Interactive Frequency Response Analysis of Linear Systems Using Poles and Zeros

In partial fulfillment of the requirements for

EE 4012 Senior Design

By:

Jonathan Hamaker
hamaker@isip.msstate.edu

Instructor:

Dr. Bert Nail

Department of Electrical & Computer Engineering
Mississippi State University
Mississippi State, MS 39762
Spring Semester 1997

Table of Contents

Abstract 1

1. Introduction 1

2. Background 2

3. Critical Design Issues 4

    3.1 Availability and Portability 4

    3.2 Interface Considerations 5

    3.3 Future Compatibility 5

4. Design and Implementation 6

    4.1 Determination of Functionality 6

    4.2 User Interface 7

    4.3 Software Design 9

5. Conclusions 11

6. Future Work 11

7. Acknowledgements 12

8. References 12

Appendix A.1 Source Code Overview 13

Appendix A.2 Source Code Details 14

Interactive Frequency Response Analysis of Linear Systems Using Poles and Zeros

Jonathan Hamaker

EE 4012 -- Senior Design Project
Department of Electrical and Computer Engineering
Mississippi State University, Mississippi 39762
hamaker@isip.msstate.edu

ABSTRACT

Pole/Zero response of linear systems is one of the fundamental topics covered in any electrical engineering curriculum. Yet, so many students leave these courses without a full understanding of this concept. Visualization of pole/zero response of systems is not intuitive (despite what the textbooks may say). Thus, the student needs tools which allow him to visualize the pole/zero interaction and correlate this visualization to the discussion in the textbook. To this end, we have developed a graphical user interface (GUI) which allows the student to explore the basic pole/zero analysis concepts for both analog and digital systems. This GUI is based on the platform-independent Java programming language and follows a hierarchical design. These factors provide both compatibility to future Java enhancements and accessibility to students (the target audience). This GUI is one of many tools which will aid the undergraduate and novice engineer in understanding of fundamental signals and systems concepts.

1. INTRODUCTION

Poles and zeros are fundamental to any linear system. They are the components which produce the frequency responses that electrical engineers have been taught to know as magnitude and phase response of the system. Commonly, these responses are shown in a two-dimensional plot of

magnitude vs. frequency or phase vs. frequency. However, the pole/zero response is not a 2-D response but rather a 3-D surface produced by the interaction of the poles and zeros. A sample of this surface is shown in Figure 1.

2. BACKGROUND

Conventional textbook analysis is based on pole and zero placement in either the s-plane or the z-plane. The position of a pole or zero in each of these planes is fundamentally related to the frequency and bandwidth of the response produced by the pole or zero. The basic concept is shown in Figure 2.

For analog systems, the center frequency of the response produced is related to the distance from the real axis. This relation is shown in Equation 1. The bandwidth of the response produced is similarly a function of the distance from the imaginary axis as shown in Equation 2.

$$f = \frac{\text{distance}}{2\pi} \quad (1)$$

$$BW = \frac{-(\text{distance})}{2} \quad (2)$$

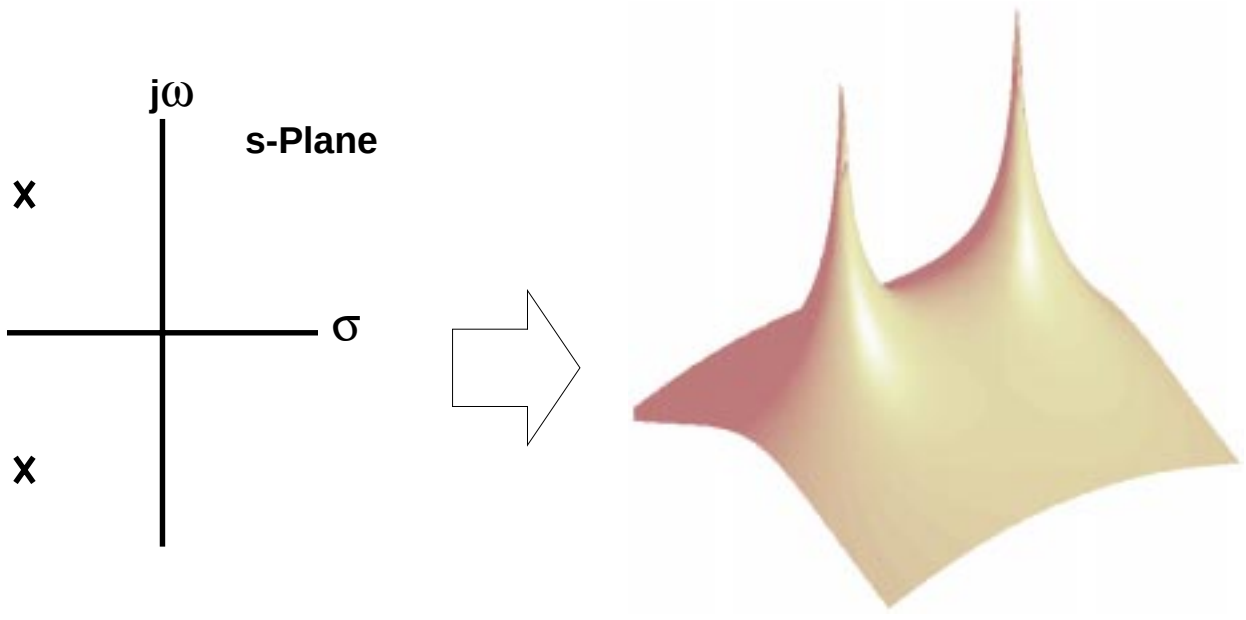

Figure 1. 3 dimensional frequency response to a two-pole system. Note that the conventional 2 dimensional response is a slice of this 3-D surface.

