MTHE 393 Lab Manual

May 16, 2023

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 were used jointly by the courses MTHE 393 and MTHE 332.

In the summer of 2023, this manual was updated again to reflect changes to the MTHE 393 curriculum once MTHE 332 was replaced by MTHE 335, which is now taken concurrently with MTHE 393. These updates were completed by Henry Wilson and Ben van Eeden.

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.

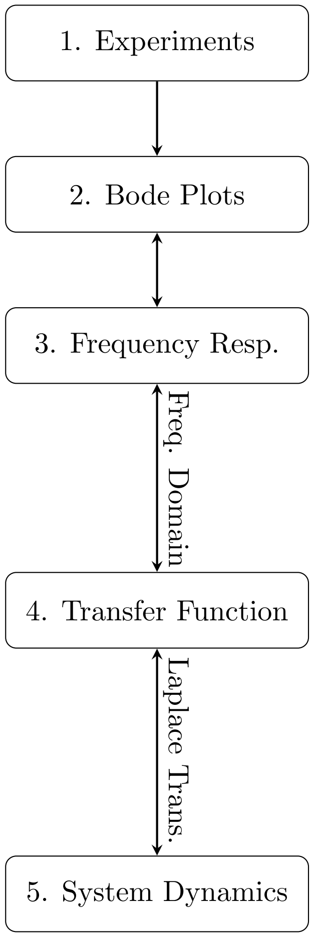


Figure 1: Project Outline

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 3 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.

Lab Safety

Safety and security is paramount when carrying out a lab. Please conduct yourself in a responsible manner at all times during these controls lab. In addition, please follow these safety instructions:

1. Follow all written and verbal instructions carefully. If you do not understand a part of a procedure, ask your TA for further instructions;
2. be alert and proceed with caution at all times in the laboratory;
3. as these labs involve electrical equipment (e.g., power amplifiers, etc.), participants should not stick any metal objects into electrical receptacles, and liquids should be kept out of the lab;
4. notify your TA immediately of any unsafe conditions you observe;
5. report any accident (e.g., breakage of equipment or cables, etc.) or injury (cut, etc.) to the TA;
6. equipment instructions must be read carefully before use;
7. know where the fire alarm and the exits are located;
8. work areas should be kept clean and tidy at all times;
9. do not eat food or drink beverages in the laboratory.

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:

$$\begin{tikzcd} u(t) \arrow[r] & \boxed{\left(\frac{d^2}{dt^2} + \frac{1}{\tau}\frac{d\theta}{dt}\right) = k_E u} \arrow[r] & \theta(t) \end{tikzcd}$$

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 4 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 a 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.
- Week 2 of the lab notes

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. Under the `New` header, go to `QUARC`→`Blank QUARC Model` on the open dialogue box to create a new Simulink model. **You must use the QUARC model, otherwise you will not be able to connect to the motor**.

