MTHE 332/393 Lab Manual

June 2, 2022

Preface

This lab manual, and the labs described herein, have been developed over many years by many people. The labs are intended as a companion to a course taught from the draft book *A Mathematical Introduction to Classical Control* by Andrew D. Lewis, and which has been used for some years as the text for MTHE 332 (formerly MATH 332).

The genesis of these labs were experiments performed in The Cave in Jeffery Hall, beginning in the early 1990's, and developed by Jon Davis and Ron Hirschorn. These labs used a crude Linux real-time implementation, and were a true adventure for students. Support for this development was provided by the "Access to Opportunities" programme of the Ontario government, the BED Fund from the Queen's Engineering Society, and the Department of Mathematics and Statistics. The current set of labs, i.e., the labs described in this manual, were developed by Andrew Lewis and Neill Patterson in the early 2000's, still on the Linux platform. The move to equipment supplied by Quanser Consulting, a Canadian company developing real-time control systems for education and research, was undertaken by Martin Guay in 2004 with the aid of a grant from the McConnell Foundation. The first working version of this lab manual was produced by Bernard Chan in the summer of 2004. This version of the lab manual was used and further developed by the lab TAs, Thomas Norman, Jeffrey Calder, Steven Wu, and Jack Horn, over the next few years. The alterations were smoothed during the summer of 2012 by Daniel Blair and Zachary Kroeze, and the labs are now being used jointly by the courses MTHE 393 and MTHE 332.

Table of Contents

Intentionally left blank.

MTHE 393 Project Intro

MTHE 393 consists of two components: a lab component, and a project component. It is the goal of the first four labs in this manual to teach you the skills and concepts necessary to successfully complete your project.

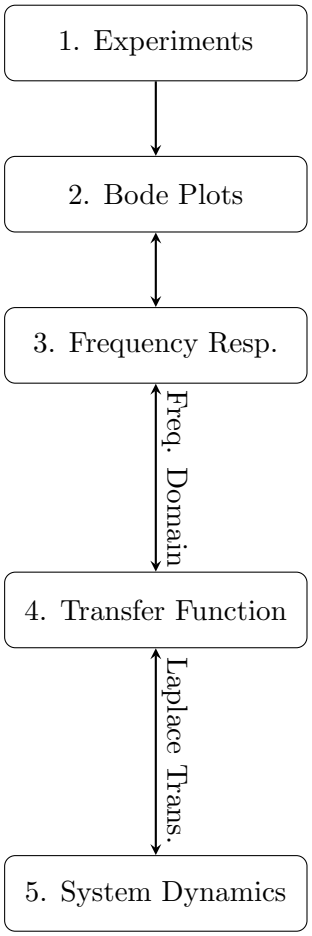

Figure 1: Project Outline

The project itself is fairly straightforward. Teams of students will be given a simulated "Black Box" system, and it will be your job to design a controller that will control the outputs of this system, given a set of performance specifications. Figure 1 gives a general outline for the steps you will be required to take in order to successfully do this. The techniques you will learn in these labs will be useful for your project.

1. In lab 2 you will be conducting experiments on the DC motor by applying various voltages and measuring the difference between the desired input, and the actual output (in this case, the outputs will be angular position and angular velocity of the motor). These will give you the necessary information to construct what is called a Bode plot.

2. A Bode plot is a graph of a systems frequency response. More specifically, several sinusoids of varying frequencies will be inputted into the system, then the magnitude and phase difference of the desired input vs the actual output will be measured, and plotted on a logarithmic scale. These plots are in the frequency domain, therefore they are plots of frequency (ω) vs magnitude (dB), or frequency (ω) vs phase difference ($rads$).

3. Once you have both Bode plots (magnitude and phase), it is then possible to determine your systems frequency response. The frequency response is in the form of a polynomial with various poles and zeros in the complex plane. The general shape of a Bode plot is dependent on the number of poles and zeros, and their location in the complex plane. Therefore, it is possible to determine the frequency response from your experimental bode plots.

4. The transfer function of a system is what relates the inputs and outputs. If the dynamics of your system are unknown, then you have what is called a *Black Box System*. When given a black box system, the transfer function is unknown, and you will want to determine an accurate model for it so that you can begin to control the system. To determine the transfer function from the frequency response, you must simply substitute the Laplace variable s in place of $j\omega$.

5. The dynamics of a given system are typically given by some differential equation. See equation 1.1 for the governing equation of the DC motor you will be using in the labs. To find a transfer function for this system, you need to work in the *Laplace* domain. This is done by creating a variable $s = j\omega$, and taking the Laplace transform. You may want to review your notes from MTHE 237 on this topic. Once you have taken the Laplace transform, you can assess the open-loop stability of the system, and the frequency response. Similarly, you can recover the governing equation of the system from the transfer function by simply taking the inverse Laplace transform. For your project, it is at this point that you will begin designing your controller.

Note that this intro may not be completely clear right away. You may find that as you work through the labs, and attend the weekly tutorials that the material will make more sense. This will also be a helpful reference guide throughout the project.

Part I

MTHE 393 Labs

Lab 1
Introduction to lab equipment

The purpose of this lab is to introduce the computer programs and the equipment you will be using in this course. You will simulate the operation of an open-loop motor scheme, illustrated in Figure 1.1. Hence, the differential equation governing the system is:

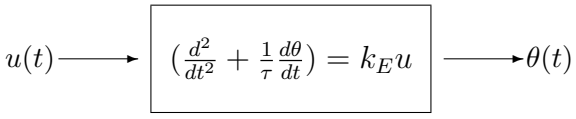

Figure 1.1: Open-loop motor schematic

$$\frac{d^2\theta}{dt^2} + \frac{1}{\tau}\frac{d\theta}{dt} = k_E u. \tag{1.1}$$

We are interested in the angle, θ, and the angular velocity, $\omega = \frac{d\theta}{dt}$, of the motor shaft. By the end of the lab, you will have enough data to calculate the motor time constant, τ, and torque constant, k_E.

1.1 Key Concepts

As the first lab is primarily used to have students familiarize themselves with laboratory equipment, it does not focus heavily on course related material. However, this lab will introduce you to certain applications of Matlab and Simulink which will be used continuously throughout all 9 labs in this manual. Simulink is a program that allows us to simulate a system, define an input to this system, and send the systems output to Matlab. In these labs our system will be a DC motor, to which we will be applying various inputs and observing the outputs. Simulink will also be used continuously throughout the project portion of the course.

You will be using Matlab throughout this course, so start using some best practices. Some useful tips:

1. *Always* start a script. Whether you are trying to generate plots, simulate dynamics, or do calculations, it is significantly easier to edit, re-use, re-run code in a script as opposed to the workspace.

2. Save your Simulink models to the cloud. This will save you from having the rebuild your model each week.

3. Use a semicolon ";" to surpress outputs.

4. You can create Section in your script using double percent signs.

5. The `plot` function can handle multiple x, y-tuples. Use this to your advantage. The only requirement is that each vector pair has to be the same *length*. You can have multiple y values for one x vector.

1.2 Prelab

It is assumed that the students of this course will have working knowledge of personal computers. Before you go into the lab, you should read the following:

- Appendix A: Matlab;
- Appendix C: Lab equipment;
- Appendix B: Simulink;
- Section 1.2 from the course text.

Before the lab, you must solve the ODE given in Equation 1.1, for a constant input $u(t) = 1$, using an appropriate technique. Remember that for an equation of the form $\frac{dy}{dt} + P(t)y(t) = Q(t)$, the integrating factor $\mu(t) = e^{\int P(t)dx}$. Or, you can use Laplace Transformations.

You will be expected to be able to look up material in the appendices during the course of the various labs, so it is best that you be familiar with what is in them. If you are already familiar with any of the topics, you may skip that section.

1.3 Procedure

The following steps should be followed to set up the simulink models to properly communicate with the hardware. You should perform the following steps *before* adding any blocks to your simulink model to avoid manually configuring each block in your model. Concerning any check boxes, if it is not explicitly stated that you should check a box then it *must* be left unchecked.

1. Create the directory

   ```
   C:\Documents and Settings\<Qlink ID>\My Documents\MATLAB
   ```

   if it does not already exist.

2. Start Matlab.

3. Type `simulink` at the prompt.

4. Click `File→New→Model` to create a new empty simulink model.

5. Build a Simulink model (the following steps will guide you through this) as shown in Figure 1.2. The model applies a constant voltage to the servomotor. The encoder is

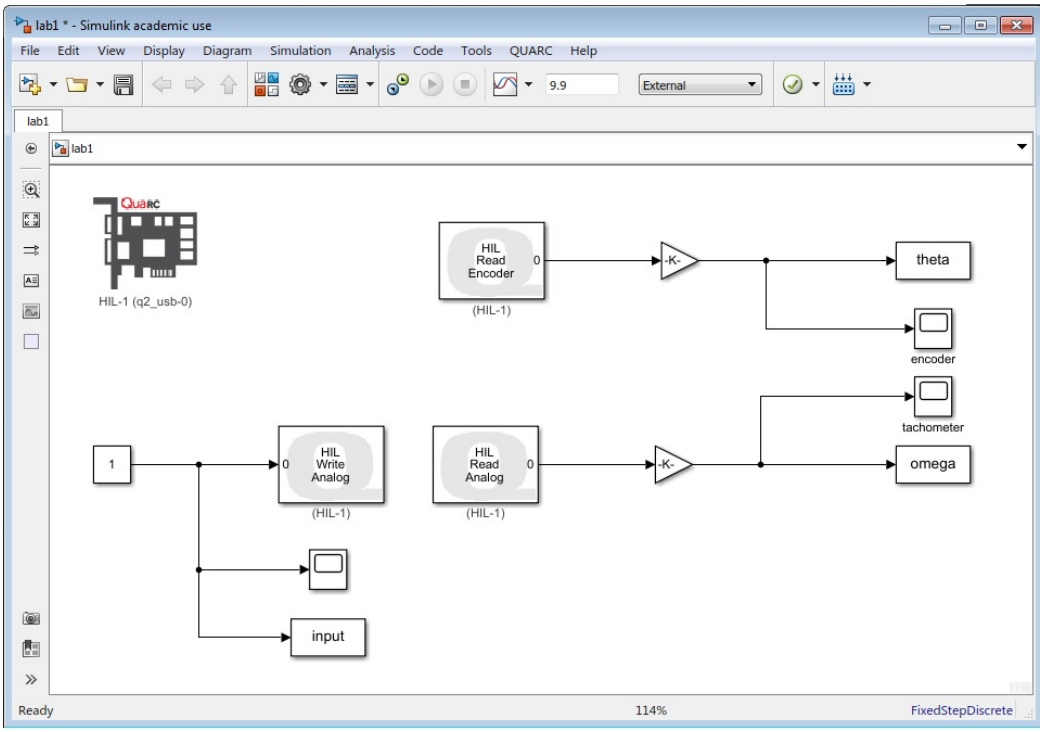

Figure 1.2: Simulink model for Lab 1

employed to acquire the angular position (θ) and the tachometer is used to acquire the angular velocity (ω) as functions of time.

6. Make a new folder on the hard drive under the name or number of your group and save your model under the name `lab1_name_of_your_group.mdl`. Save all files created (e.g., model file, plots) in each lab session in that folder. It might be a good idea to create a folder for each lab session as well.

7. Drag the `HIL Initialize` block from the library window into the model. You can find this block under:

   `QuaRC Targets→Data Acquisition→Generic→Configuration`

8. Double click on the new `HIL Initialize` block in your model to configure the parameters.

   (a) Main Tab

- Board Type = `q2_usb`

(b) Encoder Inputs Tab

- Encoder Input Channel = `[0]`
- Encoder Quadrature = `[4]`
- Encoder Frequency in Hertz = `[ ]`
- Initial Encoder Counts = `0`
- Check box `Set encoder input parameters at model start`
- Check box `Set initial encoder counts at model start`

(c) Analog Outputs Tab

- Analog Output Channels = `[0]`
- Initial Analog Outputs = `0`
- Final Analog Outputs = `0`
- Analog Outputs on Watchdog Expiry = `0`
- Check box
  `Set initial analog outputs when switching to this model`
- Check box `Set final analog outputs at model termination`
- Check box
  `Set final analog outputs when switching from this model`

9. Click `Apply` and then `OK` to close the properties dialog box.

10. Once the `HIL Initialize` block is set up properly, you can add blocks to your simulink model to read and write analog signals to the interface board. The main blocks of interest are `HIL Read Encoder`, `HIL Read Analog` and `HIL Write Analog`, which replace the old blocks `Encoder Input`, `Analog Input` and `Analog Output`, respectively. Consult the instructions to see which blocks to use in each lab. There are no `calibration`, `encoder`, or `tachometer` blocks. These blocks are simply "gain" or "scope" blocks which have been renamed. It is always best to refer to the block pictures instead of the block names. These blocks can be found at

    `QuaRC Targets`→`Data Acquisition`→`Generic`→`Immediate I/O`

    `Simulink`→`Commonly used blocks`

    When using the HIL `Read Analog` block, make sure that the correct channel is set (channel `0`).

11. The input and output channel numbers in the Simulink blocks should match the channels used on the terminal board. Refer to Figures 1.3, 1.4, and 1.5 for the correct channels.

