator Model and Properties:

Gain is equal to 0.1642011.
No poles present.
No zeros present.
This is an AUTOMATIC compensator.
Compensation Engine is Proportional



Compensated System Model and Properties:

Gain is equal to 16.420109.
System pole(s):
-10.0 -100.0
No zeros present.
Crossover is equal to 110.0.
Phase margin is equal to 47.468117.
Steady-state error for a step input is equal to 0.060900934.
```

Figure 9-6: The system properties frame accompanying the primary frame from the run in Figure 9-5



```
Specifications:
 --------------------
The compensation method chosen is:   Proportional
Specification(s) requested:

Phase Margin has to be greater than or equal to 47.0 degrees.
All specifications met: true
```

Figure 9-7: The specification summary frame accompanying the primary frame from the run in Figure 9-5

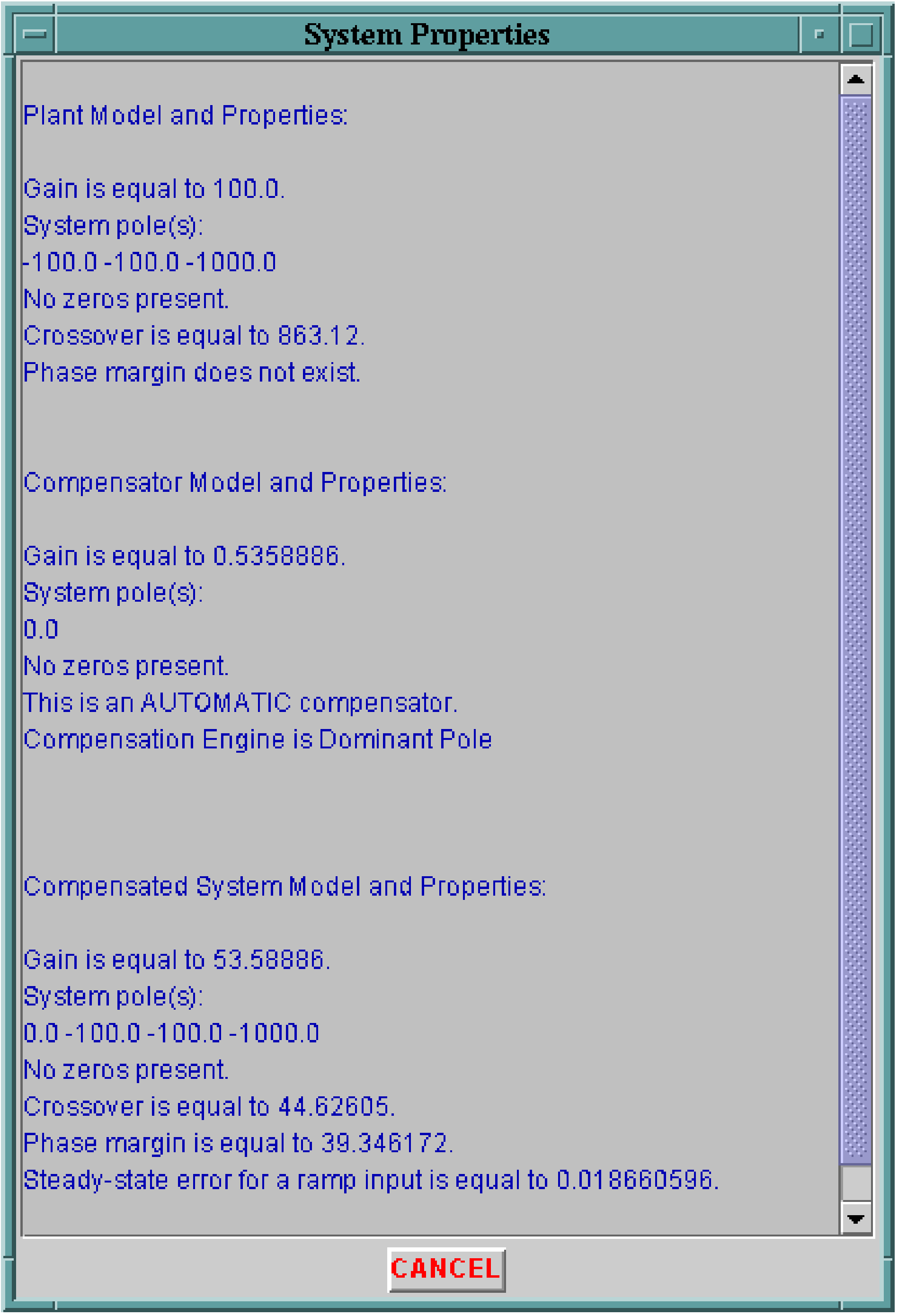


Poles: -100 -1000 -100

Gain: 100

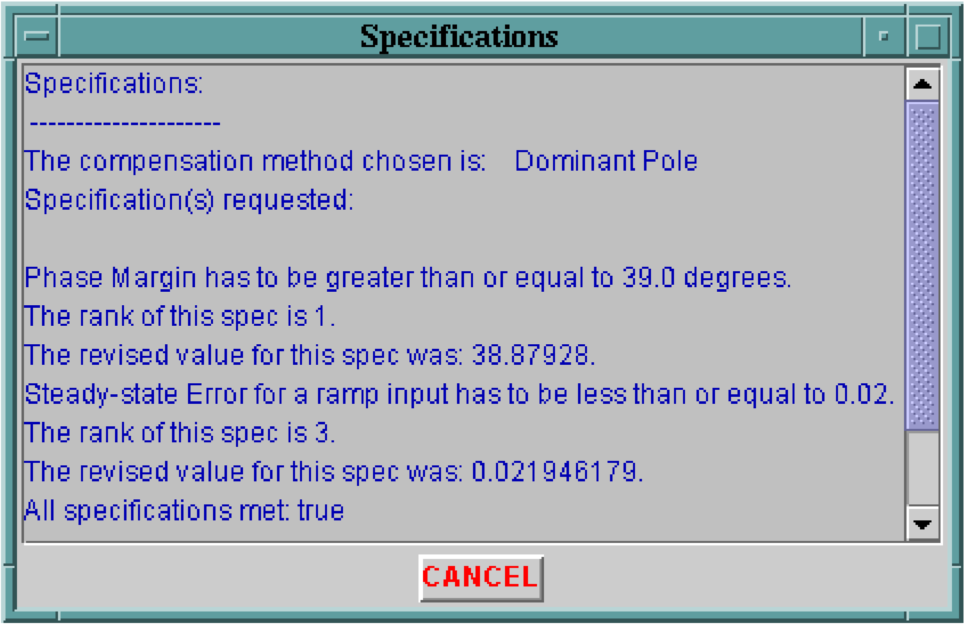
Auto-compensator running...
Auto-compensator finished.
Auto-compensator running...
Auto-compensator finished.
Provide Specifications
Launching auto-compensator...
Auto-compensator running...

Figure 9-8: The primary *MASCoT* frame after "automatic compensation" using the Dominant Pole Engine. The plant model is $G_f(s) = \frac{100}{(10^{-2}s+1)^2(10^{-3}s+1)}$.



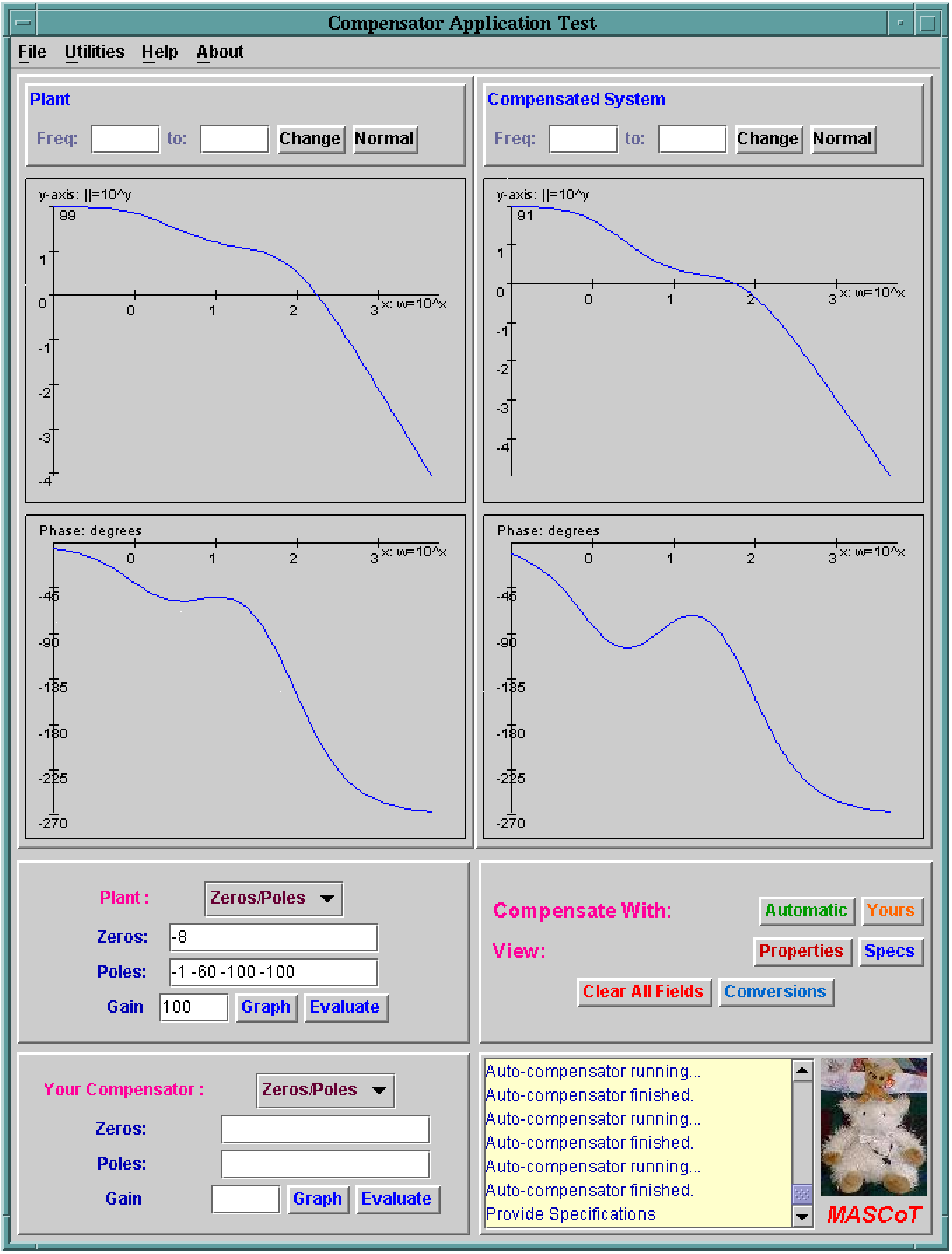
```
Plant Model and Properties:

Gain is equal to 100.0.
System pole(s):
-100.0 -100.0 -1000.0
No zeros present.
Crossover is equal to 863.12.
Phase margin does not exist.


Compensator Model and Properties:

Gain is equal to 0.5358886.
System pole(s):
0.0
No zeros present.
This is an AUTOMATIC compensator.
Compensation Engine is Dominant Pole



Compensated System Model and Properties:

Gain is equal to 53.58886.
System pole(s):
0.0 -100.0 -100.0 -1000.0
No zeros present.
Crossover is equal to 44.62605.
Phase margin is equal to 39.346172.
Steady-state error for a ramp input is equal to 0.018660596.
```

Figure 9-9: The system properties frame accompanying the primary frame from the run in Figure 9-8



```
Specifications:
 --------------------
The compensation method chosen is:   Dominant Pole
Specification(s) requested:

Phase Margin has to be greater than or equal to 39.0 degrees.
The rank of this spec is 1.
The revised value for this spec was: 38.87928.
Steady-state Error for a ramp input has to be less than or equal to 0.02.
The rank of this spec is 3.
The revised value for this spec was: 0.021946179.
All specifications met: true
```

Figure 9-10: The specification summary frame accompanying the primary frame from the run in Figure 9-8



Plant : Zeros/Poles

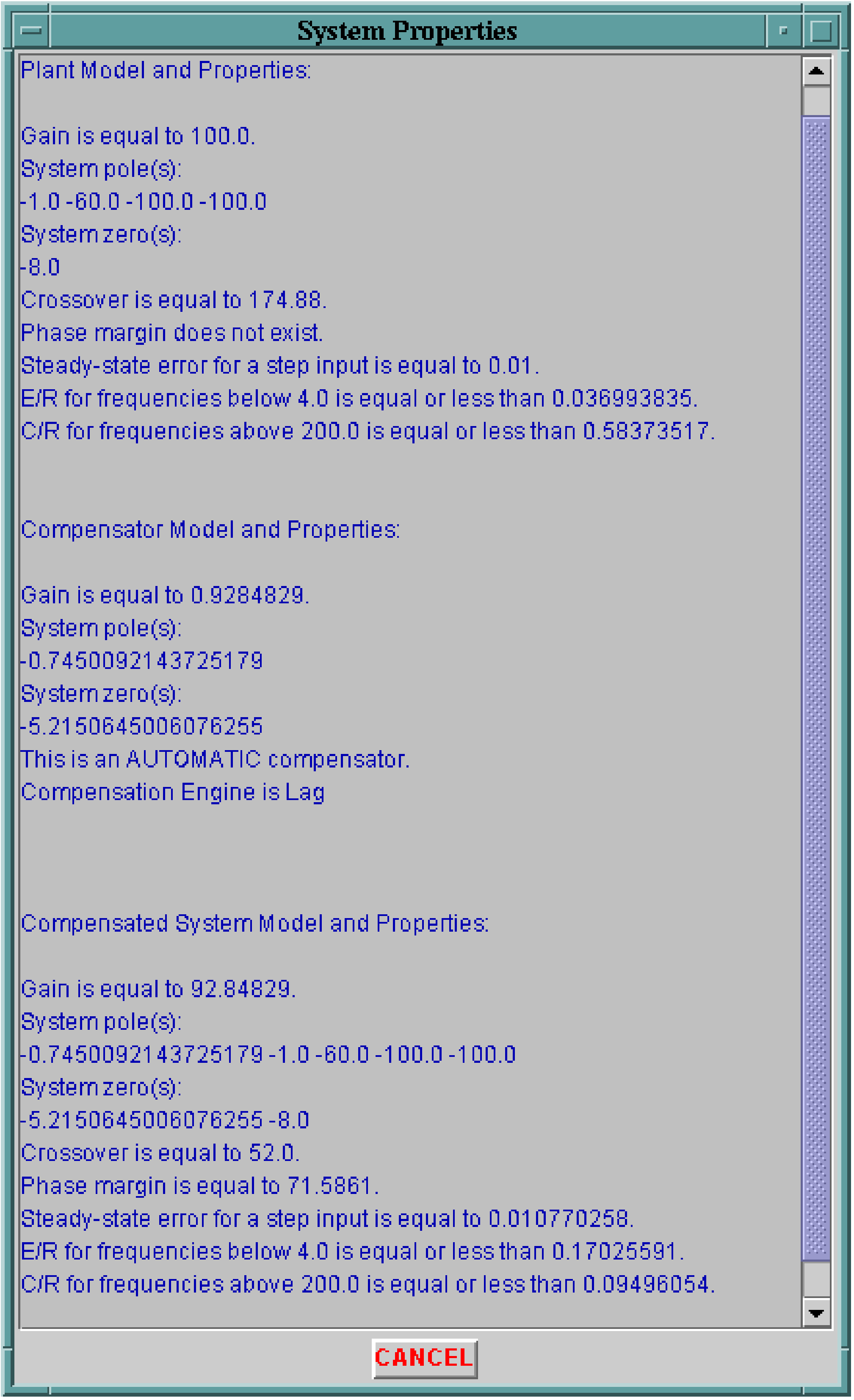
Zeros: -8

Poles: -1 -60 -100 -100

Gain 100

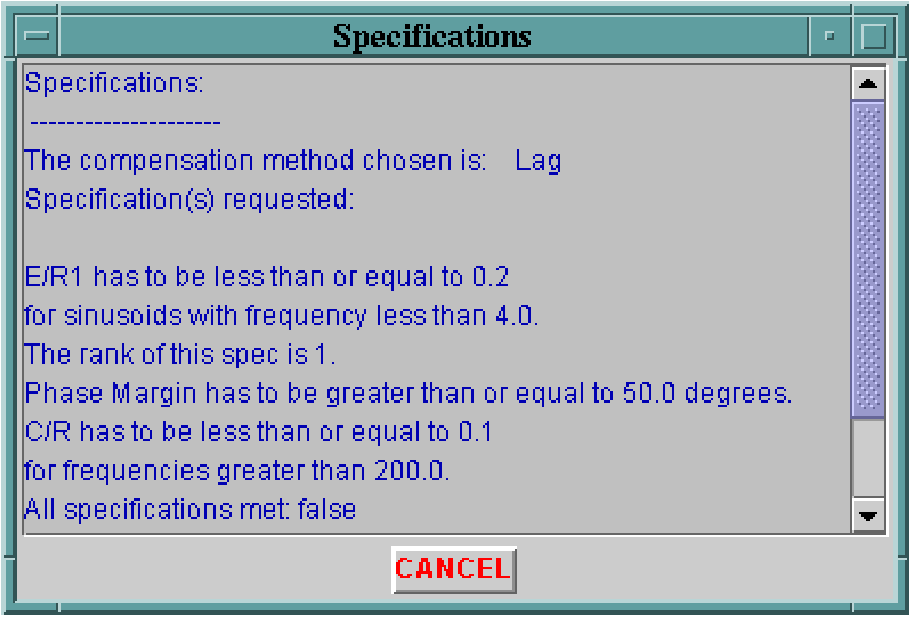
Auto-compensator running...
Auto-compensator finished.
Auto-compensator running...
Auto-compensator finished.
Auto-compensator running...
Auto-compensator finished.
Provide Specifications

Figure 9-11: The primary *MASCoT* frame after "automatic compensation" using the Lag Engine. The plant model is $G_f(s) = \frac{100(0.125s+1)}{(s+1)(\frac{1}{6}s+1)(10^{-2}s+1)^2}$



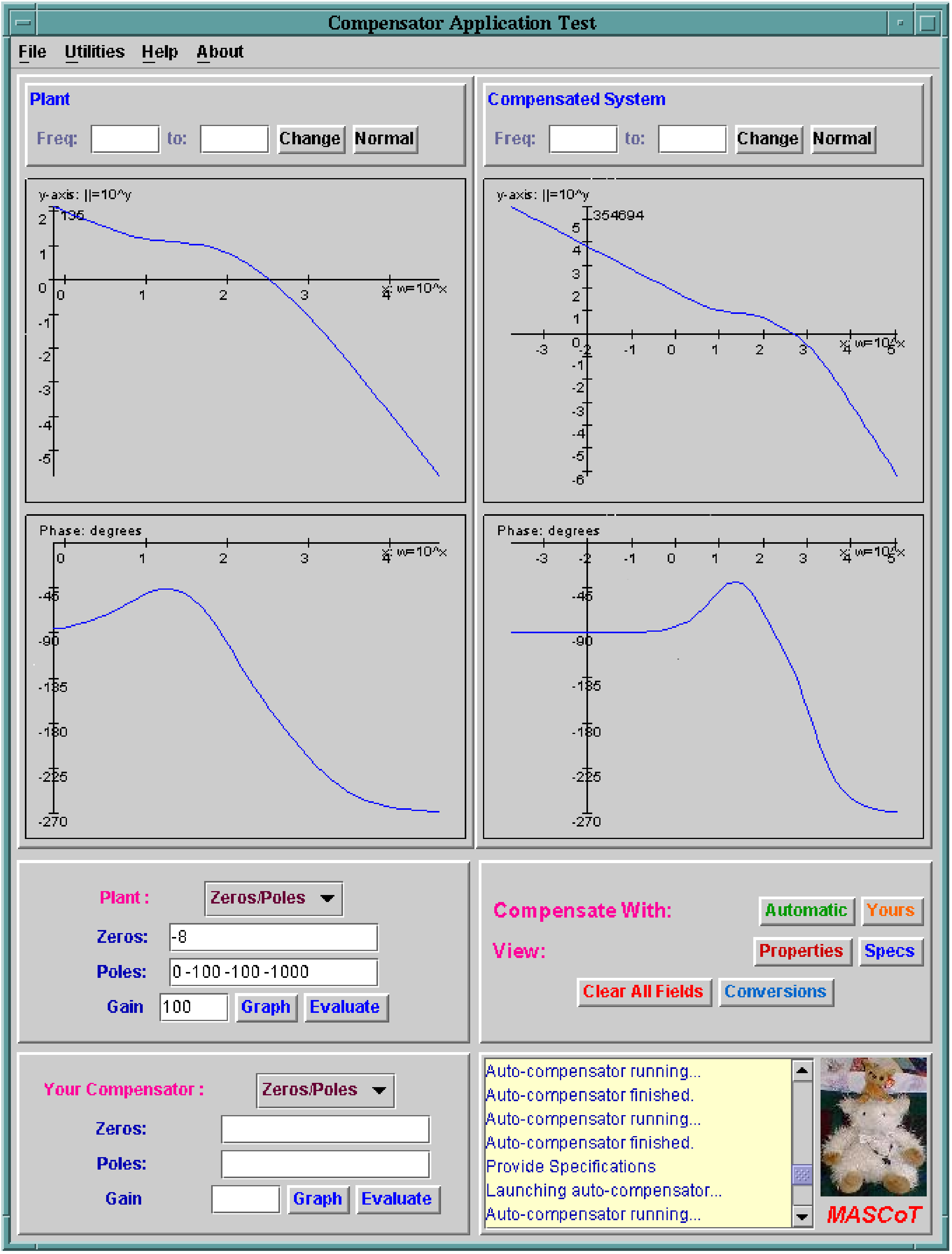
```
Plant Model and Properties:

Gain is equal to 100.0.
System pole(s):
-1.0 -60.0 -100.0 -100.0
System zero(s):
-8.0
Crossover is equal to 174.88.
Phase margin does not exist.
Steady-state error for a step input is equal to 0.01.
E/R for frequencies below 4.0 is equal or less than 0.036993835.
C/R for frequencies above 200.0 is equal or less than 0.58373517.


Compensator Model and Properties:

Gain is equal to 0.9284829.
System pole(s):
-0.7450092143725179
System zero(s):
-5.2150645006076255
This is an AUTOMATIC compensator.
Compensation Engine is Lag



Compensated System Model and Properties:

Gain is equal to 92.84829.
System pole(s):
-0.7450092143725179 -1.0 -60.0 -100.0 -100.0
System zero(s):
-5.2150645006076255 -8.0
Crossover is equal to 52.0.
Phase margin is equal to 71.5861.
Steady-state error for a step input is equal to 0.010770258.
E/R for frequencies below 4.0 is equal or less than 0.17025591.
C/R for frequencies above 200.0 is equal or less than 0.09496054.
```

Figure 9-12: The system properties frame accompanying the primary frame from the run in Figure 9-11



```
Specifications:
 --------------------
The compensation method chosen is:    Lag
Specification(s) requested:

E/R1 has to be less than or equal to 0.2
for sinusoids with frequency less than 4.0.
The rank of this spec is 1.
Phase Margin has to be greater than or equal to 50.0 degrees.
C/R has to be less than or equal to 0.1
for frequencies greater than 200.0.
All specifications met: false
```

Figure 9-13: The specification summary frame accompanying the primary frame from the run in Figure 9-11



Plant : Zeros/Poles

Zeros: -8

Poles: 0 -100 -100 -1000

Gain 100

Your Compensator : Zeros/Poles

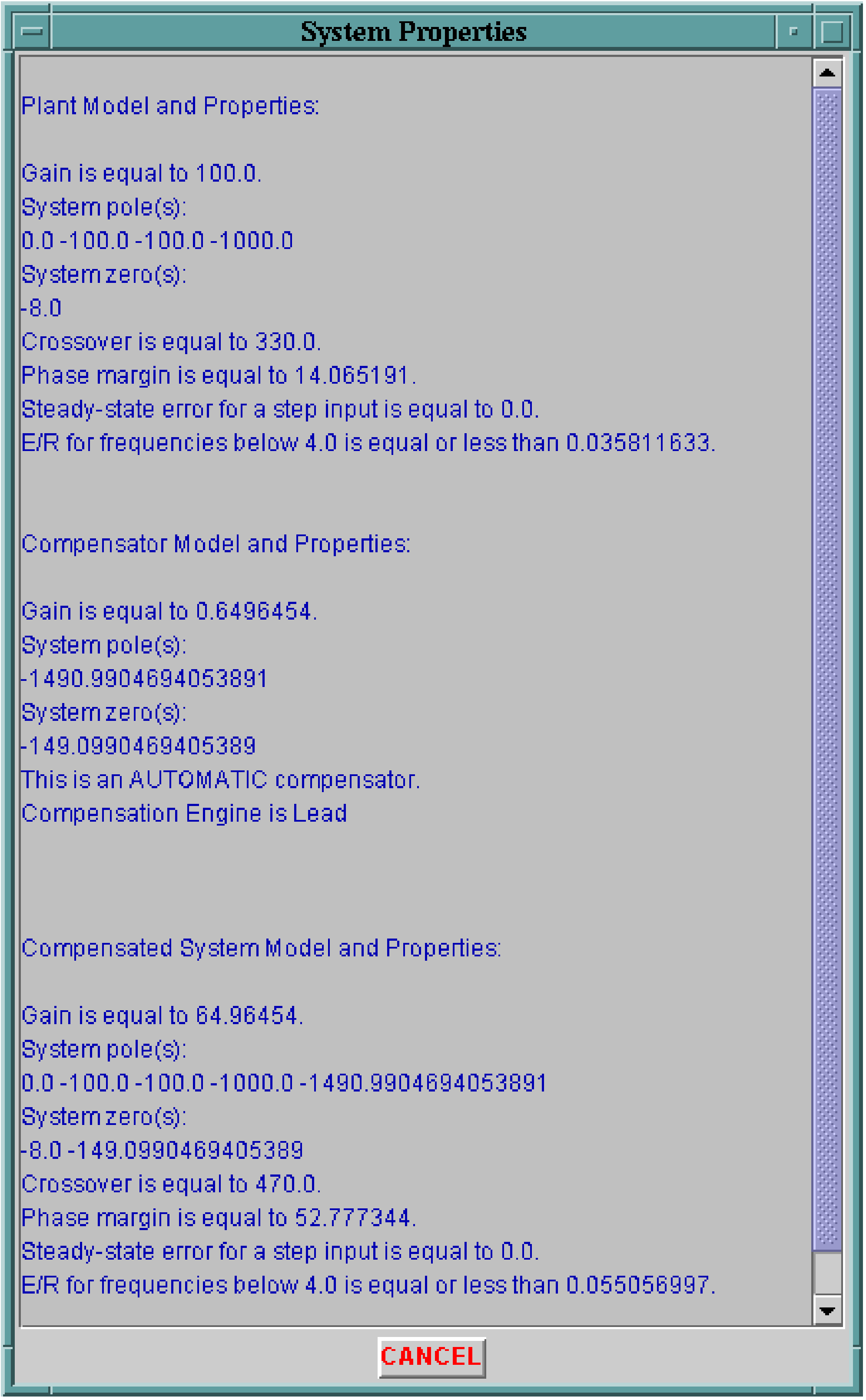
Zeros:

Poles:

Gain

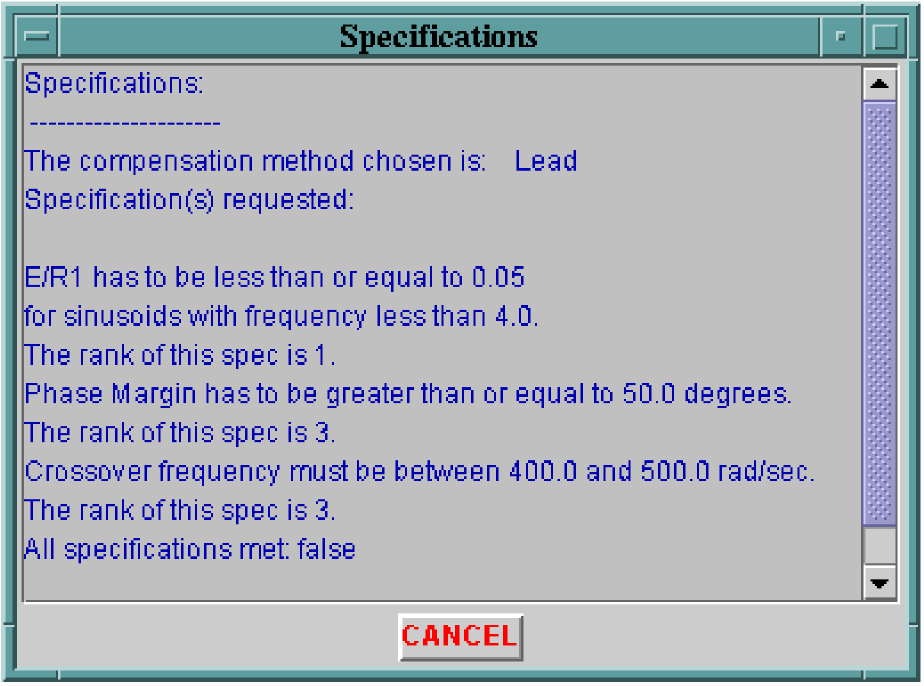
Auto-compensator running...
Auto-compensator finished.
Auto-compensator running...
Auto-compensator finished.
Provide Specifications
Launching auto-compensator...
Auto-compensator running...

Figure 9-14: The primary *MASCoT* frame after "automatic compensation" using the Lead Engine. The plant model is $G_f(s) = \frac{100(0.125s+1)}{s(10^{-2}s+1)^2(10^{-3}s+1)}$.



```
Plant Model and Properties:

Gain is equal to 100.0.
System pole(s):
0.0 -100.0 -100.0 -1000.0
System zero(s):
-8.0
Crossover is equal to 330.0.
Phase margin is equal to 14.065191.
Steady-state error for a step input is equal to 0.0.
E/R for frequencies below 4.0 is equal or less than 0.035811633.


Compensator Model and Properties:

Gain is equal to 0.6496454.
System pole(s):
-1490.9904694053891
System zero(s):
-149.0990469405389
This is an AUTOMATIC compensator.
Compensation Engine is Lead



Compensated System Model and Properties:

Gain is equal to 64.96454.
System pole(s):
0.0 -100.0 -100.0 -1000.0 -1490.9904694053891
System zero(s):
-8.0 -149.0990469405389
Crossover is equal to 470.0.
Phase margin is equal to 52.777344.
Steady-state error for a step input is equal to 0.0.
E/R for frequencies below 4.0 is equal or less than 0.055056997.
```

Figure 9-15: The system properties frame accompanying the primary frame from the run in Figure 9-14



```
Specifications:
 --------------------
The compensation method chosen is:   Lead
Specification(s) requested:

E/R1 has to be less than or equal to 0.05
for sinusoids with frequency less than 4.0.
The rank of this spec is 1.
Phase Margin has to be greater than or equal to 50.0 degrees.
The rank of this spec is 3.
Crossover frequency must be between 400.0 and 500.0 rad/sec.
The rank of this spec is 3.
All specifications met: false
```

Figure 9-16: The specification summary frame accompanying the primary frame from the run in Figure 9-14

Chapter 10

Future Extensions

The best way to predict the future is to invent it.

Alan Kay

10.1 Improvement on the Algorithms

Although I was able to get satisfying results with the compensators, there are other algorithms which I have not implemented but which could be tried in the future. These algorithms include:

1. The number of degrees of freedom available can be increased for some of the compensators. This would make *MASCoT* more flexible and would allow it to find more efficient solutions for some systems.

2. The convergence process between the compensator models from the different specifications can be made more complex and diversified depending on the types of the specifications which specify the particular compensators.

3. Instead of running a single sequence of compensation, multiple sequences can be performed with a different usage of the rank or conversion process. The results can be compared against one another based on how well they meet the specifications, such that the 'best' compensator in the set can be selected.

4. The validity process can be improved to find systems which are inherently inappropriate for this type of compensation, and in these cases, a different compensating engine can be used.

A more detailed analysis of the efficiency and speed of the algorithms could be performed in order to find places where the processor spends the most time computing. The most computationally intensive algorithms are responsible for about 80% of the processing time. As a result of this analysis I can make appropriate optimizations, and the compensator can come up with the same solution significantly faster. The user can run the program on a slower machine and obtain a solution in a reasonable amount of time.

Another possibility is to implement *multi-threading*. Multi-threading allows multiple *MASCoT* processes run in parallel. This can greatly reduce the time it takes to find a satisfying solution. I did not venture into multithreading now, since it makes the software significantly more complex and prone to bugs resulting in *deadlocks* or other types of unstable behavior. The main objective of my research was finding methods for automatic compensation, not optimization of algorithms or increasing the computational speed.

10.2 Changes in the GUI

Suggested extensions and/or improvements on the GUI design:

1. The *SystemPanel*, where the user inputs the system model, has currently two options available from the option bar. One option allows the modeling of a system by specifying poles, zeros, and the gain of a system. Another allows the modeling of a system by specifying the denominator and the numerator of the system transfer function. At this point the latter option is not implemented, since I haven't had time to implement solving for the roots of an equation.

2. The AWT (Advanced Windowing Toolkit) Java graphics components are currently intermixed with the Swing graphics components. The AWT package is the older graphics package, while the Swing GUI components are new graphics module libraries. Numerous problems with the proper display of components come up when the different types are mixed together. For example, the Canvas class is used to plot on and is an AWT component. The Swing JMenu objects cannot display their menus on top of this class, because they hide underneath.

This improvement would mean replacing any of the remaining AWT components by Swing components.

3. All of the user-defined classes are currently grouped in a single package, *thesis*. Separating the GUI, the Engines, and the System modules into separate packages will make handling the code a lot easier.

4. Some of the panels used in the GUI can be avoided, leading to a more stable and faster loading of *MASCoT*. Right now, I have many panels, each of which derives from different classes. A more compact version of *MASCoT* can have fewer panels and everything loaded into a single frame.

5. The Graphical User Interface could be designed to be more colorful and eye-pleasing.

10.3 Add Minor Loop Compensation

The current *MASCoT* Engines are capable of designing compensators which are in the forward path and in series with the controlled element. This is the simplest type of compensation; there is a direct relationship between the closed-loop and the open-loop singularities. However, there is another powerful method which can be used to compensate a system and which can lead to an even more stable behavior and thus a more robust system. It is a type of compensation called *minor loop compensation*, and would make a very useful extension to *MASCoT*.

Minor loop compensation involves putting the compensating element in both the feedback and in the forward path, and adding another feedback loop around the plant. This gives us more degrees of freedom and allows us to radically change the system's dynamics. It can also increase the frequency range in which the system remains stable. For a more detailed description of minor loop compensation, see section 4.4.

10.4 The Ability to Handle Nonlinear Systems

The current version of *MASCoT* is capable of compensating linear systems with monotonically decreasing amplitude and phase. In real life, there are many nonlinear and non minimum phase systems. *Pure time delay*, $(Ae^{-j\omega})$, is an example of a non minimum phase system. Its phase is a function of frequency and never stops decreasing.. An ability to handle some types of non minimum phase would be a useful extension. Nonlinear systems is yet another extension, however these are much more complex and require an amount of research suitable for a doctoral thesis.

10.5 Bug Fixes

Although *MASCoT*'s modules have been tested throughout the implementation and integration stages, a rigorous *black box* and *glass box* testing has never been done. Thus, some number of bugs can be expected. The bugs can be computational, or in case of the GUI component display, related to variations in the Java Virtual Machine interpretation of the graphics code.

10.6 More Object-oriented Approach

Some of the design choices were made considering a rapidly approaching deadline, and in the future should be reconsidered in terms of a more object-oriented approach:

1. The *MagnitudePlot* and *PhasePlot* classes should both derive from the Plot class as opposed to having completely separate representations. This design would allow the reuse of some of the methods common to both plots, and present the connection between the two plot types better from the object-oriented perspective.

2. The plotting classes should be included as part of the *SystemModel* class and not separately, as part of the GUI which is the current model. Since the system plot can be thought of as a property of a system, these two characteristics should be among the system properties classes.

Chapter 11

Conclusion

Research is what I'm doing when I don't know what I'm doing.

Wernher Von Braun

11.1 Contributions

MASCoT is a fully functional compensating engine. It can run on any platform as a stand-alone application or as an applet in a web browser. While other compensating software tools are expensive, *MASCoT* is freeware which anyone can download and use. The toolkit is fully documented within the application, however for more detailed information, users can refer to this document. *MASCoT* has a fully functional graphical interface which makes the system very easy to use. Instead of getting bogged down with details, the user can immediately start evaluating his own compensators or run the compensating engine for hints.

Unlike other software compensation tools *MASCoT* is specifically designed to handle series compensation only. Thus the relative size of the program is small, making it more manageable. The toolkit combines all of the most important aspects of series compensation in one package without overwhelming the user with less important additions. It steps the user through a design process and solves the resulting compensation problem based on the user specifications as encountered in the real world.