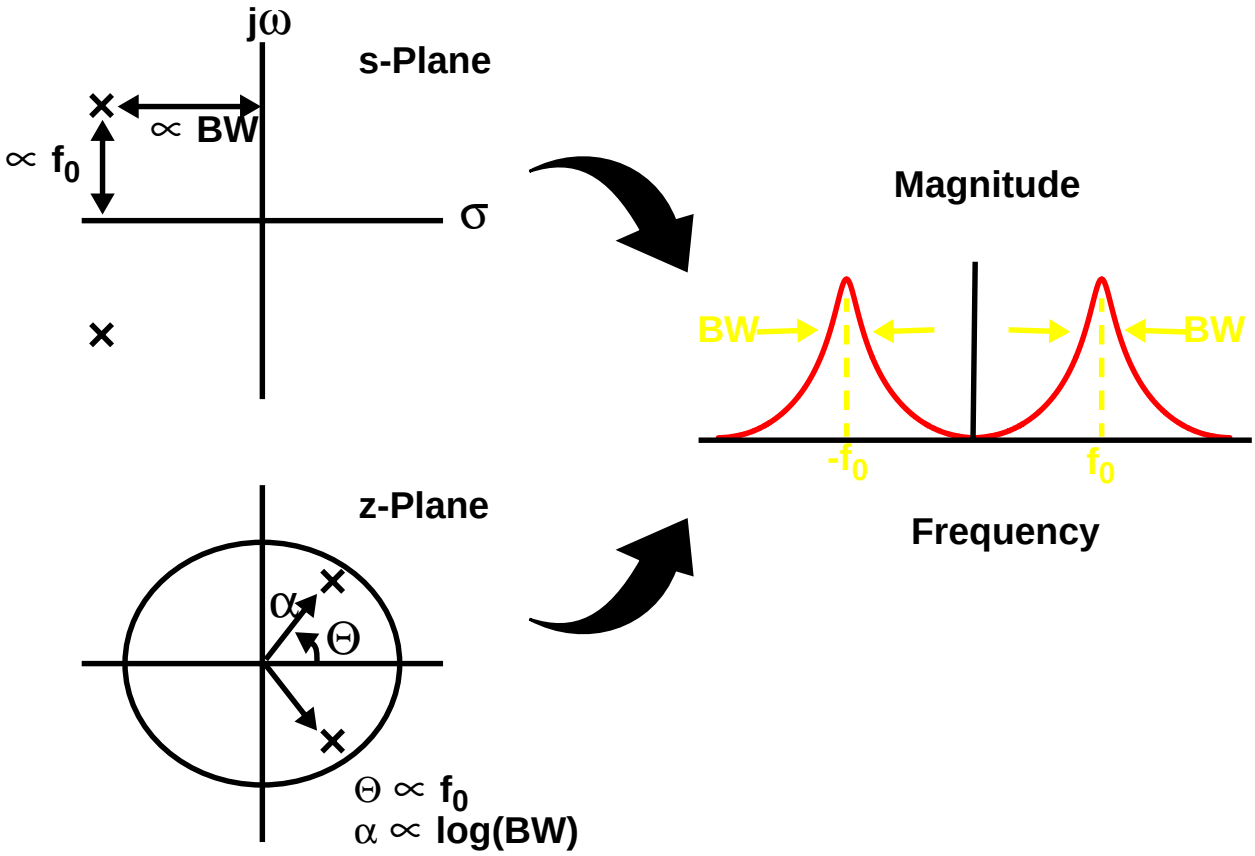

Figure 2. Relationship between frequency, bandwidth and location in the s-plane and z-plane. Note that in the z-plane the relationship is not linear.

The response due to digital poles and zeros is quite different from that caused by analog poles and zeros. The z-plane is a non-linear mapping of a strip of the s-plane into a unit circle. The conversion is such that the left half of the s-plane falls inside the unit circle, the right half outside the unit circle and the imaginary axis onto the unit circle. Due to sampling theorem, this strip is repeated at equal intervals and, thus, each point in the unit circle represents an infinite number of frequency points. As with the s-plane, the position of the poles and zeros in the z-plane is related to the frequency and bandwidth of the response produced. These relations are shown in Equations 3 and 4 respectively.

$$f = \frac{\Theta}{2\pi} * (\text{sample frequency}) \qquad (3)$$

$$BW = -\frac{(\log(\alpha))*(\text{sample frequency})}{2\pi} \quad (2)$$

3. Critical Design Issues

The system response tool was created to facilitate learning by students and, as such, requires certain attributes. These include: effortless availability and portability to a wide range of students, an easy to use as well as comprehensive interface, and (of course) accurate results. In addition, this project is not intended to be extinct in a few years. To eliminate this possibility, care was taken to insure compatibility with future versions of Java.