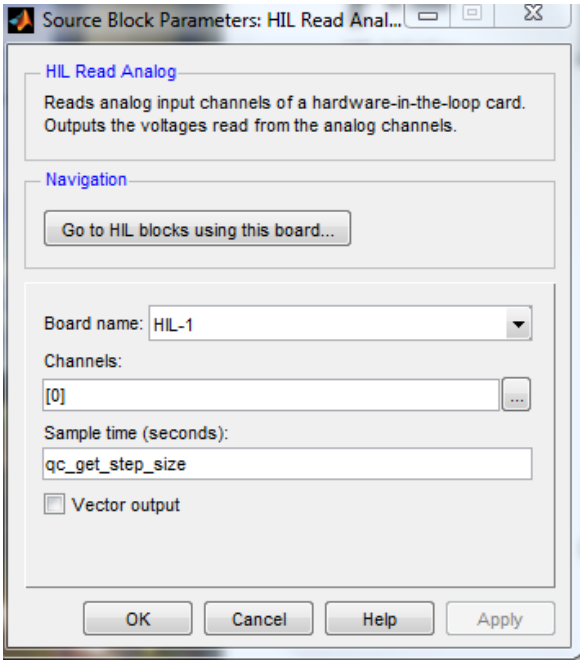

Figure 1.3: HIL `Read Analog` block settings

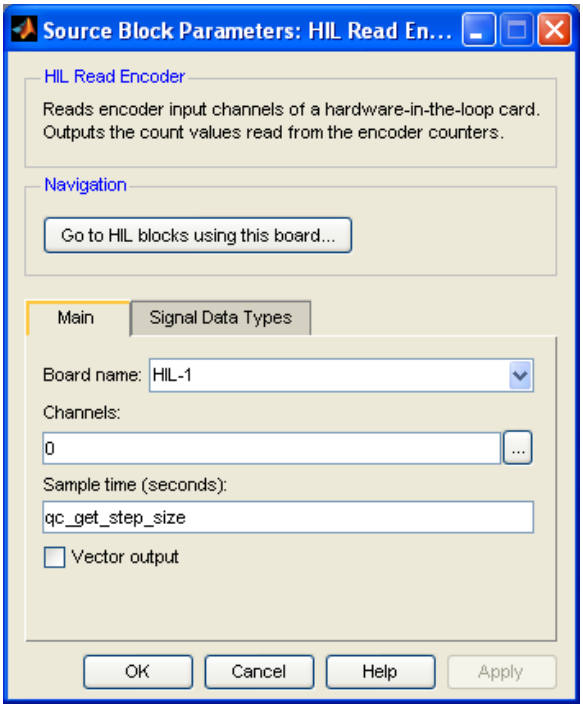

Figure 1.4: HIL `Read Encoder` block settings

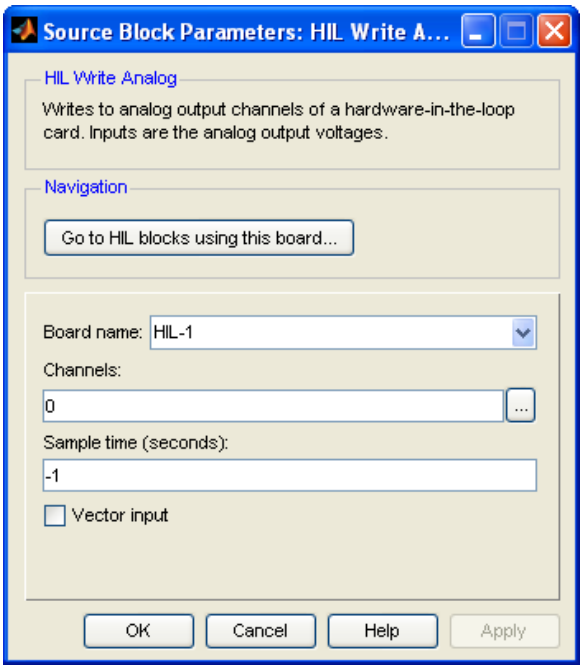

Figure 1.5: HIL `Write Analog` block settings

12. The calibration factors need to be set so that servomotor angular position and velocity are acquired in appropriate units. Double click on the `Gain` block for the encoder and set the gain to $-\frac{2\pi}{4096}$. Similarly, set the Tachometer gain to $\frac{100\pi}{63}$. These values can be found in Table C.1 in Appendix C.

13. The `To Workspace` block can be found at `Simulink`→`Sinks`. Drag this block into your workspace and connect it to the variable you wish to save. Double click on the block to configure it. Choose a good variable name and, in the `save format` drop down menu, select `Structure with time`.

14. Click on `Simulation`→`Configuration Parameters` from your Simulink screen (or `Ctrl+E`). Make sure you are using `fixed-step` integration and choose `ode 4` as your method.

15. Select `Set default options` from the `QuaRC` drop down menu.

16. Select the `External` mode option from the drop down menu in the toolbar. Note that `External` mode is only necessary for labs that use the Servomotor.

17. In the toolbar, change "in" to 5 and press enter. This is the simulation time. A note: the software seems to forget all data older than 10 seconds during the simulations, so for the purpose of this lab, we set the `Stop Time` to 5 seconds.

18. Double click on the `Encoder` and `Tachometer` blocks to open the plots. More details on viewing real-time results can be found in Appendix B.

19. Select `Builld from the \verb`QuaRC— drop down menu. Wait for building to be completed before preceding to next step. Progress can be seen in the Command Window. The keyboard shortcut for building is `Control+B`

20. Provided there are no compilation errors, select `Connect to Target` from the `Simulation` menu.

21. Select `Run` from the `Simulation` menu, or click the black play button in the toolbar. The keyboard shortcut for this is `Control+T`.

22. Data will automatically be saved from the `To Workspace` blocks.

23. Plot it with the command

    ```
    plot(varname.time,varname.signals.values)
    ```

    replacing "`varname`" with the variable name you chose when configuring the `To Workspace` block. Use the `plot` command in Matlab to plot data from the `Encoder` and the `Tachometer`. Details on plotting in Matlab are discussed in Appendix A. Remember to give the plots appropriate title and axis labels and print these plots. What is the steady state angular velocity? Are the results from these plots as you expected?

24. Now that you have obtained the steady-state value from the angular velocity plot, you are ready to determine the actual value of the motor time constant, τ, and the torque constant, k_E. Recall that solving the differential equation (1.1) with zero initial condition yields

$$\begin{aligned} \omega(t) &= \dot{\theta}(t), \\ \omega(t) &= k_E\tau(1 - e^{-t/\tau}), \end{aligned} \tag{1.2}$$

    and so the steady state value is just $k_E\tau$.

25. First, determine the steady state value and the constant τ from the angular velocity plot. You can find τ by using the steady state value and finding the value of $\omega(t)$ when $t = \tau$.

26. Next, determine k_E using Equation (1.2) and the constants obtained in the last step. You might need to zoom in to the appropriate portion of the graph to see the result clearly.

27. Once you have confirmed with the TA that you have obtained the correct value of the constants, print a copy of the plot that you zoomed in on.

28. Save all files in the folder you created. Hand in all the plots you printed during this lab session and along with it the work to show how you have obtained the two constants. Please make sure the names and student numbers of all your group members are on the first page.

The constants obtained in this lab will come in handy in the future, so make sure you check with the TA that you have obtained the correct (or reasonably close) value before you leave.

Save your Simulink model to an accessible location. You will need it next week.

1.4 Deliverables

Prepare a brief write up describing what you learned from this lab. This does not need to be a formal report, but all material should be presented in a clear and logical manner, with concise descriptions where necessary. Include the following/answer the following questions:

1. Plots of motor position (θ) and velocity (ω) with respect to time. Make sure you label your axes appropriately.
2. What is the steady state angular velocity (in rads/s) of the motor? Does this correlate to the value obtained using the encoder plot?
3. Determine the motor constant τ. Include a plot at t = τ (use proper units).
4. Determine the motor torque constant, k_E (include units). Show your calculations.

Lab 2
Frequency response and the Bode plot

In this lab you will examine the relationship between the impulse response, the transfer function, and the frequency response. You will also compute the Bode (pronounced "Boh-dee") plot of the open-loop motor scheme depicted in Figure 2.1.

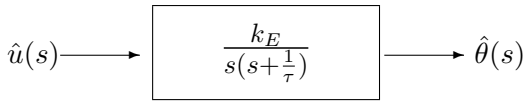

Figure 2.1: Open-loop motor schematic in frequency domain

2.1 Key Concepts

In this lab we will be determining the relationship between the inputs and outputs of a system. Here are a few key topics:

1. The *Dynamics* of a system govern the input/output relationship of a system. These dynamics can be either known, or unknown. In lab 1, we observed how our system responded to a simple constant input, however we wish to know how it will respond to *all* functions. This is done by observing what is called the "Frequency Response" of the system.

2. The frequency response of a system is determined by first constructing a Bode Plot. This is done by inputting different sine waves with varying frequencies, and measuring the magnitude and phase difference between the input and the output. For example, Figure 2.2 shows the input and output of the system for a sine wave with $\omega_u = 5$ rads/sec. Notice how the magnitude of the output is larger, and the peak time is shifted by roughly 0.05s. This observation will be made for sine waves with frequencies varying from 0.5 rads/sec to 300 rads/sec, or possibly even higher.

3. Once we have sufficient data for the magnitude and phase difference, two plots will be created (frequency vs. magnitude, and frequency vs. phase angle) using a logarithmic scale. These are your Bode Plots.

4. It is important to remember how these relationships relate to Figure 1. In this lab we are conducting experiments as if we did not already know the dynamics of the system. This will be how you determine the frequency response of the system you are given for your project, which will allow you to find a model transfer function for the "Plan".

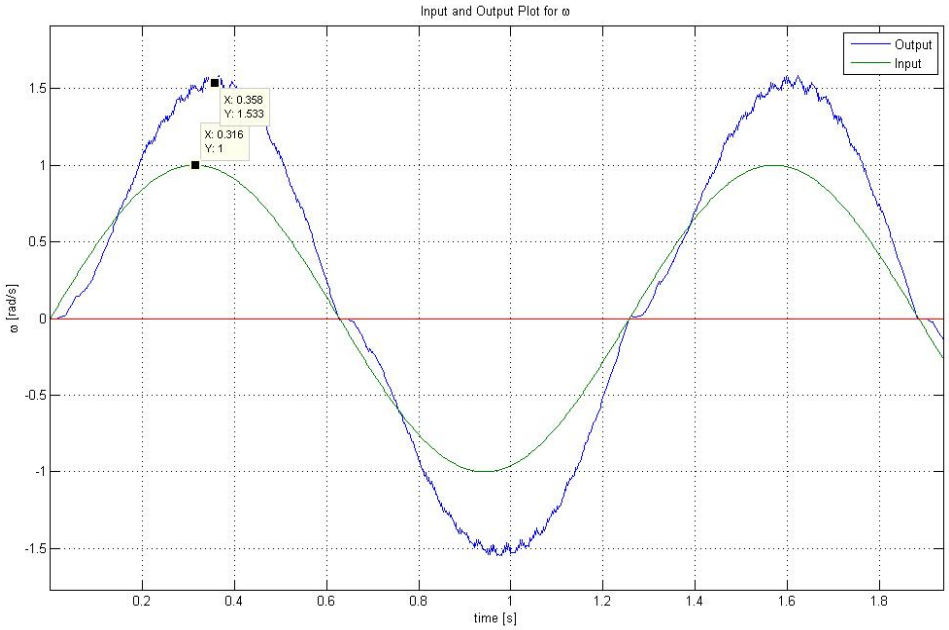

Figure 2.2: Matlab generated plot of the input and output of the system for $\omega_u = 5$ rads/sec

2.2 Prelab

1. Please review Sections 4.1, 4.2, and 4.3 of the course notes for relevant on frequency response and the production of Bode plots.

2. Compute the Bode plot (by hand) of the motor system when the output is the motor angle θ. Recall that the transfer function of that system is

$$T(s) = \frac{k_E}{s(s + \frac{1}{\tau})}.$$

   Use the motor time constant, τ, and the torque constant, k_E, obtained in Lab 1. Please consult with your TA to make sure you have the correct values.

3. Repeat the above step when the output is motor velocity ω. Recall that the transfer function of that system is

$$T(s) = \frac{k_E}{(s + \frac{1}{\tau})}.$$

2.3 Procedure

2.3.1 Preliminaries

In this lab you will be gathering data that will be used to construct the Bode plot. Determining the magnitude of the output is straightforward, but the phase between the input and the output sinusoids is somewhat more troublesome, but still no match for the wits of seasoned veterans of the Apple Math program.

To determine the phase difference between two sinusoids, we need to compare two equivalent points on each sinusoid. Convenient points to consider are peaks of each sinusoid or where each sinusoid is zero. Since the magnitude of each sinusoid can differ, it is more convenient to compare where each sinusoid is zero. From your plot, measure the time difference between when the input sinusoid is zero and when the output sinusoid is zero. Knowing the time difference and the frequency of the sinusoids, you can use the fact that $\omega = \frac{d\theta}{dt} = \frac{\Delta\theta}{\Delta t}$ to calculate the phase difference as $\Delta\theta = \omega\Delta t$.

You will find that a good way to organize your data is in a table of the following format:

ω	Magnitude	Gain (dB)	Zero-Time Difference	Phase (rad)	Phase (deg)

Table 2.1: Data collection table

2.3.2 MATLAB

In this part of the lab you will use Matlab to produce the Bode plots of the mathematical model of the system.

1. Start Matlab and enter the transfer function corresponding to each output. Use the `tf` command to generate the transfer function in the Matlab command window, then produce the Bode plots of each transfer function using the `bode` command.

   Please refer to Appendix A on how to use the `tf` and `bode` command in Matlab.

2. Do not print these plots, but do not close Matlab either.

2.3.3 Bode plot

We will now turn our attention to constructing the Bode plot experimentally, which we will accomplish by examining the output of the motor given a periodic input. The frequency response of a system determines how that system responds to a harmonic input of frequency ω_u. For this lab, we will take our harmonic input to be the simple sinusoid $\sin(\omega_u t)$.

From your work in Section 2.2, you might guess that constructing a Bode plot for the motor when the output is the motor angle will be rather difficult, in which case you would be correct. However, we will have a quick look at the angular response of the motor to see if it is consistent with the Bode plot produced in Matlab.

1. We begin our exploration of the frequency response with a crude experiment. We would like to see what happens over a large range of frequencies, so we would like ω_u to increase as t increases. A simple way to do this is to define the input function to be $5\sin(t^2)$.