11.2 Difficulties Encountered

With regards to the software language, every day I found out that Java does not always behave the way it is expected to behave. Different platforms create different problems, and the *Java Virtual Machine* differs in its interpretation of the code. When web-enabling my *MASCoT* toolkit, I spent many hours trying to make the applet run on a browser. Java Swing classes turned out to be a big inconvenience, since not all browsers currently have the proper plug-ins for Swing. This situation, however, can be rectified if the users download the latest Java plug-ins for their web browser.

With regards to the software structure, the algorithms for the compensation were the most difficult to implement. Since there are no clear algorithms to follow for compensation, I encountered problems when deciding how to pattern-recognize the input model in order to choose the right compensator when the compensation method was not specified. The proportional and dominant pole compensation was easy to implement since there are only a few degrees of freedom. However, as the number of potential solutions grows larger, I encountered difficulties in deciding how to properly trade off meeting different system requirements.

11.3 Extensions

This version of *MASCoT* is the first prototype of a compensating tool. Thus there are many potential improvements and extensions which can be implemented for this software. New and more complex compensating engines could be added to handle nonlinear systems or minor loop compensation. The existing compensating engines could be equipped with more effective algorithms, which improve the odds of finding a better solution. The graphical user interface can be implemented to operate faster and more reliably, to give the user more flexibility, and to be more aesthetically pleasing. More detailed description of suggested future directions is in chapter 10.

11.4 Summary

I hope that *MASCoT* will make learning and understanding compensation a less painful process for the future generations of control systems theory students. Although the toolkit can still undergo much improvement, it is a fully functional program which can serve both educational and practical needs. *MASCoT* is free to use and distribute. It can run on any platform supporting Java or in a web browser. The toolkit has a graphical interface, thus it is easy to learn and use. It can design a compensator for a linear system or it can evaluate and graph the compensator supplied by the user. I am happy and proud to develop something useful that can aid the learning process

since

Learning is what most adults will do for a living in the 21st century.

Perelman

Appendix A

Glossary

Actuating Signal The signal which is the difference between the *Reference Signal* and the *Feedback Signal*. It actuates the control unit in order to maintain the output at a desired value.

Bandwidth or 3dB Frequency (ω_h) When the magnitude falls to $\frac{1}{\sqrt{2}}$ (0.707) of its DC value.

Closed-loop Control System A system in which the output has an effect upon the input quantity in such a manner as to maintain the desired output value.

Compensation Refers to addition of other components to the system in order to alter its time and frequency characteristics.

Compensator or Controller A dynamic element that reacts to an *Actuating Signal* to produce a desired output. This unit does the main work for controlling the output. This element can be modified by the control systems designer. It is commonly implemented by a power amplifier.

Crossover (ω_c) The frequency when $|L(j\omega)| = 1$.

Damping ratio (ζ) Determines the characteristics of the input response of the system and the amount of oscillation or overshoot to be expected.

Deadzone Region where the output of the system remains zero for some range of input values.

Desensitivity ($1 + L(s)$) Denominator of all the closed loop functions, relates the closed-loop gain to the forward-path gain.

Feedback Element The unit that provides the means for feeding back the output quantity or some appropriately scaled function of the output called *Feedback Signal*, in order to compare it with the *Reference Signal*.

Feedback Signal The signal from the feedback loop which is a quantity proportional to the output variable. This variable is subsequently compared to the *Reference Signal* to determine the *Actuating Signal*.

Gain margin The factor of increase in the magnitude of the loop transmission which would cause instability.

Input Command or Reference Signal The input signal to the system, which is independent of the output of the system.

Loop gain Product of the magnitudes of all of the elements in a feedback loop.

Loop transmission or return ratio The product of all the elements along the loop when all the inputs and disturbance signals are set to zero.

Magnitude ($|\frac{C(j\omega)}{R(j\omega)}|$) The magnitude of the system.

Natural Frequency (ω_n) The undamped natural frequency of the system.

Open-loop Control System A system in which the output has no direct effect on the input signal.

Peak magnitude (M_p) The maximum magnitude of the frequency response; can be used to estimate phase margin or vice-versa, $M_p = \frac{1}{sin(p.m.)}$

Percentage overshoot The maximum output value for a step input as compared to its final value.

Phase angle (ϕ) A function of frequency; relates the phase of a system.

Phase crossover frequency (ω_ϕ) When $\phi = -180^o$.

Phase margin The amount of negative phase at crossover needed to make a system unstable.

Plant The hardware or process which is under control; usually that part of the loop which is fixed in advance and constrained.

Pole It is used to model some physical characteristic of a system when there is a negative change in the slope of a system's magnitude.

Regulator A system in which there is a constant steady-state value for a constant input signal; often encountered regulators are called speed and voltage regulators.

Rise time The time it takes for the output to go from 10% to 90% of its final value.

Servomechanism or servo A mechanical system, sometimes people refer to servo as a mechanical system in which the steady-state error is zero for a constant input signal.

Settling time The time it takes for the output to settle within a given percentage of its final value.

Singularities The poles and zeros in the system model.

Steady-state error A time-domain measure of how well a feedback system tracks its input.

System A combination of components that act together. The words *systems* could be interpreted to include physical, biological, organizational, and other entities or a combination thereof.

Time constant (τ) The time for the decaying exponential transient response to be reduced to e^{-1} of its initial value.

Zero It is used to model some physical characteristic of a system which causes some a positive change in the slope of a system's magnitude.

Appendix B

Contact Information

For digital copies of the thesis document or software please email me at:

patrycja@alum.mit.edu

or go to my web address at:

http://www.patrycja.com

Also, the thesis document and code should be available online from MIT Theses Online at:

http://thesis.mit.edu

Appendix C

Development Enviroment

C.1 Platform

MASCoT software was developed and compiled on Sun Microsystems Ultra 5/10 SPARC-II machines with 128 MB RAM and 333 MHz CPU. The operating system was SunOS 5.6 Distribution for Solaris 2.6.

C.2 Java Release

Java(TM) Development Kit, version 1.2.2. The Java Development Kit (JDK(TM)) is the development environment for building applications, applets, and components that can be deployed on the Java platform.

C.2.1 Sun Java Web Pages

For additional information, refer to these Sun Microsystems pages on the World Wide Web:

http://java.sun.com/ The Java Software web site, with the latest information on Java technology, product information, news, and features.

http://java.sun.com/products/jdk/1.2/index.html JDK 1.2 Product and Download Page