3.1. Availability and Portability

With students as the target audience, availability and portability are especially important. We can not simply create a piece of software which requires an installation process so difficult that only a computer expert could compile it. Nor could we produce software which is available on a single platform or for a single machine. Luckily, there is a programming language which takes care of both of these problems - Java.

Java is a programming language which provides platform-independent coding. This means that code compiled on one machine will run on another “Java-compatible” machine without recompiling. This is a huge advantage for students since this allows them to use it with no knowledge of compilers. In this respect, Java is a suitable choice over languages such as Visual C++ or Visual Basic.

Java also provides the portability. To date, Java is freely available on the most popular platforms (Windows, Unix, SGI, etc.). Java is also integrated into the two most popular web browsers - Netscape and Microsoft Internet Explorer. This is an extremely useful characteristic of Java because it allows our tool to be used by nearly any student who has access to a computer.

There is a down-side to using Java as the: the low-level graphics are platform-dependent. Thus, a widget which is rendered the way one would like on a certain system, may not be rendered exactly the same on a different system. This leads to difficult design, because in the attempt to make the software as generic as possible, the designer must constantly adjust the GUI to look good on every possible user's system. However, despite this problem, Java is a good choice because it allows us to reach most every member of our target audience.

3.2. Interface Considerations

To be used as a learning tool, the interface must not interfere with the learning experience. The user must be given as much control as possible over the tool without being overwhelmed with options. The input and output should be integrated such that the user can see the result of input actions. On-line help should also be provided for all aspects of the interface.

Another important consideration for the interface is the accuracy and clarity of the results displayed. Extensive testing must be undertaken to insure that the user gets the expected results for all cases. Also the user should be given control over the output displays. The user may want to see a part of the output that is not visible and, thus, should have full control over the look of the output

3.3. Future Compatibility

Compatibility is always an issue in software design. The software would be virtually useless if it did not operate in 2 years or if it could not be upgraded easily. Using an object-oriented approach is meant to eliminate this concern. By defining a hierarchy which is not based on low-level Java code, we remove ourselves from the conflicts caused by future Java upgrades. Also, we must use only

"standard" classes for the interface. If we were to use non-standard classes there is the chance that future upgrades would not be compatible with our code.

Despite the hoopla surrounding object-oriented programming, there are some problems. The first downfall is that the designer might have to give up desired functionality in order to use the standard classes. The designer can not merely change the standard class because upon upgrading he would have to change the new version also.

The more tasking problem is that Java is still a relatively new language which derives from an unfinished standard. Java is a relatively new language which has been through only 2 major revisions. Thus it is subject to drastic changes which might greatly effect the future operation of the code running under it. This is another reason that a hierarchy far removed from the low-level code is extremely important. The low-level code is much more likely to be changed than the high level code, thus we shrink our chance of losing compatibility during an upgrade.

4. DESIGN AND IMPLEMENTATION

This design was a systematic one in which the foundations were thought through at length well before a single piece of code was written. The process involved consisted of four major steps: determination of necessary functionality; design of user interface; design of the software; and integration of all parts into a final product and testing it.

4.1. Determination of Functionality

A clear vision of the final product was necessary before beginning the software construction. There were two major concerns in the design of this system. The first of these was that the user understand the planar locations of poles and zeros. To

allow for this we design the system such that the input area is relatively intuitive and so the user can manipulate the pole/zero position once on the map. Secondly, the output needed to include the frequency response as well as the time-domain response of the system. This was the fundamental component that the rest of the design revolved around.

4.2. User Interface

The user interface was designed with the concept of separating input and output. Figure 3 shows the final interface design for the system response tool. One can see that the top half of the screen contains all of the input components including an area to enter the pole/zero frequency and bandwidth. Frequency and bandwidth were chosen as the method for input instead of the location in the planes because this was the concept we were trying to impart to the student - the connection between poles, zeros and the frequency and bandwidth of the response they produce.

The pole/zero map provides an area for the student to manipulate the poles and zeros to note how the response changes as the poles/zeros are moved in the plane. This is important because it helps the student gain an intuition of the interaction between the poles and zeros based on their relative location in the planes.