2. Follow Figure 2.3 and build a Simulink model to implement this signal. *Alternatively, you can modify your model from lab 1.* The `Fcn` block can be found in Simulink in

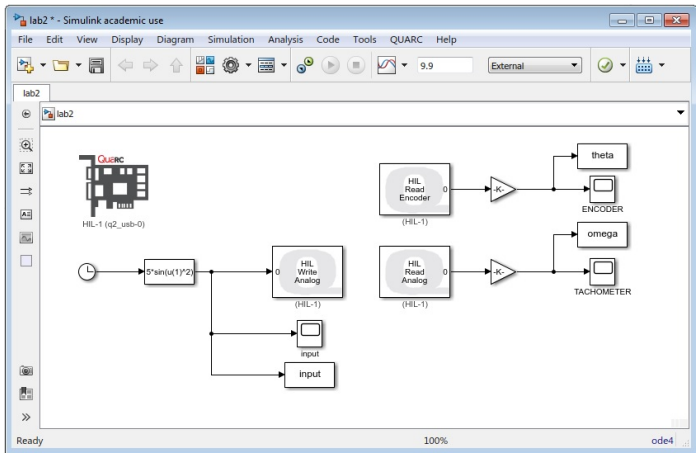

Figure 2.3: Simulink model for the implementation of the signal $5\sin(t^2)$

the `User-Defined Functions` section. Double-click on your `Fcn` block to enter the function `5*sin(u(1)^(2))` where `u(1)` represents the time. The time is obtained by introducing the `Clock` block as an input to the function. The output of the function block is simply connected to the `Write Analog` block. Do not forget to change the `solver` to `ode 4` in `Configuration Parememters` (`Ctrl+E`). Also, change the gain constants to the desired units, as in Step 12 from Lab 1. Note: these values can be found in Table C.1 in Appendix C.

3. Change the simulation time in the toolbar to 9.9 seconds and press enter.

4. Build and run the Simulink model. Plot the input function u, and the output θ *simultaneously.* You will notice that θ is not as well behaved as you might like, but all you should be concerned with is a rough guess of relative magnitude and phase with respect to the input function u. Does this concur with the Bode plot you computed with Matlab? Specifically, comment on the magnitude and the phase at low and high frequencies.

5. Repeat Step 4 but plot the output variable $\dot{\theta}$ and u instead of θ and u.

6. We will now systematically construct the Bode plot of the motor when the output is the angular velocity ω using experimental data. Doing so will require information about the magnitude and phase of output sinusoid at various frequencies. To do this you replace the `Clock` and `Fcn` blocks by a `Sine Wave` Block. After you have connected the `Sine Wave` block to the `Write Analog` block, you can enter the parameters of the sinusoid you would like to implement. For the generic signal, $\sin(\omega_u t)$, you may enter the parameters as highlighted in Figure 2.4. In this case, the variable `omega`, is

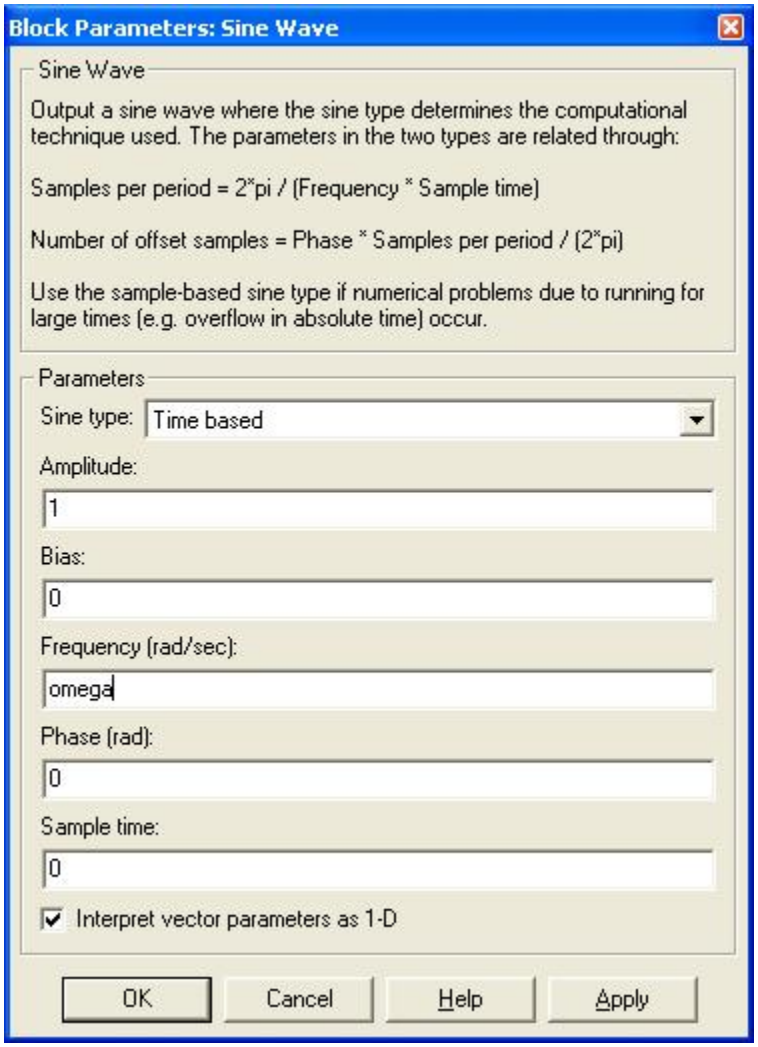

Figure 2.4: Configuration for the implementation of the signal $\sin \omega_u t$

undetermined. The value of ω_u must be specified at the Matlab command line before each run. The amplitude of u should be set to 1.

7. Build the Simulink model. For various values of ω_u, run the system and plot the output variable, $\dot{\theta}$, and the input variable, u. A "good" Bode plot can be created using 10–15 well chosen ω_u values. Use the Bode plot generated using `tf` to pick proper omega values. Note that you do not need to rebuild the Simulink model for each new value of ω_u. A good range of ω_u values would be between 0.5 and 300, keeping in mind that this is plotted on a logarithmic scale.

8. Determine the magnitude of the output sinusoid by recording its peak amplitude. It is recommended that you zoom in and out as needed, to ensure a reasonable degree of accuracy.

9. To determine the phase difference, you will find it most accurate to look for zero-crossings, as explained above. You will find that creating an output variable set to zero is a helpful visual aid in find zero crossings. Zoom in to accurately observe the time difference when u is zero and when $\dot{\theta}$ is zero. When you are recording the phase differences, you should be careful that you are recording zero-crossings when both functions are either increasing or decreasing. This is crucial to keep in mind for higher frequencies. This procedure might sound confusing, but usually it is straightforward.

10. Calculate the phase difference as described in the introduction.

11. With a reasonable amount of data, plot by hand the Bode plot of the motor when the output is angular velocity, ω. You could also use a spreadsheet program to organize and plot your data.

    When you have completed the lab, make sure you save your files in the folder you created in the Lab 1.

2.4 Deliverables

Prepare a brief write up describing what you learned from this lab. This does not need to be a formal report, but all material should be presented in a clear and logical manner, with concise descriptions where necessary. Include the following / answer the following questions:

1. Include the Matlab generated bode plots for θ and ω. Use your k_E and τ values from lab 1

2. Plots of your motor angular position and velocity compared to the input function $u = 5\sin(t^2)$ (do this on the *same* plot). Comment on the general trend of the magnitude and phase shift of the outputs. Does this agree with the bode plots you generated?

3. Include tabulated data from Table 2.1

4. Include experimental bode plots for both magnitude and phase difference. Ensure the plots *look* like Bode Plots.

Lab 3
Impulse response and the transfer function

In this lab you will be examining the impulse response and the transfer function of the motor, and the relationship between the two. Throughout this lab we will be using the open-loop motor scheme, as depicted in Figure 3.1, but we will be using a constant DC

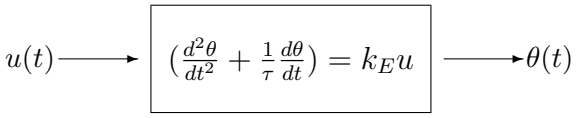

Figure 3.1: Open-loop motor schematic

voltage. We will use Simulink to simulate and implement the system.

3.1 Prelab

The impulse response can be loosely defined as the response of the system to a "jolt" input. While it is physically impossible to achieve a true impulse, it is still interesting to study because it reveals much about the properties of the dynamic system. Material on impulse response can be found in Sections 2.4 and 3.5 of the course notes.

To make sure that you have a reasonable understanding of the idea of an impulse response, describe, in words, what happens when a jolt input (a very short step response) is applied to the motor when:

1. the output is the motor angle, θ;
2. the output is the motor angular velocity, ω.

Accompany each with a rough sketch. You do not have to actually compute anything, just use your intuition.

3.2 Procedure

3.2.1 Impulse Response

1. For the open-loop scheme, write out the state equation:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u,$$

identifying $\boldsymbol{x}$, u, $\boldsymbol{A}$, and $\boldsymbol{b}$.

2. Determine the output equation:

$$y = \boldsymbol{c}^t \boldsymbol{x} + \boldsymbol{D}u, \tag{3.1}$$

identifying $\boldsymbol{c}$ and $\boldsymbol{D}$ when

- the output is motor angle, θ, and
- the output is the motor angular velocity, ω.

3. Prepare the Simulink model shown in Figure 3.2.

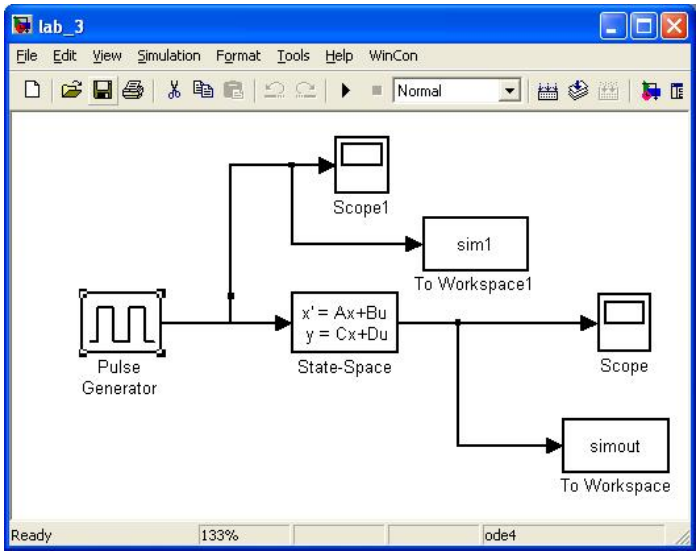

Figure 3.2: Simulink model for Lab 3

4. As in Lab 9, enter the motor state equation using the values of k_E and τ that were estimated in Lab 1 (make sure that you specify the appropriate output). Double-click on the `Pulse Generator` icon and enter the information as shown in Figure 3.3. The `Pulse Generator` introduces the impulse function

$$u_\epsilon(t) = \begin{cases} \frac{1}{\epsilon}, & t \leq \epsilon, \\ 0, & \text{otherwise.} \end{cases}$$

every ten seconds.

Recall that the impulse function is defined as $\lim_{\epsilon \to 0} u_\epsilon$ (the limit being. . . er. . . in the space of distributions). Clearly, this is physically impossible to achieve, but we will approximate as best we can. The highest voltage that can be provided by the power supply is 5 volts, so $\epsilon = \frac{1}{5}$ is the lower limit for our system. Pulse width is calculated using the following formula:

$$\text{Pulse Width} = \frac{\frac{1}{\epsilon}}{\text{period}} 100$$

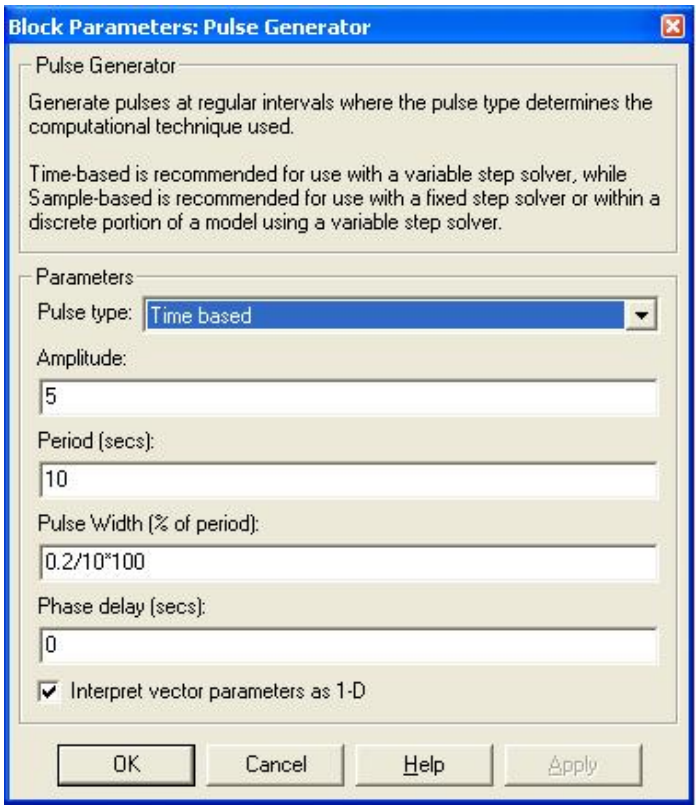

Figure 3.3: `Pulse Generator` configuration for Lab 3

5. Run the Simulink model. Define the initial conditions to be zero. The final time is set to 10 by default. You may change it by opening the window

   `Simulation→Configuration Parameters`

   and entering the required time in the `Stop time` box. Start with $\epsilon = 2$ and reduce the value of ϵ until you see no further change in the process response.

6. Plot the impulse response for each output described above, using an appropriate title for each.

7. You will now introduce an impulse into the real system by providing a "jolt" by hand. This can be done by tapping the gear. Prepare the Simulink model shown in Figure 3.4. Note that there is no `Write Analog` block required in this case since you are providing

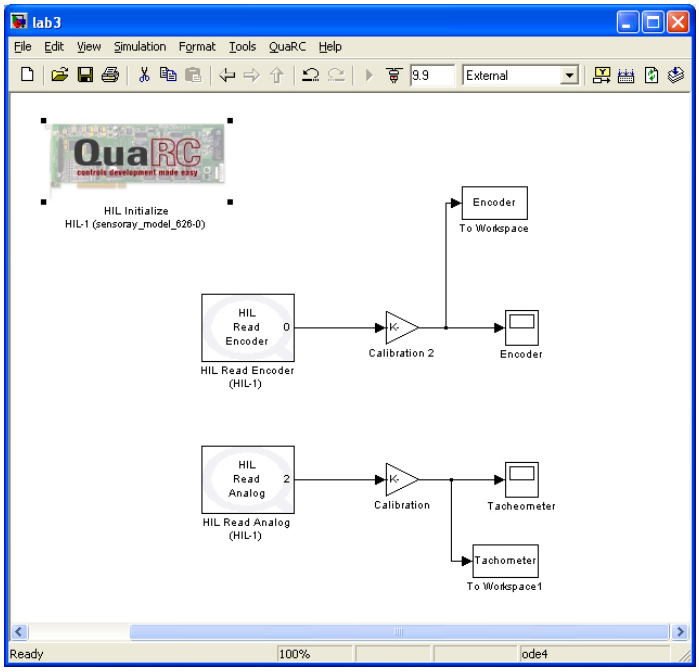

Figure 3.4: Another Simulink model for Lab 3

   the input action. Using Simulink, build the model. Under the Simulation menu, select `Connect to Target`. Then choose `Start Real Time Code`. Quickly give the motor a "jolt" by giving it a quick twist. Clockwise is taken as the positive direction.