http://java.sun.com/docs Java Platform Documentation provides access to white papers, the Java Tutorial and other documents.

http://developer.java.sun.com The Java Developer Connection web site. (Free registration required.) Additional technical information, news, and features; user forums; support information, and much more.

http://java.sun.com/products Java Technology Products & API

http://www.sun.com/solaris/java/ Java Development Kit for Solaris - Production Release

Bibliography

[1] George A. Biernson. Quick method for evaluating the closed-loop poles of feedback-control systems. Technical Report 38, Massachusetts Institute of Technology, Servomechanisms Laboratory, Cambridge, Massachusetts, 1952.

[2] George A. Biernson. Fundamental equations for the application of statistical techniques to feedback control systems. Technical Report 7138, Massachusetts Institute of Technology, Servomechanisms Laboratory, Department of Electrical Engineering, Cambridge, Massachusetts, 1955.

[3] Robert A. Bruns and Robert M. Saunders. *Analysis of Feedback Control Systems; Servomechanisms and Automatic Regulators.* McGraw-Hill Electrical and Electronic Engineering Series. McGraw-Hill, New York, New York, second edition, 1955.

[4] Frank M. Callier and Charles A. Desoer. *Multivariable Feedback Systems.* Springer Texts in Electrical Engineering. Springer-Verlag, New York, New York, first edition, 1982.

[5] Sheldon S. L. Chang. *Synthesis of Optimum Control Systems.* McGraw-Hill Series in Control Systems Engineering. McGraw-Hill, New York, New York, first edition, 1961.

[6] Kenneth Paul Davis. Integer-state feedback compensators as an approach to finite-state feedback compensators. Master's thesis, Massachusetts Institute of Technology, 1977.

[7] John J. D'Azzo and Constantine H. Houpis. *Feedback Control System Analysis and Synthesis.* McGraw-Hill Book Company, USA, second edition, 1966. Very thorough descriptions of different methods of analysis for feedback systems such as Bode, Root Locus, Nyquist, and more. Quantitative approach to compensation design covering both series and parallel compensation.

[8] H. M. Deitel and P. J. Deitel. *Java How to Program.* Prentice Hall, Upper Saddle River, New Jersey, third edition, 1999. Very thorough coverage of everything you need to know to program and design for Java. Covers everything from the basics of object oriented design to data structures and algorithms; filled with examples.

[9] David Flanagan. *Java in a Nutshell.* O'Reilly, Cambridge, Massachusetts, third edition, 1999. A quick reference book for Java. Accelerated version to the Java language with its key APIs with all the Java development tools documented.

[10] Jay Frank. *IEEE Standard Dictionary of Electrical and Electronics Terms.* Institute of Electrical and Electronics Engineers, New York, New York, fourth edition, 1988.

[11] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems.* Addison-Wesley Series in Electrical Engineering. Addison-Wesley, Reading, Massachusetts, first edition, 1986.

[12] L. A. Gould, W. R. Markey, J. K. Roberge, and D. L. Trumper. *Control Systems Theory.* Massachusetts Institute of Technology, Cambridge, MA, second edition, 1997. A collection of chapters written by MIT professors who share their practical experience with feedback system design. Covers everything from basics on block diagram models to analysis methods for complex feedback systems through compensation design techniques.

[13] Vinton B. Haas. Evaluation of feedback control systems. Technical report, Massachusetts Institute of Technology, Servomechanisms Laboratory, Energy Conversion Group, Cambridge, Massachusetts, May 1956.

[14] Isaac M. Horowitz. *Synthesis of Feedback Systems.* Academic Press, New York, New York, first edition, 1966. Covers some feedback systems theory, however the focus is on feedback control.

[15] Zoher Z. Karu. *Signals and Systems Made Ridiculously Simple.* ZiZi Press, Cambridge, Massachusetts, first edition, 1995. Great handbook of useful information on Laplace theorems, signal characteristics, feedback, Bode plots, filters and much more.

[16] Helmut Kopka and Patrick W. Daly. *A Guide to Latex.* Addison-Wesley, Reading, Massachusetts, third edition, 1999. Covers latex fundamentals and more advanced techniques for document preparation in Latex. Used it as a Latex resource.

[17] Benjamin C. Kuo. *Automatic Control Systems.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, fifth edition, 1987. Control theory book covering everything from the fundamentals of system modeling and analysis to feedback design techniques. Examples and exercises are included in each chapter. The book assumes strong mathematics background. The examples include computer models of different systems and digital design.

[18] Robert Lafore. *Data Structures & Algorithms in Java.* Mitchell Waite Signature Series. Waite Group Press, Corte Madera, California, first edition, 1998. Book filled with examples on data structures and algorithms as used in computer programming. Examples are in Java but they can be understood by any sofware developer.

[19] Gladwyn Vaile Lago and Lloyd M. Benningfield. *Control System Theory: Feedback Engineering.* Ronald Press Co., New York, New York, first edition, 1962.

[20] Richard Harris Mahn. Algorithm for and implementation of pole assignment using output feedback in linear multivariable time-invariant systems. Master's thesis, Massachusetts Institute of Technology, 1977.

[21] Theo Mandel. *The Elements of User Interface Design.* Wiley Computer Publishing, New York, NY, first edition, 1997. Covers foundations of user interface design, graphical and object-oriented interfaces, user interface design process, internet interfaces, agents and social interfaces.

[22] Otto Mayr. *The Origins of Feedback Control.* MIT Press Publication. MIT Press, Cambridge, Massachusetts, first edition, 1970.

[23] Emanuel Pericles and Edward Leff. *Introduction to Feedback Control Systems.* McGraw-Hill Series in Electrical Engineering. McGraw-Hill, New York, New York, first edition, 1979. Covers feedback systems' analysis and synthesis on the introductory level.

[24] Vartan Piroumian. *Java GUI Development.* SAMS, A division of Macmillan Computer Publishing, Indianapolis, Indiana, first edition, 8August 1999. Concepts of GUI design, MVC architecture, AWT and Swing components, layout managers, event handling, colors and fonts, graphics, Swing model architecture.

[25] James K. Roberge. *Operational Amplifiers, Theory and Practice.* John Wiley & Sons, Inc., USA, first edition, 1975. Great source of inside knowledge on the topic of operational amplifiers. It is a great teaching source for a feedback system control design.

[26] James R. Rowland. *Linear Control Systems: Modeling, Analysis, and Design.* Wiley, New York, New York, first edition, 1986.

[27] Andrew P. Sage. *Linear Systems Control.* Matrix Series in Circuits and Systems. Matrix Publishers, Champaign, Illinois, first edition, 1978. Covers fundamentals of linear systems control analysis. Goes into the synthesis stages and gives examples.

[28] C. J. Savant. *Control System Design.* McGraw-Hill, New York, New York, second edition, 1964.

[29] Stanley M. Shinners. *Control System Design.* Wiley, New York, New York, first edition, 1964.

[30] William M. Siebert. *Circuits, Signals, and Systems.* The MIT Electrical Engineering and Computer Science Series. The MIT Press, Cambridge, Massachusetts, first edition, 1995. Provides all the fundamentals of the circuit theory and linear systems theory followed by feedback systems analysis and Laplace transforms.

[31] Otto Joseph Mitchell Smith. *Feedback Control Systems.* McGraw-Hill Series in Control Systems Engineering. McGraw-Hill, New York, New York, first edition, 1958.

[32] George Julius Thaler. *Design of Feedback Systems.* Dowden, Hutchinson & Ross, Stroudsburg, Pennsylvania, first edition, 1973.

[33] John G. Truxal. *Automatic Feedback Control Synthesis.* McGraw-Hill Electrical and Electronic Engineering Series. McGraw-Hill, New York, New York, first edition, 1955.

[34] Jan C. Willems. *The Analysis of Feedback Systems.* MIT Press Publications. The MIT Press, Cambridge, Massachusetts, first edition, 1971.