The output panel consists of the magnitude and phase responses. (The impulse response is not included at this point but will be added in future versions.) These panels show the frequency response produced by the poles and zeros in the pole/zero map. These plots can be interactively zoomed in on and panned. They also give the user the ability to trace the plot. This, in addition to the pole/zero map, are the most important areas in the GUI because they allow the user to make

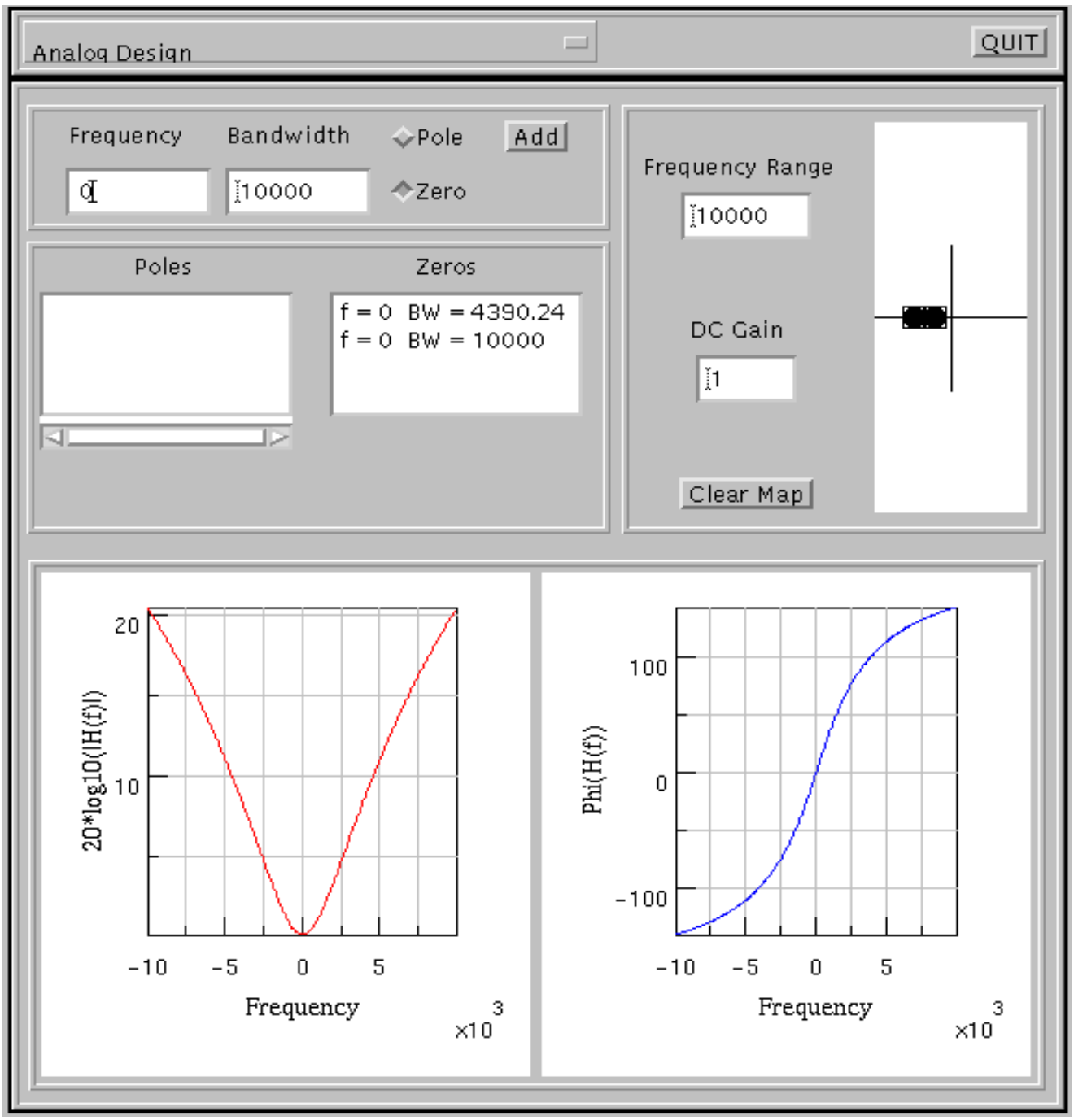

Figure 3. System response GUI. Note the clear division between the input and output areas.

the connection between frequency response and the locations of the poles and zeros.

4.3. Software Design (Appendix A)

The software was designed according to the hierarchical object-oriented pattern shown in Figure 4. One can see that the input and output are the major divisions in the software. This tool is also event-driven such that the output only responds to actions initiated at the input. Thus, the input effectively controls the output. Note also that their is only one connection to the Java classes. This helps to insure compatibility as discussed earlier since there need only be concern over the