   The `Read Encoder` and `Read Analog` parameters are the same as in Lab 1.

8. Plot the response of each state variable while adding an appropriate title to each plot.

9. Comment on the similarities and differences between the impulse response and the response of the motor to a hand-powered impulse.

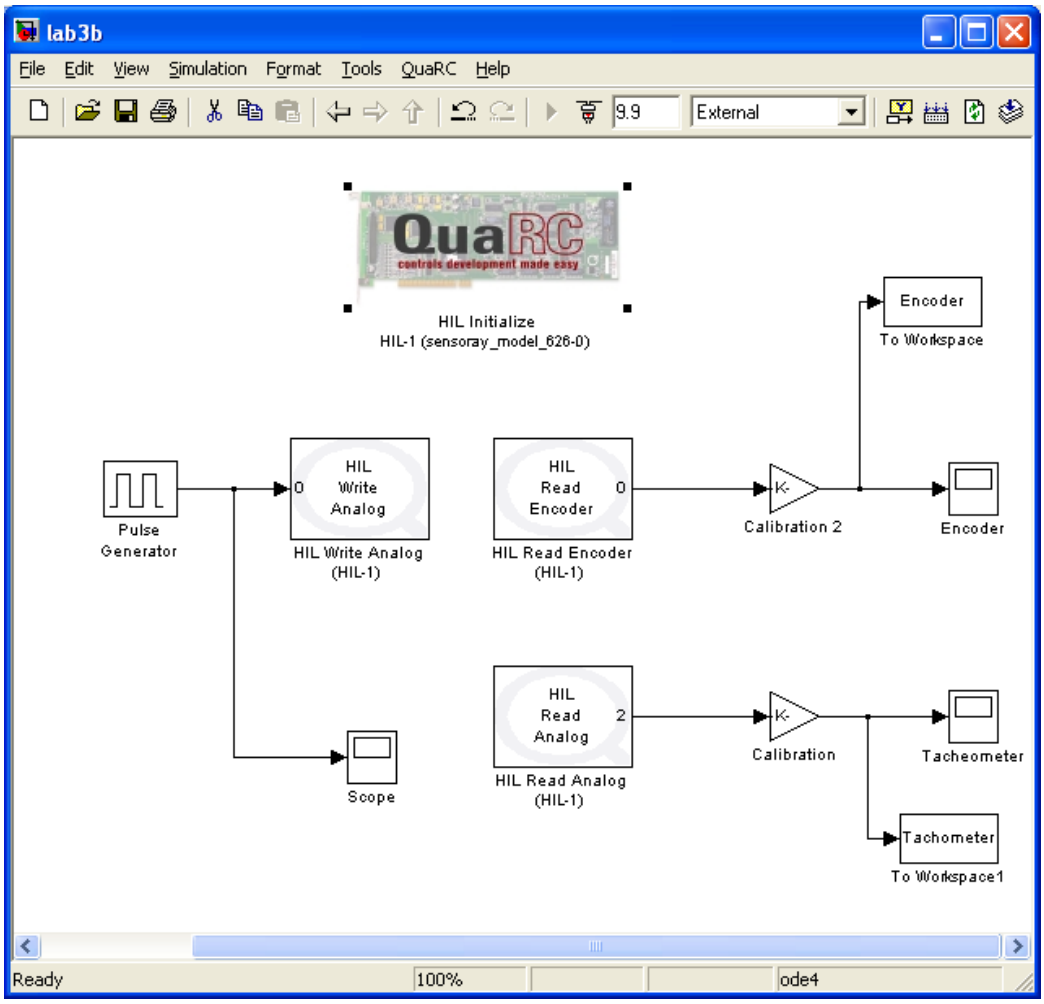

Figure 3.5: Simulink model for Lab 3

10. To obtain a "better" impulse, we introduce the `Pulse Generator` model as shown in Figure 3.5. Note that the analog output icon has been added since the impulse will be added automatically in this case.

11. Build and run the model with $\epsilon = 1$.

12. Run the model again with $\epsilon = \frac{1}{5}$.

13. Plot and print the response of the motor angle (encoder input) and the motor velocity (analog input). How do these compare with the simulation impulse response? With the response due to hand-powered impulse?

3.2.2 Transfer Function

In this part of the lab, you will work in the Laplace transform domain to analyze this system. Recall that the transfer function $T_\Sigma(s)$ is defined, for example, as the Laplace transform of the impulse response of the system.

1. Compute the transfer function from the standard form, $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u$, $y = \boldsymbol{c}^t\boldsymbol{x} + \boldsymbol{D}u$, when

   - the output is the motor angle, θ, and
   - the output is the motor angular velocity, ω.

2. Compute each transfer function again, this time by directly computing the Laplace transform of the impulse calculated in Section 3.2.1.

   When you have completed the lab, make sure you save your files in the folder you created in the Lab 1.

Lab 4
BIBO Stability of systems

A system is said to be bounded-input, bounded output stable (BIBO stable) if, given a bounded input, it produces a bounded output. In this lab you will be using Matlab and Simulink to examine the BIBO stability of several systems.

4.1 Prelab

Before you go into the lab, you should read the following:

- Sections 5.1 and 5.2 of the course notes on system stability.

4.2 Procedure

In this part of the lab, you are given three different control systems. For each system you will prepare a Simulink model to analyze the outputs of the system to various bounded inputs, and examine their "impulse response" (even keeping in mind that the true impulse response is a physical impossibility).

1. Consider the linear system:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \boldsymbol{x} \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \boldsymbol{x} \end{aligned}$$

   and determine $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}^t, \boldsymbol{D})$.

2. Use the `impulse` command in Matlab to produce the impulse response of the system (recall that $h_\Sigma(t) = \boldsymbol{c}^t \mathrm{e}^{\boldsymbol{A}t}\boldsymbol{b}$). Comment on the feature of the impulse response that determines BIBO stability. Do you expect the system to be BIBO stable?

3. You can calculate the transfer function in several ways: take the Laplace transform of the impulse response, noting that the systems you are given are all in controller canonical form; using the `tf` command in Matlab. Choosing whichever method you like, calculate the transfer function, and comment on its poles. Is this consistent with your prediction of BIBO stability?

4. Prepare the Simulink model shown in Figure 4.1. You could use the model in Lab 3 if you saved it. They are the same.

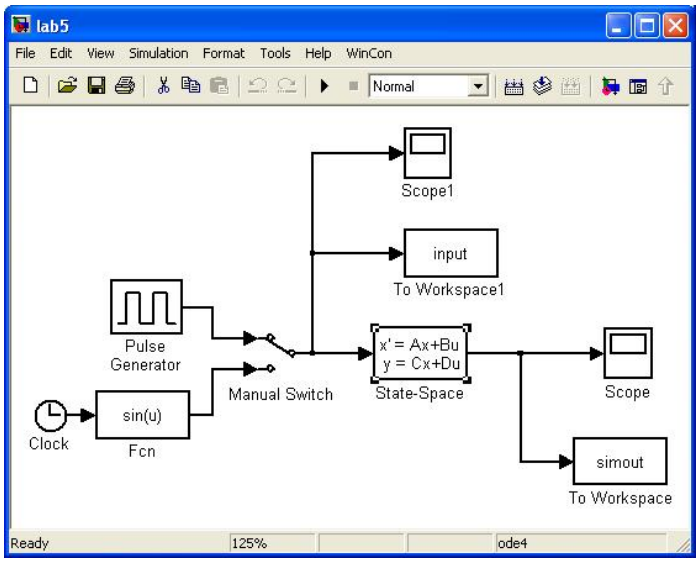

Figure 4.1: Simulink model for Lab 4

5. If you determined that the system is not BIBO stable, modify your Simulink model by replacing the `Pulse Generator` with the `Fcn` block and specifying a bounded input that will produce an unbounded output. If the system was determined to be BIBO stable, give the system any bounded input you desire. In either case, print a plot of the output variable y and the input variable u.

6. We will now examine the impulse response of the system. You can use the same Simulink model generated above and proceed as in Lab 3. Recall that the `Pulse Generator` block introduced the impulse function

$$u_\epsilon(t) = \begin{cases} \frac{1}{\epsilon}, & t \leq \epsilon, \\ 0, & \text{otherwise} \end{cases}$$

at every specified period.

Run the Simulink model with the initial conditions set to zero. The default final time is set to 10s by Simulink. You may change it by opening the window `Simulation`→`Configuration Parameters` and entering the desired time in the `Stop Time` box. Larger stop times may be useful here. Reduce the value of ϵ until you see no further change in the process response. Show at least three impulse responses using different values of ϵ and note the changes between them.

7. Repeat the procedure for the following two models:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 2 \end{bmatrix} \boldsymbol{x} \end{aligned}$$

and

$$\begin{aligned} \dot{\boldsymbol{x}} &= \begin{bmatrix} 0 & 1 \\ 4 & -3 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ y &= \begin{bmatrix} 0 & 1 \end{bmatrix} \boldsymbol{x}. \end{aligned}$$

8. In Lab 3 you examined the impulse response of the motor system when the output of the system was the motor angle, θ, and the motor velocity, ω. Using the data you gathered in that lab, comment on the BIBO stability of both systems, commenting on the impulse response, and poles of the function.

   When you have completed the lab, make sure you save your files in the folder you created in Lab 1.

Part II

MTHE 332 Labs

Lab 5
Feedback and PID control

In this lab you will be examining the effects of feedback on system performance. In particular, you will design a control, R_C, for the motor using the principles of PID control as shown in Figure 5.1.

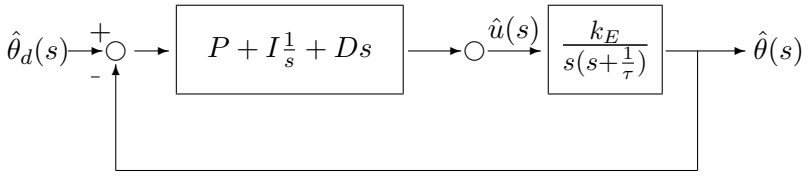

Figure 5.1: PID control system

5.1 Key Concepts

In this lab, you will be implementing a PID controller into a closed loop system. The main issue with open loop systems is that we only have control over the reference trajectory, and therefore cannot account for disturbances. Now, if we were to use feedback, we could attempt to control the *error* signal, which is the difference between the reference trajectory (i.e. desired angle, or state of the motor) and the measured output.

A goal of a standard PID controller is to tune the system to behave a certain way by using various constants to correct the error signal. The three contstants of a PID controller are *Proportional*, *Integral*, and *Derivative* controls.

- **Proportional** control acts on the present value of the error signal. This is the most dominant of the three terms, but it can also leave some steady state error.

- **Integral** control accounts for the past values of error which is accumulated over time. This term will eliminate the steady state error that is left behind by the proportional control term, and also reduce rise time.

- **Derivative** control looks at the rate of change in the error signal, and attempts to correct for possible future values of error. For example, if the system is rapidly approaching its reference trajectory, (i.e. the error signal is rapidly approaching zero) the system will be able to slow down and avoid overshoot. Derivative control helps improve the settling time and stability of the system.

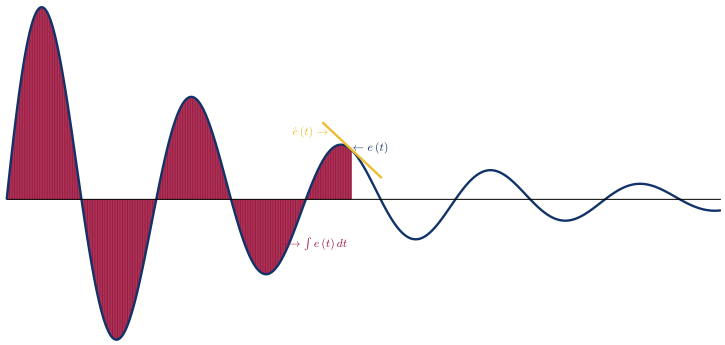

Figure 5.2: Arbitrary error signal with a PID controller acting on it

- Figure 5.2 illustrates the concepts of a PID controller. You have some error signal, where the proportional term acts on the current state of the system, the integral term sums up all previous error, and the derivative term attempts to predict and control future error.

It is important to remember that when using PID controllers, there will always be an element of compromise within your system. For example, the ideal system will have a very low rise time, with little to no steady state error or settling time, and no overshoot. However, this is very unrealistic in the real world. If you were designing a highly precise robotic arm, you may need to tune your controller in a way that there is practically no steady state error, but in order to do so, you will need to have a higher rise time (i.e. the arm will move slowly, but it will go exactly where you need it). On the other hand, you may have a system that requires a quick reaction time, for which you may need to accept that there could be overshoot, or some steady state error.

5.2 Prelab

Before you go into the lab, you should read the following:

- Sections 6.3 and 6.5 of the course notes.
- Learn the definitions for rise time, settling time, overshoot, steady-state value and steady state error.

Before the lab, complete the following steps:

1. Using the Simulink model shown in Figure 5.3, and identify the corresponding components from Figure 5.1, i.e. what block acts as the plant, what is the reference trajectory, input, output, etc.

2. Determine the closed loop transfer function.

3. Determine the location of the closed loop poles when using P, I, D controls individually. What condition is required on the poles such that the system is BIBO stable?