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

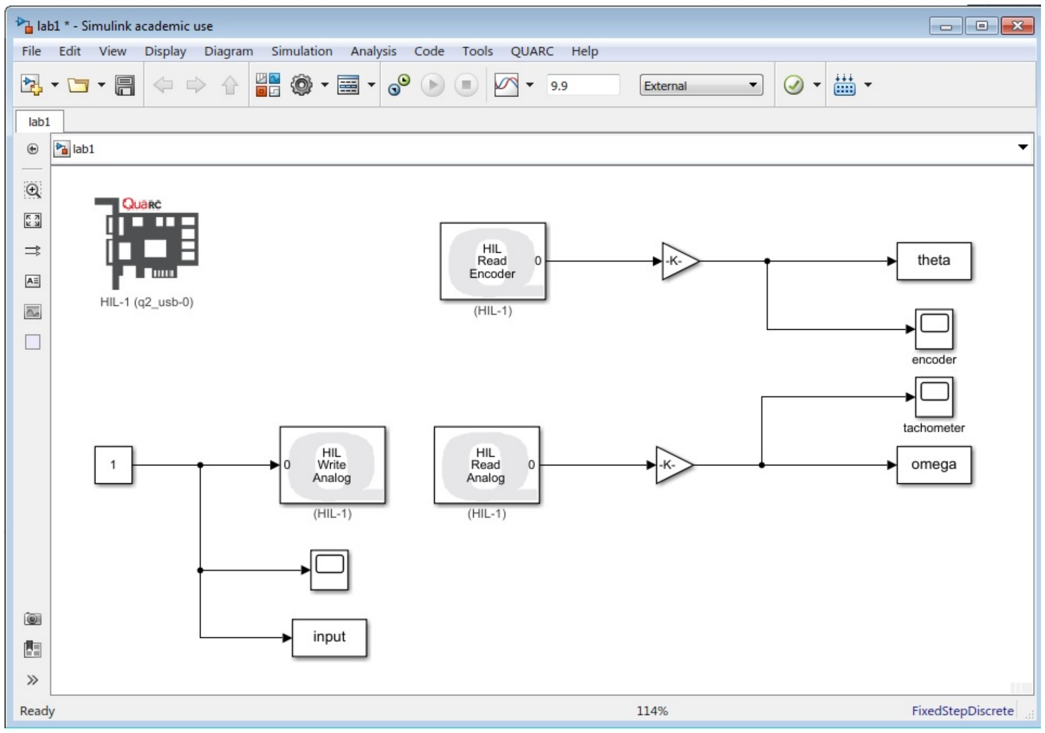


Figure 1.2: Simulink model for Lab 1

6. Make a new folder in your working directory folder 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. Under the “Simulation” heading, click on the library browser and search up the `HIL Initialize` block. This can be done by typing “HIL Initialize” in the Library Browser.

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`. Consult the instructions to see which blocks to use in each lab. You can find these blocks by searching the Library Browser. Add the appropriate HIL blocks to your model now.

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

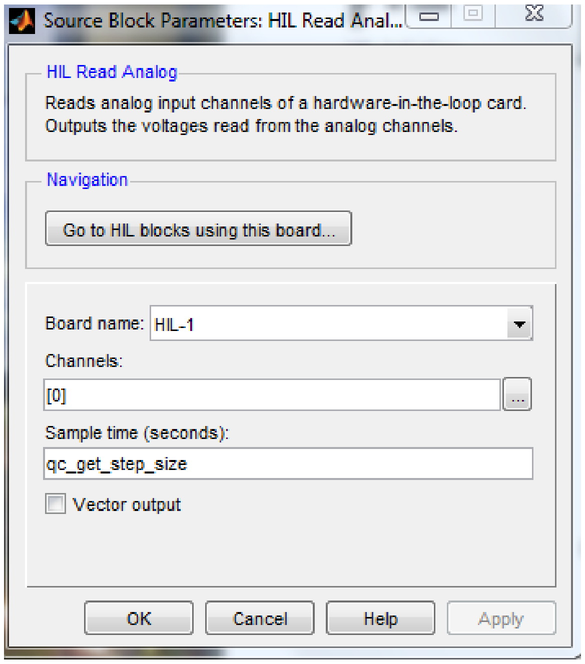
11. There are no `calibration`, `encoder`, or `tachometer` blocks. These blocks are simply "gain" or "scope" blocks which have been renamed. It is always best to refer to the block pictures instead of the block names.

    The blocks labelled as `theta`, `omega`, and `input` are just "To Workspace" blocks that have been renamed. Connect these blocks 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`.

    The `1` block is a "constant" function block.

Add in the rest of the blocks so that your model matches Figure 1.2.

12. 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.


Source Block Parameters: HIL Read Anal...

HIL Read Analog

Reads analog input channels of a hardware-in-the-loop card. Outputs the voltages read from the analog channels.

Navigation

Board name: HIL-1

Channels:

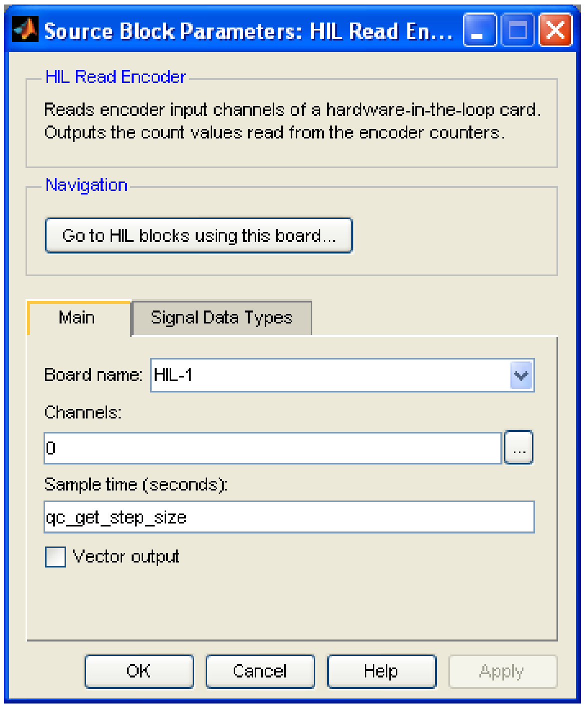
[0]

Sample time (seconds):

qc_get_step_size

Vector output

Figure 1.3: HIL `Read Analog` block settings


Source Block Parameters: HIL Read En...
HIL Read Encoder
Navigation
Go to HIL blocks using this board...
Main
Signal Data Types
OK
Cancel
Help
Apply

Reads encoder input channels of a hardware-in-the-loop card. Outputs the count values read from the encoder counters.

Board name: HIL-1

Channels:

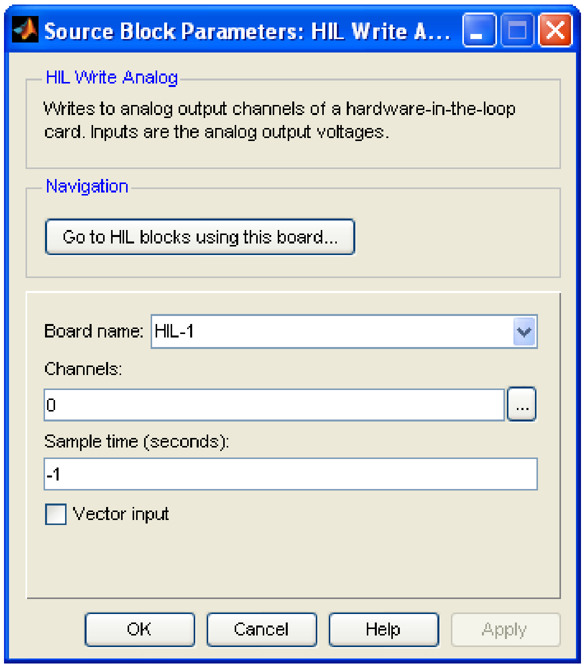
0

Sample time (seconds):

qc_get_step_size

☐ Vector output

Figure 1.4: HIL `Read Encoder` block settings


Source Block Parameters: HIL Write A...
HIL Write Analog
Navigation
Go to HIL blocks using this board...
OK
Cancel
Help
Apply

Writes to analog output channels of a hardware-in-the-loop card. Inputs are the analog output voltages.

Board name: HIL-1

Channels:

0

Sample time (seconds):

-1

☐ Vector input

Figure 1.5: HIL `Write Analog` block settings

13. 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.

14. Click on `Modeling` header →`Model Settings` from your Simulink screen (or `Ctrl+E`). Once you are in the menu, navigate to `Solver` → `Solver selection` and set the `type` setting to `fixed-step` and set the `solver` setting to `ode 4`.

15. In the `Hardware` toolbar, change "inf" 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.

16. 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.

17. Click `Monitor and Tune` from the `Hardware` menu to run the program. If any errors arise, refer to the Common Errors section below.

18. Data will automatically be saved from the `To Workspace` blocks to your Matlab workspace.

19. You can plot the data from the `To Workspace` blocks 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?

20. 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

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

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

21. 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 $t = \tau$.

22. 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.

23. 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.