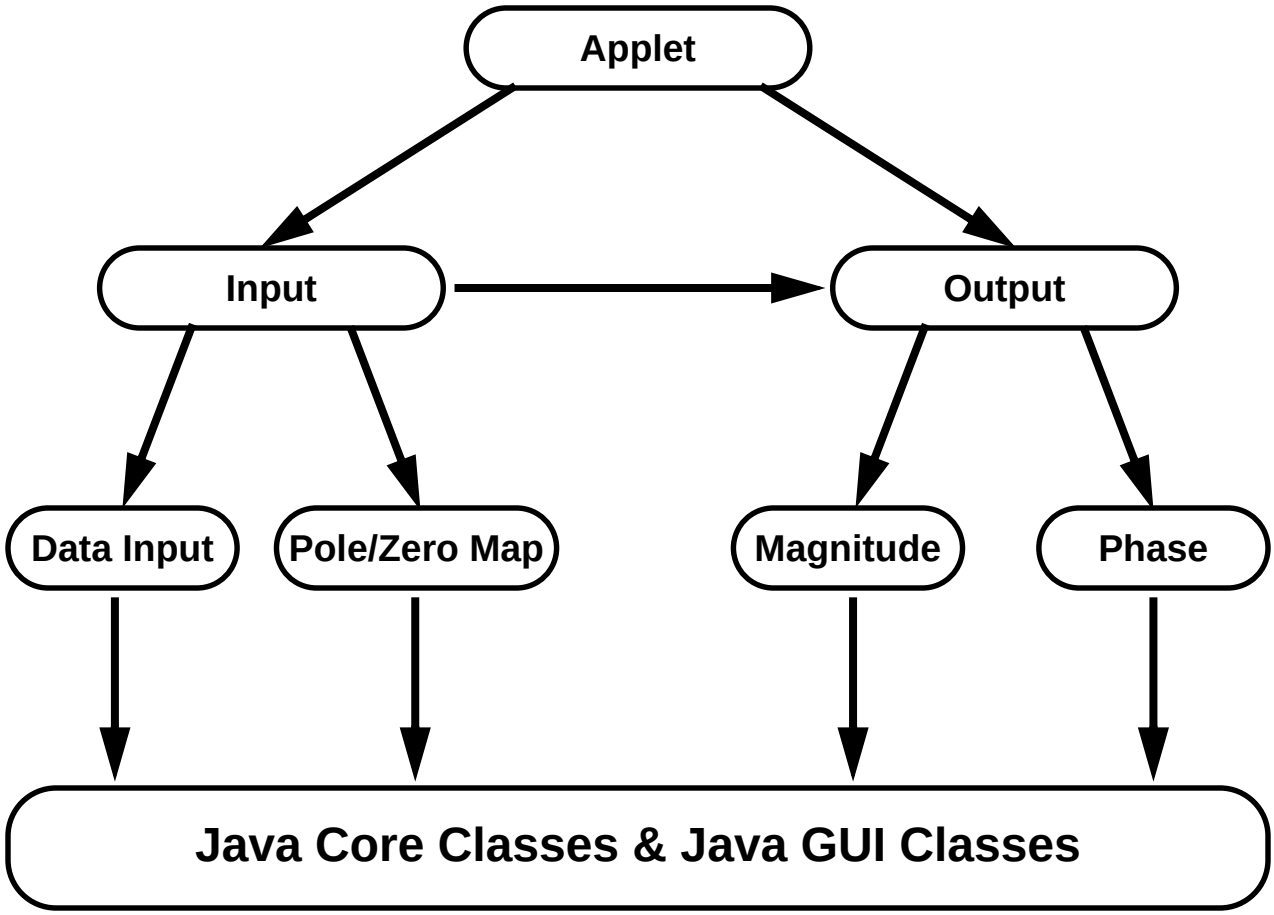

Figure 4. Overview of hierarchical software design. Note the single dependence on Java classes

change of the single connection.

The design of the software also included consideration of the interface. The interface was not intended to be slow to respond but, upon initial testing, this was the case. The GUI was required to respond to events in a sequential fashion. To circumvent this problem, we chose to use the thread capabilities inherent in Java. Provided in Java is a thread class which allows our classes to operate in parallel. For instance, the magnitude and phase responses can be calculated simultaneously instead of sequentially. This allows a smoother interface and allows the user to continue tweaking the pole/zero system without waiting after each change. Each of the classes which required time consuming calculations which could be done independently of another class were instantiated with these threads.

Another concern in writing this software was the need for future upgrades. To facilitate this need, we have provided Appendix A which details the hierarchy of our code and the class structures and methods. This should prove to be a useful tool for future improvement of this tool.

4.4. The Final Product and Testing

With all of these considerations accounted for, the final pieces were put into place. The most important of these was testing of the product. Testing was performed by comparing the output of the tool to a large number of pole/zero systems in textbooks. The total systems tested were 12 analog systems and 8 digital systems. Each of the system responses from our tool matched perfectly with the responses listed in the textbook for both magnitude and phase.

Being reasonably assured that the output was correct, the testing moved to the next phase: testing for accessibility. The GUI

was run on a variety of platforms including Windows95, SunOS, and Solaris. On each of these systems we tested using Netscape Navigator 3.0 and the Java Appletviewer. We also tested using a number of different windowing systems. The results of these tests were discouraging. Although, the software worked fine in each of the different cases, the appearance of the GUI was different. In some cases, the GUI was not even useful because certain parts of it were obscured by others. This is indicative of the problem discussed earlier - the low-level graphics are platform-dependent.

There is no intuitive way to combat this problem unless and until Java develops platform-independent widgets or until the various systems set some kind of standard for the look of their widgets.

5. CONCLUSIONS

In this paper, we have described a Java-based system for pole/zero analysis of linear systems. This tool incorporates the fundamental concepts of system response into a visualization tool that students can use for enhancement of the textbook understanding. All software (source and executable) can be found on our web site at

http://isip.msstate.edu/software/java_system_response/

6. FUTURE WORK

Although this project accomplished the majority of the desired goals, it is still lacking in a couple of areas. The most important of these are the need for on-line help and time-domain response. Both of these features will be added to the next release. Also, the interface needs to be expanded to allow the user even greater flexibility and control over the design process.