5.3 Procedure

1. Prepare a Simulink model to implement a PID controller as shown in Figure 5.3. *Modify your model from lab 1 or 2.* The `PID` block can be found in the `Simulink` menu under the `Continuous` section parameters.

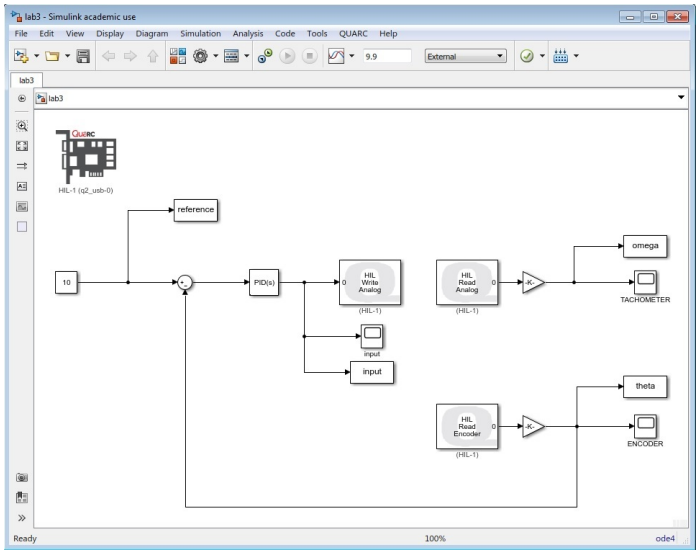

Figure 5.3: Simulink model for the implementation of PID control

   As shown in Figure 5.4, the `PID` block contains three parameters: P is the proportional gain, of the controller; I is the integral action parameter which is equal; the D term provides the derivative action.

2. Set the desired angle to 10 radians. The desired input is entered as shown in Figure 5.3 in the `Constant` block. Remember to use the appropriate gain values from Table C.1 for the encoder and tachometer to show results in radians. Do not forget to change the `solver` to `ode 4` in `Configuration Parameters` (`Ctrl+E`). Build and run the Simulink model using only the proportional term. Comment on the effects of using various values of P. In particular, comment on the overshoot, rise time, settling time, and the steady-state error for different values of P, and be sure to tabulate this data (a table in Excel is an efficient way to do this). Use at least three different values of P.

3. Set D and I to 0 and increase the value of P so the system oscillates about the desired value of 10 radians without (seemingly) decreasing in magnitude. Print a copy of the

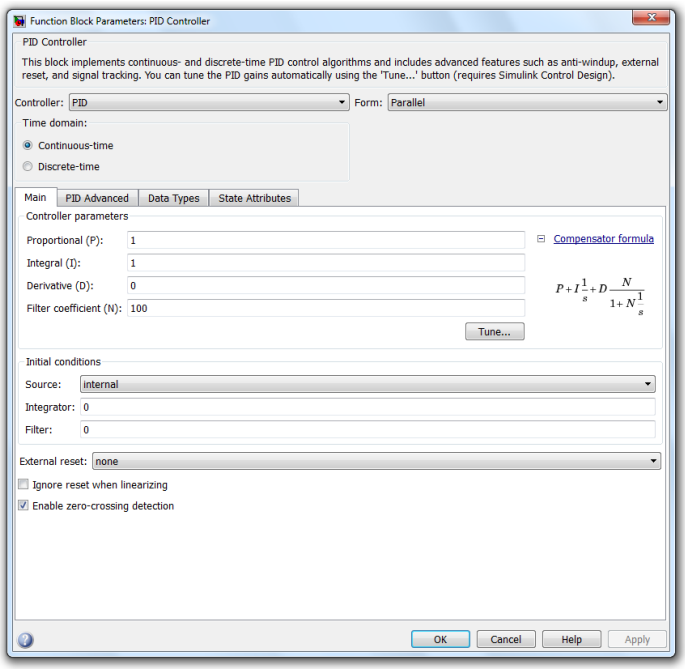

Figure 5.4: Screen shot of the `PID Controller` block

encoder output at that value and print another plot with P having a value slightly less than this value.

4. Now reset your P value to the "good" value (i.e. not the value that makes the system oscillate about 10 radians, but the value where you get a "desirable" response). Test various D values while keeping P constant, and comment on overshoot, rise time, settling time, and steady-state error. Again, tabulate this data.

5. Decide on some satisfactory values of P and D, and run the system again, using the integral control as well. Comment on the overshoot, rise time, settling time, and steady-state error for various values of I, and tabulate the data.

6. Run the system using only the integral control term (i.e., set P and D to zero) and comment on the response of the system. Using the transfer function, determine the roots of the characteristic polynomial. What insight does this give into the behaviour of the system using only integral control?

7. We will now use a periodic saw-tooth function as our input. To do this, replace the constant block by the block by the `Repeating Sequence` block under the `Simulink`→`Sources` menu. The first parameter, `Time Values`, indicates the switching times within one period. The second parameter, `Output Values`, indicate the values of the output of the block at the switching times considered. So, for example,

a saw-tooth of amplitude 1 and period 2 would have `Time Values` set to `[0 1 2]` and `Output Values` set to `[1 -1 1]`.

8. Run the system with various combinations of proportional, integral, and derivative control. Once you have a controller that tracks the input reasonably well, print a plot of the angle and the desired trajectory.

9. Change the reference angle trajectory to add more switching times within a period. To make things interesting, have the desired angle behave as four different functions on different intervals. This can be achieved by preparing the Simulink model shown in Figure 5.5. *You do not need to use the exact same functions as shown in the figure.* Pick four functions that you think would be interesting. In this model, a number of

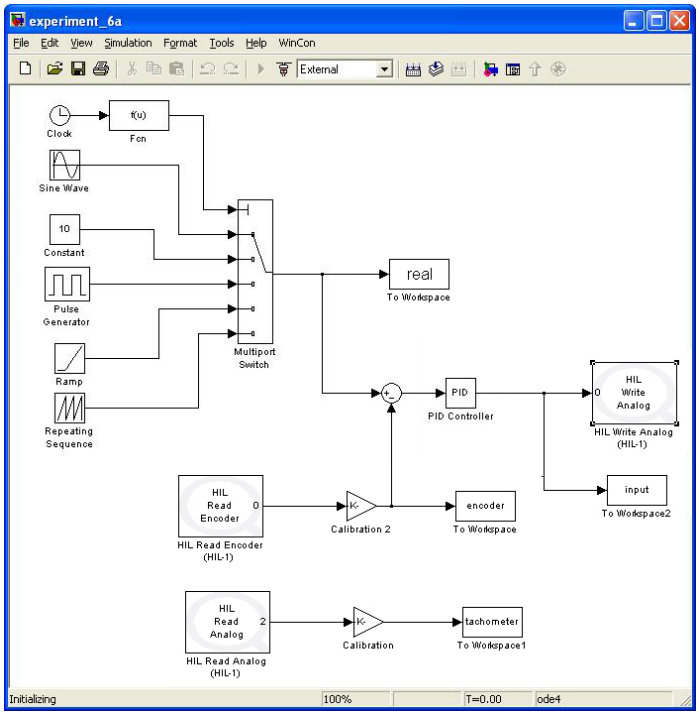

Figure 5.5: Simulink model for the implementation of multiple inputs and PID control

possible sources have been introduced along with the switching logic. The switching logic is based on the value of the function, `4 sin(0.2*t)^{2}+1`. This function, although arbitrary, assumes a value between 1 and 5 which is associated with ports 1 through 5 of the `Multiport Switch` block. The switching function is entered using the `Fcn` block as indicated in Figure 5.6.

10. Run the system and prepare a plot comparing the angle and the desired angle trajectory, making sure to note the values of the PID coefficients. You can reference Appendix C for definitions of the previous concepts.

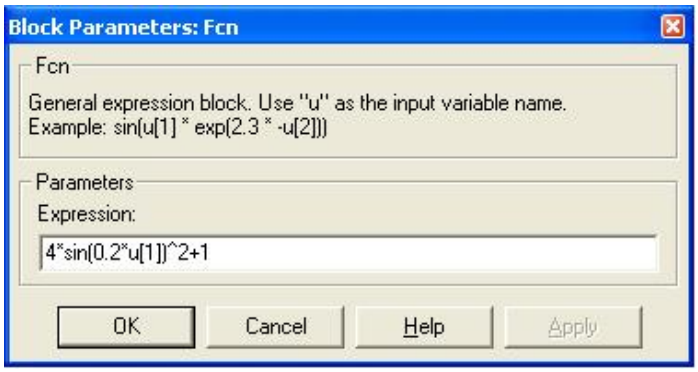

Figure 5.6: Configuration of the `Fcn` block for the implementation of multiple inputs and PID control

When you have completed the lab, make sure you save your files in the folder you created in Lab 1.

5.4 Deliverables

Prepare a brief write up describing what you learned from this lab. This does not need to be a formal report, but all material should be presented in a clear and logical manner, with concise descriptions where necessary. Include the following / answer the following questions:

1. Include tabulated data of the response characteristics from steps 2, 4, and 5. Be sure that you are using your "good" P value (i.e. NOT the value that makes the system oscillate about 10 rad / s) when collecting data for I and D tests.

2. Comment on how varying P, I, and D values impact response characteristics (i.e. rise time, settling time, etc. . .)

3. Include the plot of the output that oscillates consistently about the reference trajectory of 10 radians generated in step 3. Why do higher P values increase the oscillation of the output? What is happening to the location of the closed loop poles?

4. Include the plot of the output when using only integral control. Using the transfer function, determine the roots of the characteristic polynomial. What does this tell you about the behaviour of a system using only integral control?

5. Include a plot of the output using your PID controller when the desired angle is generated by the multi-input switch. Be sure to specify your final P, I, and D values.

Lab 6
Performance Specifications

The purpose of this lab is to give you an understanding of the performance of a second-order system. In particular, you will be examining the effects that system parameters have on various features of the output of that system. In the second part of the lab, you will examine the disturbance type of several systems.

6.1 Prelab

Before you go into the lab, you should read the following:

- Sections 8.2.2, 8.2.3, and 8.3.1 in the course notes on performance specifications.
- Be sure to familiarize yourself with the concept of system types.

Then, determine the state space representation $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}^t, \boldsymbol{D})$ of the generic second order model:

$$\ddot{y} + 2\zeta\omega_0\dot{y} + \omega_0^2 y = \omega_0^2 u.$$

Next, solve the above differential equation with a step response ($u = 1$), and comment on the effect of ζ and ω_0, and how this relates to the location of the poles of your transfer function.

6.2 Key Concepts

1. The main idea behind this lab is to understand how the addition of a zero affects the response of a system. For your system, let's say you add a zero at the point α. What this will do to your governing equation is add a scaled derivative of the step response. The derivative of the step response (i.e. the impulse response) will be scaled by a factor of $\frac{\omega_0}{\alpha}$. Thus, for very large alpha, this term will have little impact on the systems response. However, if this zero exists very close to the imaginary axis, your system will have what is called undershoot.

2. Secondly, you must understand the definition of system types. Essentially, systems can be type 0, 1, 2, etc ... A systems "type" determines its ability to track the error on a given reference trajectory. For example, a system of type k can track a reference trajectory with a bounded error for polynomials up to degree k. See Proposition 8.11 from the course notes for further clarification.

6.3 Procedure

6.3.1 Simulation of a second-order system

The model we will be examining is the generic second-order equation, where the output of the system is position:

$$\ddot{y} + 2\zeta\omega_0\dot{y} + \omega_0^2 y = \omega_0^2 u. \tag{6.1}$$

Since the physical constants of our friendly motor system cannot be changed in the simple open-loop configuration, we are confined to the world of simulations. We will examine the behaviour of the second-order system in Simulink.

1. Find the poles of your generic second-order system (or eigenvalues of your $\boldsymbol{A}$ matrix).

2. Open Matlab and build a Simulink model according to Figure 6.1. You should have

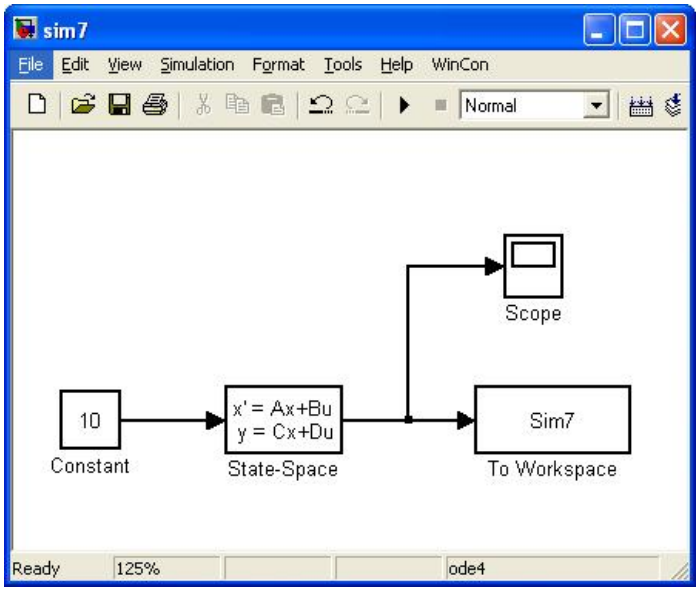

Figure 6.1: Simulink model for a generic second-order system

   determined the values for $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}^t, \boldsymbol{D})$ in your prelab. Define these matrices, in terms of ζ and ω_0, in a script and simply call them in your model. We will use this model in Section 6.3.2.

3. Run the system in simulation with initial values of $\zeta = 0.5$ and $\omega_0 = 1$. As this is a simulation, you do not need to build your system.

4. Using the `step()` function in Matlab (and a `for` loop), plot several values of ζ on the same graph using your $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}^t, \boldsymbol{D})$ matrices. Print out the graph and write down your observations related to the response characteristics. It may be helpful to use different style of lines for various values of ζ. Details on plotting with various line styles are discussed in Appendix A.