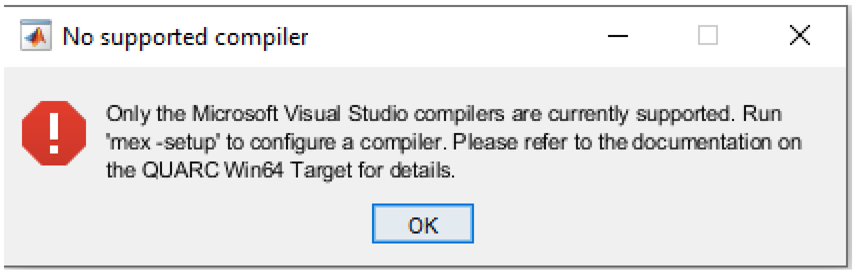
24. Save all files in the folder you created. Hand in all the plots you printed during this lab session and along with it the work to show how you have obtained the two constants. Please make sure the names and student numbers of all your group members are on the first page.

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 Common Errors

- If you encounter a compiler error when you click `Monitor and Tune` that mentions a Visual Studio compiler: simply type `mex -setup` in the Matlab terminal, then click


No supported compiler

Only the Microsoft Visual Studio compilers are currently supported. Run 'mex -setup' to configure a compiler. Please refer to the documentation on the QUARC Win64 Target for details.

  the `Microsoft Visual C++` option.

- A default setting may be changed which can result in only part of the data from Simulink being outputted to Matlab workspace. If you encounter this, please complete the following steps:

  In Simulink: Go to the `Hardware` tab → `Control Panel` (under Prepare) → `Signal & Triggering` → Increase the value in the "Duration" field - we've found that 10,000 works well.

  This increases the buffer size so that more data can be stored.

1.5 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
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 2.1, but we will be using a constant DC

$$\begin{tikzcd}
u(t) \arrow[r] & \boxed{\left(\frac{d^2\theta}{dt^2} + \frac{1}{\tau}\frac{d\theta}{dt}\right) = k_E u} \arrow[r] & \theta(t)
\end{tikzcd}$$

Figure 2.1: Open-loop motor schematic

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

2.1 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.

2.2 Procedure

2.2.1 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 $\boldsymbol{b}$. Refer to the lab notes for assistance with this.

2. Determine the output equation:

$$y = \boldsymbol{c}^t \boldsymbol{x} + \boldsymbol{D}u, \tag{2.1}$$

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 2.2.

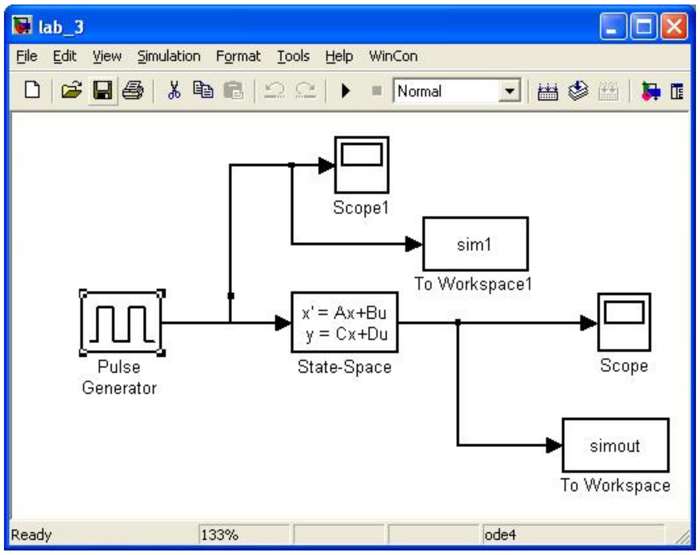


Figure 2.2: Simulink model for Lab 2

4. 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 2.3.