7. ACKNOWLEDGEMENTS

The author wishes to thank a number of people for their support in this effort. First, I wish to thank Janna Shaffer for her artistic direction in designing the GUI. Secondly, Dr. Joe Picone and Aravind Ganapathiraju and all of the members of ISIP for their technical direction in times of need. Lastly, I would like to thank all of those who tolerated my absence from their lives over the past 4 months.

8. REFERENCES

1 Blinchikoff, Herman J., and Anatol I. Zverev, Filtering in the Time and Frequency Domains. New York: John Wiley and Sons, 1976.

2 Williams, Charles S., Designing Digital Filters. Englewood Cliffs, New Jersey: Prentice Hall, 1986.

3 Bogner, R.E. and A.G. Constantinides. Introduction to Digital Filtering. New York: John Wiley and Sons, 1975.

4 Van Valkenburg, M.E., Analog Filter Design, Fort Worth, Texas: Holt, Rinehart and Winston, Inc., 1982

APPENDIX A.1: SOURCE CODE OVERVIEW

FilterApplet -> java.lang.Applet

TogglePanel -> SubPanel -> FramedPanel -> java.awt.Panel

Designer -> SubPanel -> FramedPanel -> java.awt.Panel

InputPanel -> SubPanel -> FramedPanel -> java.awt.Panel

DataInputPanel -> SubPanel -> FramedPanel -> java.awt.Panel

DataDisplayPanel -> SubPanel -> FramedPanel -> java.awt.Panel

PzmapPanel -> SubPanel -> FramedPanel -> java.awt.Panel

Pzmap -> java.awt.Canvas

OutputPanel -> SubPanel -> FramedPanel -> java.awt.Panel

MagResponse -> G2Dint -> Graph2D -> java.awt.Canvas

PhaseResponse -> G2Dint -> Graph2D -> java.awt.Canvas

ImpulseResponse -> G2Dint -> Graph2D -> java.awt.Canvas

APPENDIX A.2: SOURCE CODE DETAILS

file: Constants.java

class Constants

contains all of the constants for the applet classes

Hierarchy: PoleZero

file: PoleZero.java

class PoleZero

Implements an object representing a pole or zero

Hierarchy: PoleZero

public double mag_response(double frequency_in, double sample_frequency)

returns the magnitude response of this digital pole or zero object at the indicated frequency

public double mag_response(double frequency_in)

returns the magnitude response of the analog pole or zero object at the indicated frequency

public double phase_response(double frequency_in, double sample_frequency)

returns the phase response of this digital pole or zero object at the indicated frequency

public double phase_response(double frequency_in)

returns the phase response of the analog pole or zero object at the indicated frequency

public boolean debug()

prints the indicated debugging statement

file: ZPlane.java

public class ZPlane
this class contains the necessary methods to convert from z-plane coordinates to frequency coordinates

Hierarchy: ZPlane

public static double get_freq (double z_real, double z_imag, double sf)
this method returns the frequency of the point in the z-plane given the sample-frequency

public static double get_band(double z_real, double z_imag, double sf)
this method returns the bandwidth of the point in the z-plane given the sample-frequency

public static double get_real(double frequency, double bandwidth,double sf)
this method returns the real coordinate in the z-plane given the frequency, bandwidth and the sample-frequency

public static double get_imag(double frequency, double bandwidth,double sf)
this method returns the imaginary coordinate in the z-plane given the frequency, bandwidth, and the sample-frequency

file: PzGraphicsObj.java

class PzGraphicsObj
Implements an object which holds the graphics and pixel bounds for a pole or zero in the pole/zero map

Hierarchy: PzGraphicsObj

public boolean set_limits(int xmin_a, int ymin_a, int xmax_a, int ymax_a)
sets the new limits of the object

public boolean move_by_one (int direction)
moves this object in any direction by one pixel

public int getx ()
returns the x coordinate of the object

public int gety ()
returns the y coordinate of the object

public boolean does_contain (int x, int y)
determines if the x and y coordinate are located within the graphics bounds

public boolean draw_self (Graphics graph)
this is a method to draw this pole or zero on the map when called

file: FramedPanel.java

class FramedPanel extends Panel
Draws 3-D frame around a panel.

Hierarchy: FramedPanel -> java.awt.Panel

public Insets insets()
Ensures that no Component is placed on top of the frame. Overrides the Insets method of the Panel class.

public void paint(Graphics g)
Draws the frame at this panel's edges.

file: SubPanel.java

class SubPanelpu extends FramedPanel
SubPanel is a FramedPanel with the layout manager of GridBagLayout.