5. Repeat the previous step with various values of ω_0.

6.3.2 Non-minimum phase systems

The system described in Section 6.3.1 has the transfer function

$$T(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}.$$

We add a zero to the system as follows:

$$T(s) = \frac{\hat{y}(s)}{\hat{u}(s)} = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}\frac{(s+\alpha)}{\alpha}.$$

We use the parameter α to vary the position of the zero. In particular, we can use it to place our zero in $\mathbb{R}_{<0}$ or $\mathbb{R}_{>0}$. Notice that we have α in the denominator as well. This is a done to normalize the system at $s = 0$.

To convert this back into the time domain, we simply cross-multiply the transfer function to get the state equation

$$\ddot{y} + 2\zeta\omega_0\dot{y} + \omega_0^2 y = \frac{\omega_0^2}{\alpha}\dot{u} + \omega_0^2 u.$$

You can see that the only difference is that we now have a term involving $\dot{u}$. Since u is a step function, $\dot{u}$ is going to be an impulse function.[1] Implementing an impulse function cannot be done directly, but we can side-step the problem by cooking the initial conditions so that our initial value problem is a solution to our system when given a step input. It turns out that these initial conditions are $y(0) = 0$, and $\dot{y}(0) = \frac{\omega_0^2}{\alpha}$.[2]

1. Modify the model from the previous section to incorporate the zero added to the system by adding the initial conditions. The way to enter initial conditions into the model is shown in Lab 9.

2. For $\alpha = \{0.1, 1, 10\}$, what is the effect of increasing α on the rise time, settling time, overshoot, and peak time? Using Matlab, plot the output of these positive α values and print the graph. On the graph, write down your discoveries.

3. For $\alpha = \{-0.1, -1, -10\}$, what is the effect of increasing the magnitude of α on the rise time, settling time, overshoot, and peak time? Using Matlab, plot the output of these negative α values and print the graph. On the graph, write down your discoveries. What is the most noticeable characteristic of a non-minimum phase system?

4. Lastly, we will be observing exactly what the addition of a zero does to the system.

   (a) In Matlab, plot the step response and impulse response of your original system (without the zero) shown in equation 6.1 using the `step` and `impulse` commands.

[1] The derivative here is understood in the sense of distributions.

[2] This is a little involved, but see Section 3.6.5 of the course text for details.

(b) On a separate graph, plot the addition of the step response with the impulse response scaled by a factor of $\frac{\omega_0^2}{\alpha}$ (use the same α values from above). Note: only the impulse response should be scaled. When using the `step` and `impulse` commands, you must ensure that the arrays have the same length in order to superimpose them. This may involve truncating one array to match the length of the other.

(c) Comment on your plots obtained in part 4b with those obtained in step 2 and 3. Hint: The plots should be the exact same. Comment on why that is.

6.3.3 System type

We will now examine three different motor system and the system type of each one. The basic configuration is shown in Figure 6.2. This is an interconnected system, but we will

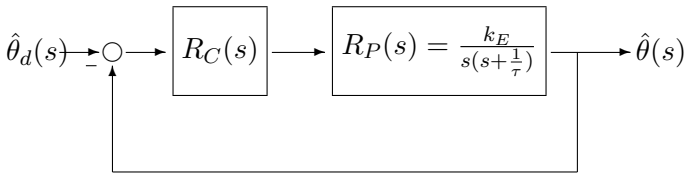

Figure 6.2: Feedback system with disturbance

take a "black box" approach, applying the disturbance at the input node, and measuring at the output node. Recall that the motor is modelled by transfer function $\frac{k_E}{s(s+\frac{1}{\tau})}$ when the output is the motor angle θ.

1. System 1 has the controller transfer function $R_C(s) = 5$. Verify by hand that the transfer function
$$T(s) = \frac{R_C(s)R_P(s)}{1 + R_C(s)R_P(s)}$$
is BIBO stable. Hint: What is $R_C = 5$ equivalent to? What do we know about this from lab 3?

2. Determine the system type for this particular system.

3. Assemble (or modify an existing) a Simulink model according to Figure 6.3.

4. Run the system using the constant input. What is the steady-state error of the system? Plot the error response in Matlab and print a copy of the plot with appropriate titles and axis labels. Is this consistent with your earlier work?

5. Repeat the above step, but use a linear and quadratic term (divide the quadratic term by 2 to ensure the encoder does not go out of range) for the reference signal. Plot all the error responses and make print outs. Make sure you have proper axis labels and titles.

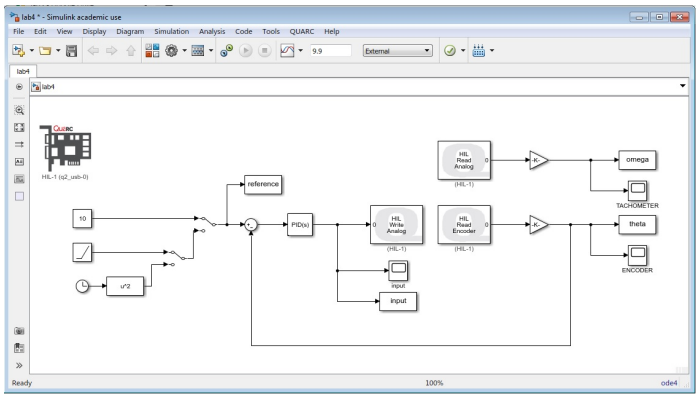

Figure 6.3: Simulink model for a DC Servo Motor system

6. System 2 has the same plant transfer function and a new controller transfer function given by $R_C(s) = \frac{1}{s}$. Is this system BIBO stable? If so, what is the system type?

7. Make changes to the system to implement the new controller and run the system with a constant, a linear, and a quadratic reference signal. Are the results consistent with your answer from above?

8. System 3 has the same plant transfer function and a new controller transfer function given by $R_C(s) = s$. Is this system BIBO stable? If so, what is the system type?

9. Make changes to the system to implement the new controller and run the system with a constant, a linear, and a quadratic reference signal. Are the results consistent with your answer from above?

10. Plot all the error responses in Matlab graph and print them. Make sure you have proper axis labels and titles.

When you have completed the lab, make sure you save your files in the folder you created in Lab 1.

6.4 Deliverables

Prepare a brief write up describing what you learned from this lab. This does not need to be a formal report, but all material should be presented in a clear and logical manner, with concise descriptions where necessary. Include the following/Answer the following questions:

1. Include plots of the systems response with varying ζ and ω_0 values, comment on the effect off varying these constants.

2. For both positive and negative α values, what is the effect of increasing the magnitude of α on the rise time, settling time, overshoot/undershoot, and peak time? Include plots.

3. In step 4 of the non-minimum phase system section, explain why your two plots are the same.

4. Create the following table for the system type section. Be sure to include and reference all necessary plots for the justification section.

System No.	Response to Constant Input	Response to Ramp Input	Response to Quadratic Input	Type?	Justification
System 1					
System 2					
System 3					

Lab 7
The Nyquist criterion

In this lab, you will use the Nyquist contour to determine if a system is IBIBO stable. The Nyquist plot is a conformal mapping of a closed loop in complex plane and it is a tool for the understanding of the stability of a closed-looped system. Furthermore, you will be examining the relationship between the Nyquist contour and the Bode plot, thus familiarizing yourself with such terms as *crossover frequency*, *gain margin*, and *phase margin*.

7.1 Prelab

Before you go into the lab, you should read the following:

- Sections 7.1.2 and 7.2 from the course text on the Nyquist criterion.

7.2 Procedure

7.2.1 Using the Bode plot to sketch the Nyquist contour

For this part of the lab, we will be using the motor in the unity gain feedback configuration shown in Figure 7.1. This setup should be quite familiar to you by now, so you should

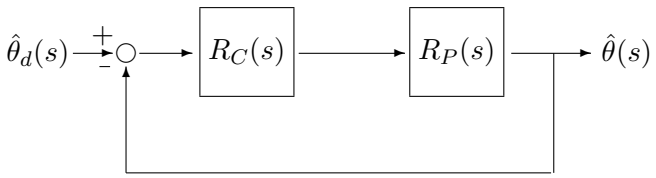

Figure 7.1: Feedback system with disturbance

have a well-developed intuition concerning the IBIBO stability of such a system. We will be taking the output of the system to be the motor velocity, ω, so our plant transfer function is $R_P(s) = \frac{k_E}{s+\frac{1}{\tau}}$. In Lab 2 you constructed the Bode plot for this system experimentally.

1. Using the Bode plot from Lab 2, determine the gain and phase crossover frequencies, and the phase and gain margins at these frequencies; see Section 7.2.2 from the course text.

2. Using the Bode plot determine the general transfer function, from the transfer function sketch the corresponding Nyquist contour. Looking also at the transfer function, determine whether or not the system is IBIBO stable.

3. Start up Matlab and define `t` to be the transfer function of the open-loop motor scheme. Essentially, we are just setting $R_C(s) = 1$ to see how motor behaves on its own.

4. To plot the Nyquist contour, use the `Nyquist` command in Matlab.

5. Is the Nyquist contour consistent with what you know about the IBIBO stability of the system? Comment on both the poles of the transfer function, and the zeroes of the characteristic polynomial.

7.2.2 Using a proportional controller

We will now turn our attention to the task of using the Nyquist contour as a tool in determining what range of values of a proportional controller can take on in order to make the system IBIBO stable.

The plant transfer function $R_C(s) = K$ will be used in the feedback configuration depicted in Figure 7.1. The transfer function for this system is simply

$$T(s) = \frac{KR_P(s)}{1 + KR_P(s)}.$$

What is important to note is that the condition $KR_P(s) = -1$ is equivalent to the condition $R_P = -\frac{1}{K}$. This seems obvious in itself, but what this allows you to do is to use the Nyquist contour for $R_P(s)$ on its own (i.e., with $K = 1$), and determine where you would like the point $\frac{1}{K}$ to be. This allows you to get a lot of mileage out of only one plot. The alternative is to construct the Nyquist contour for many values of K, and see if each contour has the required number of encirclements of the point $-1 + \mathrm{i}0$. This may not seem like a bad alternative if you have access to a computer, but the method of normalizing the proportional gain is much more efficient.

1. In Matlab, define the plant transfer function to be $R_P = \frac{s+1}{s(\frac{s}{10}-1)}$.

2. Produce both the Bode plot and the Nyquist contour. Take a minute to further reaffirm your faith in the relationship between the two.

3. What are the crossover frequencies, and what are the gain and phase margins at these frequencies?

4. Since our plant transfer function has one pole in the right-half complex plane, we require one counterclockwise encirclements of the point $\frac{1}{K}$. Using this information, in which region should we place the point $-\frac{1}{K}$?

5. What values of K satisfy the above requirement?

6. Repeat the above procedure for the plant transfer function $R_P(s) = \frac{1}{s(s+1)^2}$. Remember, Nyquist plots always close clockwise.

7.2.3 The phase margin

In this part of the lab we will quickly demonstrate the phase margin as it applies to the Nyquist contour.

1. We will use the plant transfer function $R_P(s) = \frac{1}{s(s+1)^2}$. Produce the Bode plot and the Nyquist contour of this system.

2. Determine how much you have to boost the gain (in decibels) to provide a phase margin of 0° at a gain cross over frequency of $10^n \frac{\text{rad}}{\text{sec}}$, $n \in \mathbb{Z}$. Will the resulting system, with this gain boosted by using a proportional controller, be IBIBO stable?

3. The number you get is a scalar that represents the factor by which you can "stretch" (or "shrink" as the case may be) the Nyquist contour towards the point $-1 + \text{i}0$ and still maintain IBIBO stability (assuming the system is stable for $K = 1$, which our system is). Looking at the Nyquist contour, does this scalar quantity seem to make sense? Is this consistent with your work from Section 7.2.1?

When you have completed the lab, make sure you save your files in the folder you created in Lab 1.

Lab 8
PID design using frequency response

In this lab, you will use the Bode plot you produced using the measurements for the motor to design a PID controller to meet specified performance criteria.

8.1 Prelab

Before you go into the lab, you should read the following:

- Section 12.2 from the course text on using frequency response method for controller design.

Since Simulink does not allow improper transfer functions, you must use the `PID` block for the controller in this lab. Find the relationships between the constants in equation (8.1) and (8.2):

$$\frac{K}{s}(1+T_D s)(s+\frac{1}{T_I}), \tag{8.1}$$

$$P+\frac{I}{s}+Ds. \tag{8.2}$$

8.2 Procedure

Before we implement the controller, we need to do some paper work to design it.

8.2.1 Design specifications

You are charged with the designing the controller rational function R_C in the feed back loop in Figure 8.1. The plant transfer function is that for the motor:

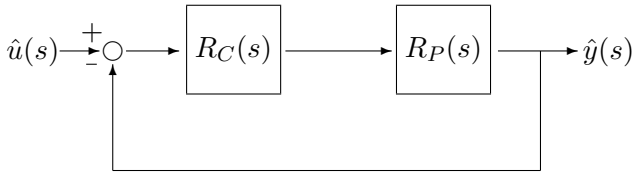

Figure 8.1: Feedback system with disturbance

$$R_P(s)=\frac{k_E}{s(s+\frac{1}{\tau})},$$

where you have measured the values for k_E and τ in Lab 1.

You are asked to design a controller so that the closed-loop system has a phase margin of 65° and a crossover frequency as large as possible. The controller you will use is a PID controller in the form

$$R_C(s) = \frac{K}{s}(1 + T_D s)(s + \frac{1}{T_I}) = K(1 + \frac{T_D}{T_I} + T_D s + \frac{1}{T_I s}).$$

8.2.2 The steps in controller design

We shall lead you by hand through the design process.

1. Produce the Bode plot for plant transfer function using Matlab.
2. In the PID controller design, let us decide to fix $T_I = 10T_D$. You are free to change this ratio during the lab if you find it unsuitable.
3. Also, to start the design process, let us fix the gain K to 1. You may want to use the `zpk` function in Matlab to define the transfer function.
4. With the above assumptions, is it true that there is a choice for T_D for which the closed loop system is IBIBO stable (IBIBO stability can be verified either by Nyquist Criterion or Hurwitz Criterion)? Is it true that, no matter what choice one makes for T_D, the closed-loop system will be IBIBO stable? Can you spot the features on your Bode and/or Nyquist plot which help you answer this question? How do these play a role in the design of the controller?
5. Based upon the considerations in the previous step, choose a value for T_D which meets the design objectives.
6. Now choose K to give the crossover frequency.
7. Make sure the closed-loop system is IBIBO stable using the Nyquist criterion.