Hierarchy: SubPanel -> FramedPanel -> java.awt.Panel

public boolean constrain(Container container, Component component,
constrain creates the constraints of a component in a container and then adds the component to the container. it is based on code from "Java in a Nutshell" by David Flanagan. Published by O'Reilly and Associates incorporated.

file: FilterApplet.java

class FilterApplet extends Applet implements Runnable
this class implements the interface to a threaded applet which allows the user to interactively design analog and digital filters

Hierarchy: FilterApplet -> java.applet.Applet

public void init()
this method initializes this applet

public void run()
this method runs the threaded applet

file: TogglePanel.java

public class TogglePanel extends SubPanel implements Runnable
this class implements the interface to allow toggling of the analog and digital designs of a Designer panel

Hierarchy: TogglePanel -> SubPanel -> FramedPanel -> java.awt.Panel

public TogglePanel(Designer designer_a, int type)
this method initializes the class

void place_components ()
this method places the individual controls on the panel

public void run()
this method runs the threaded class

public boolean handleEvent(Event event)
this method handles the various interrupt events that may occur in this component

file: Designer.java

public class Designer extends SubPanel

this class implements a tool which allows the user to place poles and zeros on a pole-zero map interactively and view the effects of these poles and zeros on the frequency response and the impulse response this class implements the interface to allow toggling of the analog and digital designs of a Designer panel

Hierarchy: Designer -> SubPanel -> FramedPanel -> java.awt.Panel

public Designer(int type)

this method constructs the default Designer object

private boolean add_components()

this method initializes the gui components of the class

public boolean set_design_type(int new_type)

this is a method to set the design type. the design type will be set to the value passed in to new_type

file: InputPanel.java

class InputPanel extends SubPanel

this class implements a tool which controls the input to a pole-zero map it allows the user to place poles and zeros.

Hierarchy: InputPanel-> SubPanel -> FramedPanel -> java.awt.Panel

public InputPanel(OutputPanel output_a, int type)

this method constructs the default InputPanel object

private boolean add_components()

this method initializes the gui components of the class

public boolean set_design_type(int new_type)

this is a method to set the design type. the design type will be set to the value passed in to new_type

file: DataInputPanel.java

public class DataInputPanel extends SubPanel implements Runnable
this class implements a canvas which allows one to enter a pole or a zero and causes it to be displayed on a pole-zero map

Hierarchy: DataInputPanel-> SubPanel -> FramedPanel -> java.awt.Panel

public DataInputPanel(PzmapPanel pz_map_a, int type)
this method constructs a DataInputPanel of the default type

private boolean add_components()
this method initializes the gui components of the class

public boolean set_design_type(int new_type)
this is a method to set the design type. the design type will be set to the value passed in to new_type

public void run()
this method runs the threaded applet

public boolean add_pz()
this is a method to add a pole or zero to the appropriate portions of our frame.

public boolean handleEvent(Event event)
this method handles the various interrupt events that may occur in this component

file: DataDisplayPanel.java

public class DataDisplayPanel extends SubPanel
this class implements a canvas which holds a list of poles and zeros.

Hierarchy: DataDisplayPanel -> SubPanel -> FramedPanel -> java.awt.Panel

public DataDisplayPanel(int type)
this method constructs a DataDisplayPanel of the default type

private boolean add_components()
this method initializes the gui components of the class

public boolean clear_all()
this is a method to clear all poles and zeros from the lists

public boolean add_zero(double frequency, double bandwidth)
this is a method to add a zero to the zero list

public boolean add_pole(double frequency, double bandwidth)
this is a method to add a pole to the pole list

public boolean set_design_type(int new_type)
this is a method to set the design type. the design type will be set to the value passed in to new_type

public boolean reset_selected()
this is a method resets the lists so that both are deselected

public boolean remove_selected()
this is a method removes the current pole/zero from its list

public boolean select_item(double frequency, double bandwidth, int type)
this is a method to set the selected item in one of the lists

public boolean change_value(double new_freq, double new_band)
this is a method to change a pole or zero in the respective list

file: PzmapPanel.java

public class PzmapPanel extends SubPanel implements Runnable
this class implements a canvas which holds either a digital or analog pole-zero map and displays the location of poles and zeros on that map.

Hierarchy: PzmapPanel -> SubPanel -> FramedPanel -> java.awt.Panel

public PzmapPanel(OutputPanel output_a, DataDisplayPanel display_a, int type)
this method constructs a PzmapPanel of the default type

private boolean add_components()
this method initializes the gui components of the class

public boolean clear_all()
this is a method to clear all poles and zeros from the map

public boolean add_zero(double frequency, double bandwidth)
this is a method to add a zero to the pzmap

public boolean add_pole(double frequency, double bandwidth)
this is a method to add a pole to the pzmap

public void run()
this method runs the threaded class

public boolean set_sample_frequency (double new_freq)
this is a method to set the sample frequency. This has an effect only on the digital map

public boolean set_dc_gain (double new_dc_gain)
this is a method to set the dc gain of the filter

public boolean set_map_type(int new_type)
this is a method to set the map type. the map type will be set to the value passed in to new_type

public boolean handleEvent(Event event)
this method handles the various interrupt events that may occur in this component

file: Pzmap.java

public class Pzmap extends Canvas implements Runnable
this class implements a canvas which holds either a digital or analog pole-zero map and displays the location of poles and zeros on that map.