You now have values for K, T_D, and T_I which you will carry into the rest of the procedure.

8.2.3 Executing the design in hardware/software

Now you will implement your controller in hardware.

1. Open Matlab and build a Simulink model according to Figure 8.2.
2. Use values for K, T_D, and T_I that make the closed loop system unstable in hardware. Use the above contraints you just calculated to determine values that cause instability. Be prepare to shut the system off! Produce plots of the unstable system.
3. Now, using an input step response of 1, use the "good" values for K, T_D, and T_I to determine the rise time, the percentage overshoot, and the ϵ-settling time for $\epsilon = 0.5$.

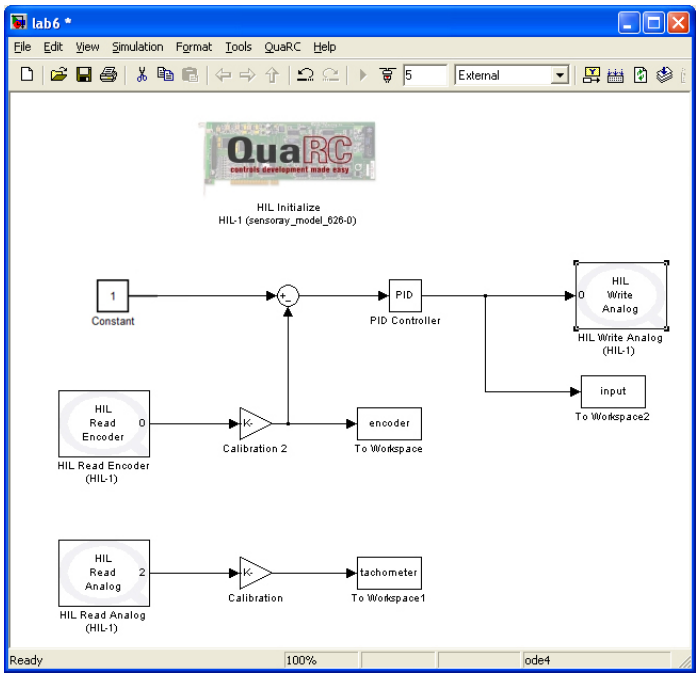

Figure 8.2: Simulink model for the implementation of PID control using frequency response method; step reference

4. Now run the system with your "good" values for K, T_D, and T_I and a sinusoidal input whose magnitude is 1 and whose frequency is the crossover frequency that was calculated above; see Figure 8.3 for the Simulink. What is the magnitude of the output compared to the input, and what is the phase of the output relative to the input?

5. Is your controller a good one? How might you improve it? What do your improvements mean in terms of loopshaping ideas?

6. Change the ratio of $\frac{T_I}{T_D}$ and discuss the effect on the performance of the controller. Does what you observe make sense? Generate plots to demonstrate your observations.

When you have completed the lab, make sure you save your files in the folder you created in Lab 1.

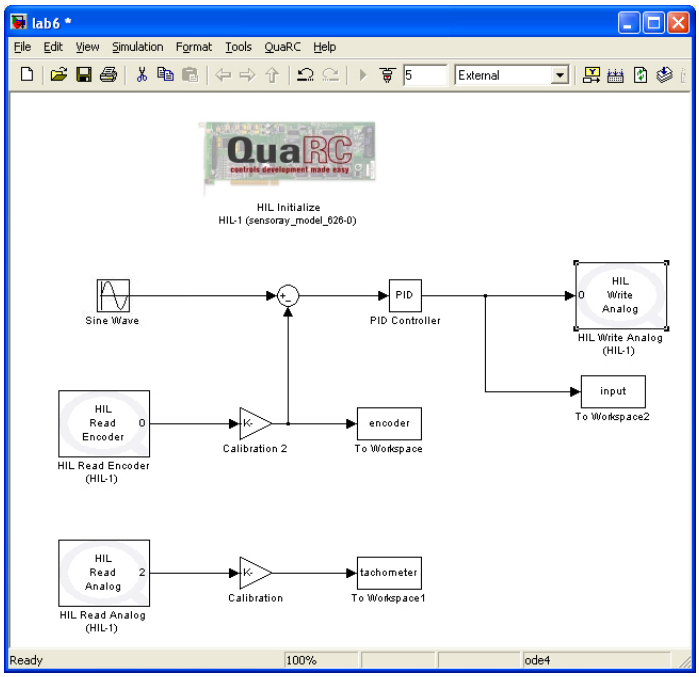

Figure 8.3: Simulink model for the implementation of PID control using frequency response method; sinusoidal reference

Lab 9
Controllability and observability

This lab will demonstrate the fundamental ideas behind the controllability and observability properties of a system. You will analyze a simple circuit and determine the conditions for observability and controllability. You will then proceed to simulate the system using Simulink under various conditions.

9.1 Prelab

You may wish to read Sections 2.3.1 and 2.3.2 from the course notes to recall some background on controllability and observability.

Using Figure 9.1 as a reference, define x_1 as the voltage across the capacitor, and x_2 as

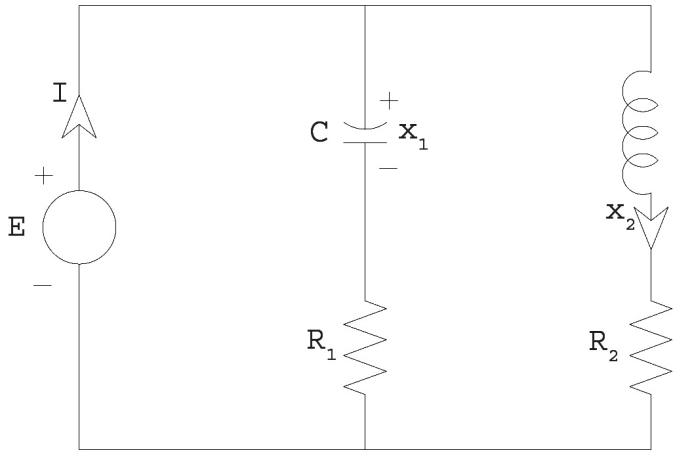

Figure 9.1: State space configuration

the current through the inductor. Take the output to be the current entering the circuit, denoted I in Figure 9.1.

1. Write out the system equations in the state-space form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u,$$
$$y = \boldsymbol{c}^t + \boldsymbol{D}u.$$

   If you are having trouble getting the differential equations, or just not "electrically inclined", you should review past course notes on electrical circuits and differential equations. Suck it up, Apple Mech students!

- When finding the differential equations for a system, your goal should be to determine an expression for the derivative of each state variable in terms of the state variable(s) and the forcing function. Once you have these, you can determine $\boldsymbol{A}$ and $\boldsymbol{b}$.
- The following facts may be useful. The current through a capacitor is given by $I_c = C\frac{dV_C}{dt}$, the voltage across an inductor is given by $V_L = L\frac{dI_L}{dt}$, and the voltage across a resistor is given by Ohm's law, $V_R = I_R R$. By Kirchoff's voltage law, the voltage across each branch of the circuit is simply E. You can use this fact to get expressions for $\boldsymbol{A}$ and $\boldsymbol{b}$.
- Since the output is the current entering the circuit, and you will by this point have expressions available for the current in each branch, you can use Kirchoff's current law (i.e., conservation of current) to determine expressions for $\boldsymbol{c}$ and $\boldsymbol{D}$.

2. Calculate the controllability matrix $\boldsymbol{C}(\boldsymbol{A}, \boldsymbol{b})$.
3. Calculate the observability matrix $\boldsymbol{O}(\boldsymbol{A}, \boldsymbol{c})$.
4. Determine the conditions under which the system is uncontrollable. Recall that a square matrix has full rank if and only if its determinant is non-zero.
5. Determine the conditions under which the system is unobservable.
6. When the system is uncontrollable, determine the set of reachable points for zero initial conditions. This is going to be a one-dimensional vector space, so there is a simple relationship between x_1 and x_2. Determine this relationship when the system is uncontrollable.
7. When the system is unobservable, determine the set of initial conditions that yield the same output, and the linear relationship between the initial conditions.

9.2 Procedure

Via simulation, you will determine the conditions under which the circuit in Figure 9.1 is controllable and/or observable.

9.2.1 Controllability

When analyzing the controllability, you will be examining the behaviour of the system states. Recall that a rough definition of controllability is: "starting from the origin, you can reach any point in state space by applying an appropriate input."

1. Build the Simulink model as shown in Figure 9.2. The model applies a constant voltage to the dynamic model defined by $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{D}$ defined previously. The output is the current in the circuit.

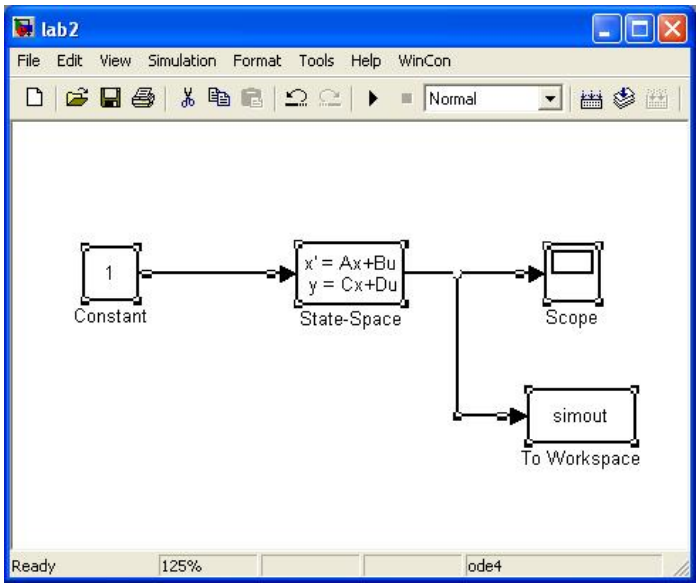

Figure 9.2: Simulink model for Lab 9

2. Define the matrices $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{D}$ by double-clicking on the `State-space` block. Each matrix is entered using the following convention. The following is an example on how to properly input values into the `State-Space` block. The values shown are *not* the proper values. Use values that correspond to the matrices in the prelab. The matrix

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix}$$

is entered by typing `[0,1;-1,-2]` in the line reserved for $\boldsymbol{A}$. Note that elements of a row are delimited by comma (or spaces) and each row is delimited by a semicolon.

The entries of Figure 9.3 correspond to the dynamical system

$$\begin{aligned} \dot{\boldsymbol{x}} &= \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \boldsymbol{x} + [0] u. \end{aligned}$$

Note that the last line specifies the initial conditions for the states. In this case, we have set $x_1(0) = 0$ and $x_2(0) = 0$. Come up with an appropriate variable to show the linear relationship between x_1 and x_2 when the system is uncontrollable (make sure you record this in your lab report). Define this variable as one of your outputs in your model. Note that you can add as many outputs as you want just by adding rows to the matrix $\boldsymbol{c}$.

Define the initial conditions to be zero. The default final time is set to 10 by Simulink. You may change it by opening the window

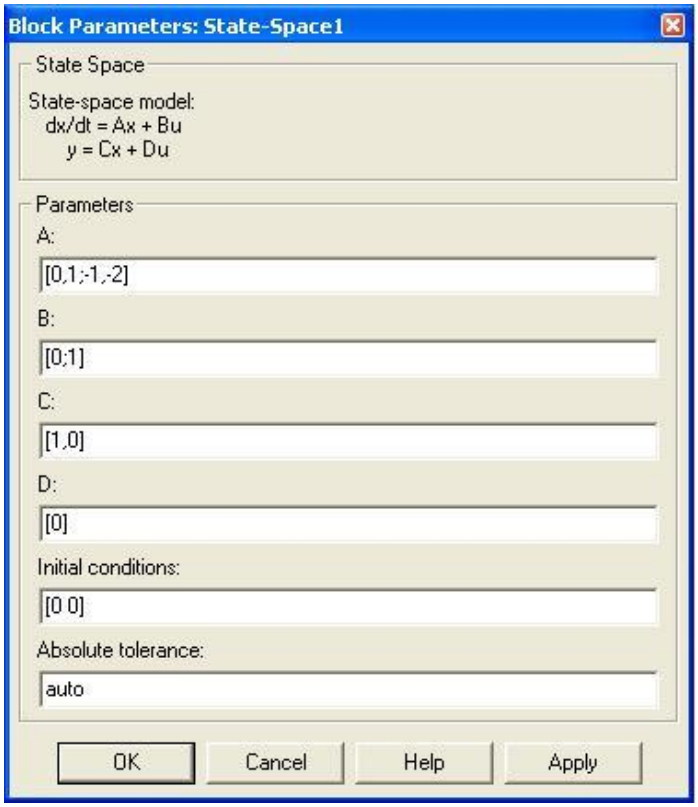

Figure 9.3: State space configuration

`Simulation→Configuration Parameters`

and entering the required time in the `Stop Time` box.

3. Run the Simulink block under uncontrollable conditions. This will depend on your choice of R_1, R_2, C, and L. Recuperate the state variable values written to the workspace and plot x_1, x_2, and the output variable you defined. Does the output variable you chose show that there is, in fact, a linear relationship between x_1 and x_2? Under these conditions, is it possible to find an input voltage E that will allow you to reach a point that is off this line?

4. Add an appropriate title to your graph and print it.

5. Rerun the system, but this time use conditions that make the system controllable. Plot and print a graph of x_1 and x_2 and your output variable. What can you now say about your output variable? What is the set of reachable points under these conditions?

6. Again, give an appropriate title to your graph and print it.

9.2.2 Observability

When analyzing observability, you will be examining the output behaviour of the system. Recall that a rough definition of observability is: "a change in initial conditions and/or input results in a change in the output."

1. Using the work from your prelab, enter the value of $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{D}$ into the `State-Space` block. Change the name of the output variable from `simout` to `I`.

2. We will want to examine the output using many different initial conditions. Build and run the system using a pair of initial conditions that are in the kernel of the observability matrix. Plot, and print a graph of the output variable I. Make sure that you include values of the constants in the title of the plot.

3. Rerun the system using a different pair of initial conditions that are also in the kernel of the observability matrix. Include a plot of the output. Does the result of this experiment confirm your work from the prelab? What happens if you use initial conditions that are not in the kernel of the observability matrix?

When you have completed the lab, make sure you move the files created in the directory created in Lab 1.

Appendix A
Matlab

The students of this course should be familiar with the basic ideas of computer programming and Matlab from first year courses.

A.1 Defining variables

- `x=3`
  Defines the variable `x` to be the constant 3.

- `x=(1,2;3,4;5,6)`
  Defines the variable `x` to be the 3×2 matrix

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

  Elements of a row are delimited by commas (or spaces) and each row is delimited by a semicolon.

A.2 General Commands

- `dir`
  Displays a list of files in the current directory.

- `open lab_1.mdl`
  Opens the specific file in the argument. We will be dealing mostly with `*.mdl` and `*.m` files.

- `who`
  Displays a list of variables in the memory of Matlab.

- `simulink`
  Opens the `Simulink Browser Library` window. Using the GUI control, you can drag and drop the blocks to build the Simulink models needed for each lab. Details on using Simulink and WinCon are discussed in Appendix B.

A.3 Plotting

- `plot (lab_1_Tachometer)`
  Plots the data from the variable `lab_1_Tachometer` in Matlab memory. You can check the list of variables in the Matlab memory by using the `who` command.

- `plot (lab_1_Tachometer,'r:')`
  Plots the result in the memory of Matlab and specifies the colour of the graph to be red and the line style to be dotted. Colour and line format are optional commands and they do not have to be specified for the plot command to produce an output. You can also specify one style parameter without the other. The default colour is blue and the default line style is solid. Table A.1 is a list of colours and line styles that can be used with the plot command.

Line style/colour	Matlab command
solid	'-'
dashed	'--'
dotted	':'
dash-dot	'-.'
blue	'b' or 'blue'
black	'k' or 'black'
cyan	'c' or 'cyan'
green	'g' or 'green'
magenta	'm' or 'magenta'
red	'r' or 'red'
white	'w' or 'white'
yellow	'y' or 'yellow'

Table A.1: Colour commands in Matlab

- `title ('Angular Velocity of the Motor')`
  Sets the title of the plot to the text in quotations.

- `xlabel ('Time (s)')`
  Sets the x-axis label of the plot to the text in quotations.

- `ylabel ('rad/sec')`
  Sets the y-axis label of the plot to the text in quotations.

- `hold`
  Hold the current graph in figure and allow the user to plot more than one set of data on the same figure.

- `hold off`
  Release the current graph in figure and allow a new plot to replace the current graph.

A.4 Control System Toolbox

- `sys = ss(A,b,c,D)`
  Defines the the state-space system from matrices $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{D}$. For a model with n states and 1 output,
  - $\boldsymbol{A}$ is an $n \times n$ matrix,
  - $\boldsymbol{b}$ is an $n \times 1$ matrix,
  - $\boldsymbol{c}$ is a $1 \times n$ matrix ($\boldsymbol{c}^t$ in our notation), and
  - $\boldsymbol{D} = [0]$ (always true for this class).
- `h = tf([1 0],[1 2 10])`
  Defines the variable `h` to be the transfer function
  $$\frac{s}{s^2 + 2s + 10}.$$
- `h = zpk([1 0],[-1 -2 -10],[3])`
  Defines the variable `h` to be the transfer function using the location of the zeros, poles, and a multiplicative constant:
  $$\frac{3s(s-1)}{(s+1)(s+2)(s+10)}.$$
- `h = tf(sys)`
  Defines the variable `h` to be the transfer function for a given state-space system.
- `sys = tf2ss[tf]`
  Gives the SISO linear system corresponding to the transfer function `tf`.
- `bode(sys)`
  Produces the Bode plots for the given system.
- `impulse(sys)`
  Produces the impulse response for the given system.
- `nyquist(sys)`
  Produces the Nyquist plot for the given system.
- `margin(sys)`
  Produces the gain and phase margins with associated crossover frequencies.

Appendix B
Simulink

Simulink allows simulation of complex control systems using a drag and drop block diagram interface. Simulink is especially useful when used in conjunction with the Real-Time Workshop which allows Simulink diagrams to be converted into C codes which can be run in real-time on a number of so-called targets (the PC being one such target).

B.1 Starting

- To use Simulink, one must first start Matlab. After starting Matlab, you would type `simulink` in the Matlab command prompt to get the `Simulink Library Browser` window.
- Selecting `File`→`New`→`Model` (or `Ctrl+N`) while in the `Simulink Library Browser` will give you a blank model window into which you can drag-and-drop system blocks from the `Simulink Library Browser` to build a Simulink model. These models can be saved as `*.mdl` files for future simulations and editing.

B.2 Building a Simulink model

- To connect the output of block A to the input of block B, simply left-click the output port of block A and drag the line that would appear into the input port of the block B.
- In order to connect the output of a block into the inputs of multiple blocks at the same time, you can right-click on an existing connection to get another line and drag that connection into the input of another block.
- When a block is dragged into the model window, it will be given a generic name. For example, when the `scope` block is dragged into a model, it would simply be labelled as "scope" and if it was the second `scope` block to be dragged into the model, it would be labelled as "scope2". It is a good practice to rename these blocks and give them more appropriate labels. These names are usually suggested in the diagram of the models in each lab. To rename a block, simply click on the existing name once and edit.

B.3 Simulations

- One could view and edit the simulation parameters by clicking on

`Simulation→Simulation Parameters`

(or `Ctrl+E`) while editing the model. For the purpose of this course, there are only three things that you have to worry about:

1. Start and stop time: The default start time is 0 and the default stop time is 10. Usually, there is no reason to change the default start time, but you might find it useful to extend the stop time so that you could observe the simulation for a longer period of time.
2. Solver Method: You must have this set to a fixed-step when implementing real-time controllers. The default solver is a discrete method. You need to change it from the default method to `ode4`, which is an implementation of the Runge-Kutta method.
3. Step size: The default setting of 0.001 corresponds to 1000 Hz sampling frequency. This is the fastest rate at which the system can sample.

B.4 Plotting

- The outputs of a simulation can be captured by using the `To Workspace` block. The data would be recorded as a variable in memory of Matlab. The default name of the output is `simout`, but you should change the name of this output just as you would give appropriate label to the block itself. One could plot the simulation output by using plot command discussed in Appendix A.

B.4.1 Saving data via the "To Workspace" block

This is the preferred method of saving data to your Matlab workspace. The `To Workspace` block can be found at

`Simulink→Sinks`

Drag this block into your workspace and connect it to the variable you wish to save. Double click on the block to configure it. Choose a good variable name and in the `save format` drop down menu select `Structure with time`. After the simulation the data will be automatically saved to your Matlab workspace and you can plot it with the command

```
plot(varname.time,varname.signals.values)
```

replacing "`varname`" with the variable name you chose when configuring the `To Workspace` block.

B.4.2 Saving data from a scope

You can also save scope data, but the scope seems to have short-term memory loss which makes it one of the most useless blocks in the simulink library. Nevertheless if you need to save the data from a scope, follow these instructions.

1. Double click on the scope you wish to save the data from.
2. Click on the Parameters Icon (in the top left of the scope dialog box).
3. Data History Tab
   - Uncheck box `Limit data points`
   - Check box `Save data to workspace`
   - Choose a variable name
   - Select `Array` from the Format drop down menu.
4. Click `Apply` and `OK` to exit.

After the simulation has been performed, the data will appear as an array in your Matlab workspace. You can plot the data with the command

```
plot(ScopeData1(:,1), ScopeData1(:,2))
```

where `ScopeData1` is the variable name that you set in the above steps.

Appendix C
Lab Equipment

In the lab, there are several computers equipped with data acquisition systems running Windows 7 with Matlab. The hardware equipment and some software tools are manufactured by Quanser Consulting, a Canadian company developing real-time control systems for education and research. This document introduces some of the hardware equipment to be used in the labs. Familiarity with this document is needed to perform the labs.

C.1 Hardware devices

The lab hardware consists of three components:

1. data acquisition system;
2. power module;
3. servomotors.

C.2 Data Acquisition Board

In order for the computer to run a controller, analog-to-digital (A/D) and digital-to-analog (D/A) conversions are necessary. These are done using the data acquisition and control board (DACB), which inputs the measured signal(s) to the computer and outputs control action to the actuator in the control loop. The DACB in this lab consists of a single board: the Q2-USB, which is made by Quanser Consulting. Figure C.1 shows a photo of the Q2-USB card. This data acquisition board is an external board connected to the computer through a USB port.

Figure C.1 also shows the proper configuration of the data acquisition board. The Encoder is the 5 pin Din and is plugged in to channel 0 in the Encoders portion of the board. The tachometer (S3 on the Quanser) is the red RCA plug and is plugged into channel 0 is the ADC portion of the board. Finally the analog output is the solo black RCA plug and is plugged in to channel 0 of the DAC portion of the board.

C.3 Universal power module

The universal power module (UPM-2405), which is shown in Figure C.2, is a linear power operations amplifier. The Q2-USB data acquisition board cannot deliver enough power to the actuators used in this lab; therefore, a signal buffer is needed. The UPM-2405 is used

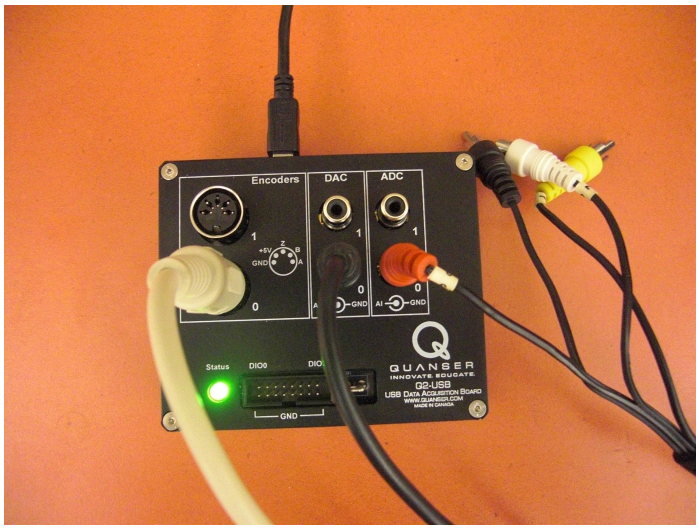

Figure C.1: Q2-USB data acquisition board

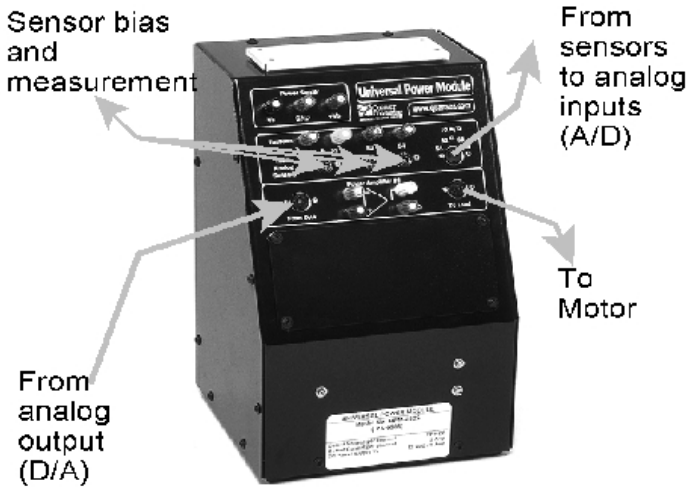

Figure C.2: Universal power module

as our signal buffer since it can deliver up to 5A to an actuator in a non-inverting, unity gain configuration.

The following connections can be made to/from the UPM (see the labels on the UPM).

- From analog sensors: there are four (S1-S4 inputs which can be connected from analog sensors (and then subsequently to the computer); the cable used is a 6-pin mini-din/6-pin mini-din cable (light tan colour), which is now referred to as the analog sensor cable.
- To A/D: the four analog sensor signals (S1-S4) can then be connected to the Q2-USB terminal board for A/D conversion into the computer; the cable used is a 5-pin din-stereo/4RCA cable (black colour), which is now referred to as the A/D cables.
- From D/A:/@ this is where you input the D/A signal from the Q2-USB terminal board to the UPM; the cable used is a 5-pin din-mono/RCA cable (black colour), which is now referred to as the D/A cable.
- To load: here you connect the amplified D/A signal to an actuator (e.g., servomotor); the cable used is a 7-pin din/4-pin din cable (black colour), which is now referred to as the load cable. Note there are two types of load cables one with unity gain and another with a cable gain of 5. Make sure you use the right one.
- Others: A few other connections are possible for convenience: e.g., a DC power supply on the top left provides 12 volts; the signal s from analog sensors S1-S4 can be easily monitored by connecting to a scope to the banana plug terminals.

C.4 DC servomotor

The Quanser DC servomotor (SRV02) is shown in Figure C.3. A 3W motor is mounted in a solid aluminium frame and drives a built-in Swiss-made 14.1:1 gearbox whose output drives an external gear, which is attached to an independent output shaft that rotates in an aluminium ball-bearing block. The output shaft is equipped with an encoder. The external gear on the output shaft drives an anti-backlash gear connected to a precision potentiometer for measuring the output angle. The external gear ratio can be changed from 1:1 to 5:1 using different gears. Two inertial loads are supplied with the system in order to examine the effect of changing inertia on motor performance. Several connections are available for the servomotor. The input voltage connects to the UPM using the load cable. The potentiometer and tachometer ports connect to the UPM-2405 using sensor cables and are used to measure angular position and angular velocity respectively. Additionally, the shaft encoder port connects to the terminal board using an encoder cable and is used to measure angular position. The calibration factors listed in Table C.1 are needed in order to use the sensors in units of degrees or radians.

Figure C.3: DC servomotor (SRV02)

Connection	Conversion (Rad)	Conversion (Deg)
Encoder	$-\frac{2\pi}{4096}$	$-\frac{360}{4096}$
Tachometer	$\frac{100\pi}{63}\frac{\text{rad}}{\text{sec}}$	$\frac{18000}{63}\frac{\text{deg}}{\text{sec}}$
Potentiometer	$\frac{1}{4096}$	$\frac{180}{4096\pi}$

Table C.1: Conversion factors