Hierarchy: Pzmap -> java.awt.Canvas

public Pzmap(int type, PoleZero pz_array_a[], OutputPanel output_a, DataDisplayPanel data_display_a)
this method constructs a Pzmap of the default type

private boolean draw_analog_map(Graphics graph)
this method will draw an s-plane pole/zero map on the input graphic object

private boolean draw_digital_map(Graphics graph)
this method will draw a unit circle pole/zero map on the input graphic object

private boolean add_analog_pole (double frequency, double bandwidth)
this is a method to add a pole to the current analog map

private boolean add_analog_zero (double frequency, double bandwidth)
tthis is a method to add a zero to the current analog map

private boolean add_digital_pole (double frequency, double bandwidth)
this is a method to add a pole to the current digital map

private boolean add_digital_zero (double frequency, double bandwidth)
tthis is a method to add a zero to the current digital map

public boolean clear_all()
this is a method to clear all poles and zeros from the map

public boolean add_zero(double frequency, double bandwidth)
this is a method to add a zero to the pzmap

public boolean add_pole(double frequency, double bandwidth)
this is a method to add a pole to the pzmap

public void run()
this method runs the threaded class

public boolean set_sample_frequency (double new_freq)
this is a method to set the sample frequency. This has an effect only on the digital map

public boolean set_map_type(int new_type)
this is a method to set the map type. the map type will be set to the value passed in to new_type

public boolean keyDown(Event e, int key)
this method determines the key pressed if the key is an arrow key then it allows micro-positioning

public boolean keyUp(Event e, int key)
this method determines the key released if the key is an arrow key then it refreshes the output screenbe set to the value

public boolean mouseDown(Event e, int x, int y)
this method determines the location of a mouse click and figures out whether or not it was intended to be on a pole or zero. It sets the current pole or zero to be the one clicked on.

public boolean mouseUp(Event e, int x, int y)
this method determines the location of a mouse click it sets the current pole or zero to be unknown

public boolean mouseDrag(Event e, int x, int y)
this method determines the location of a mouse drag and sets the current pole or zero to point to this new location

file: OutputPanel.java

class OutputPanel extends SubPanel

this class implements a tool which controls the output of a pole-zero map it displays the various responses

Hierarchy: OutputPanel -> SubPanel -> FramedPanel -> java.awt.Panel

public OutputPanel(int type, double frequency, PoleZero pz_array_a[])putPanel()

this method constructs the default OutputPanel object

private boolean add_components()

this method initializes the gui components of the class

public boolean update_screen()

this method sets the update_flag causing a repaint of the canvas

public boolean partial_update()

this method only updates a portion of this panel

public boolean set_pz_array(PoleZero pz_array_a[])

this method sets the pz_array to point to the input variable

public boolean set_frequency(double frequency)

this method sets the sample frequency or maximum frequency

public boolean set_dc_gain(double new_dc_gain)

this method sets the dc gain of the filter

public boolean set_design_type(int new_type)

this is a method to set the design type. the design type will be set to the value passed in to new_type

file: MagResponse.java

public class MagResponse extends G2Dint implements Runnable

this class implements a canvas which holds the magnitude response of a filter described by a set of poles and zeros.

Hierarchy: MagResponse -> G2Dint -> Graph2D -> java.awt.Canvas

public MagResponse(double frequency, PoleZero pz_array_a[], int type)

this method constructs a MagResponse canvas

public boolean draw_analog_response(int length)

this method draws the frequency response of the analog poles and zeros on the screen

public boolean draw_digital_response(int length)

this method draws the frequency response of the digital poles and zeros on the screen

public boolean set_design_type(int new_type)

this is a method to set the design type. the design type will be set to the value passed in to new_type

public boolean set_pz_array(PoleZero pz_array_a[])

this method sets the pz_array to point to the input variable

public boolean set_dc_gain(double new_dc_gain)

this method sets the dc gain of the filter

public boolean set_frequency(double frequency)

this method sets the sample frequency or maximum frequency

public boolean update_screen()

update_screen sets the update_flag causing a repaint of the canvas.

public void run()

run is the body of the thread. it will update the display area when necessary. The necessity is based on the condition of the update_flag.

file: PhaseResponse.java

public class PhaseResponse extends G2Dint implements Runnable

this class implements a canvas which holds the phase response of a filter described by a set of poles and zeros.

Hierarchy: PhaseResponse -> G2Dint -> Graph2D -> java.awt.Canvas

public PhaseResponse(double frequency, PoleZero pz_array_a[], int type)

this method constructs a PhaseResponse canvas

public boolean draw_analog_response(int length)

this method draws the frequency response of the analog poles and zeros on the screen

public boolean draw_digital_response(int length)

this method draws the frequency response of the digital poles and zeros on the screen

public boolean set_design_type(int new_type)

this is a method to set the design type. the design type will be set to the value passed in to new_type

public boolean set_pz_array(PoleZero pz_array_a[])

this method sets the pz_array to point to the input variable

public boolean set_frequency(double frequency)

this method sets the sample frequency or maximum frequency

public boolean update_screen()

update_screen sets the update_flag causing a repaint of the canvas.

public void run()

run is the body of the thread. it will update the display area when necessary. The necessity is based on the condition of the update_flag.