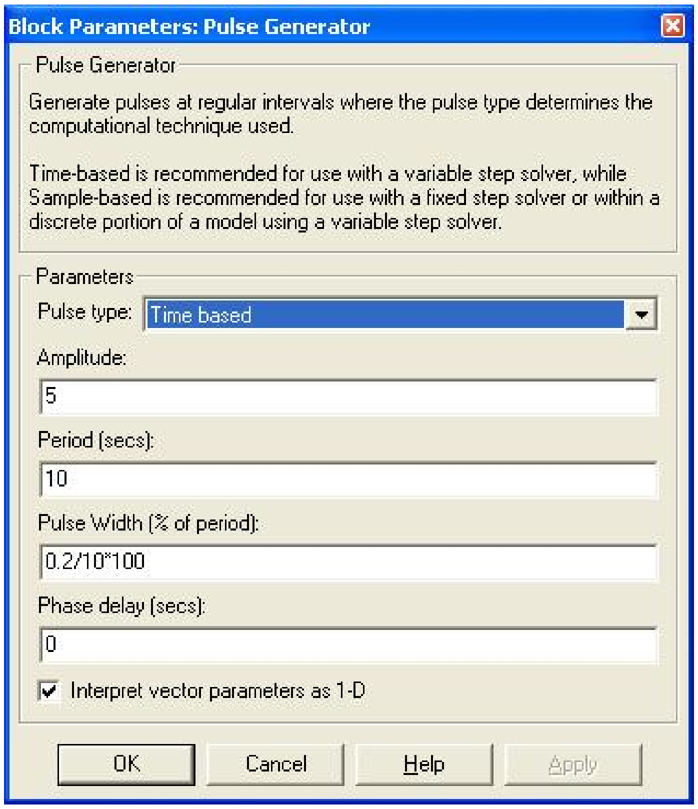
The `Pulse Generator` introduces the impulse function

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

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 $\epsilon = 5$ is the lower limit for our system. Pulse width is calculated using the following formula:

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


Block Parameters: Pulse Generator

Pulse Generator

Generate pulses at regular intervals where the pulse type determines the computational technique used.

Time-based is recommended for use with a variable step solver, while Sample-based is recommended for use with a fixed step solver or within a discrete portion of a model using a variable step solver.

Parameters

Pulse type: Time based

Amplitude: 5

Period (secs): 10

Pulse Width (% of period): 0.2/10*100

Phase delay (secs): 0

[✓] Interpret vector parameters as 1-D

Figure 2.3: `Pulse Generator` configuration for Lab 2

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 navigating to the `simulation` header → `simulate` and entering a longer time in the `Stop time` box. You can also modify the end time through the `configuration parameters` window (`CTRL+e`). Start with $\epsilon = 2$ and increase 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 2.4. Note that there is no `HIL Write Analog` block required in this case since you are providing the input action. Using Simulink, build the model. Under the `hardware` menu, select `Monitor and Tune`. 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.

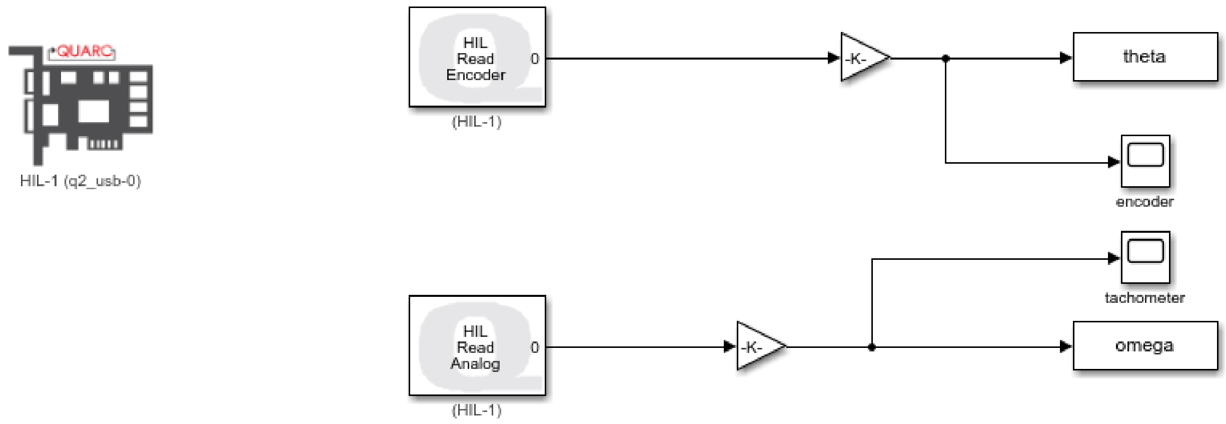


Figure 2.4: Simulink model for Lab 2 for the manual impulse

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.

10. To obtain a “better” impulse, we introduce the `Pulse Generator` model as shown in Figure 2.5. 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 $\epsilon = 1$ using `Hardware` $\rightarrow$ `Monitor and Tune`.

12. Run the model again with $\epsilon = 5$.

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?

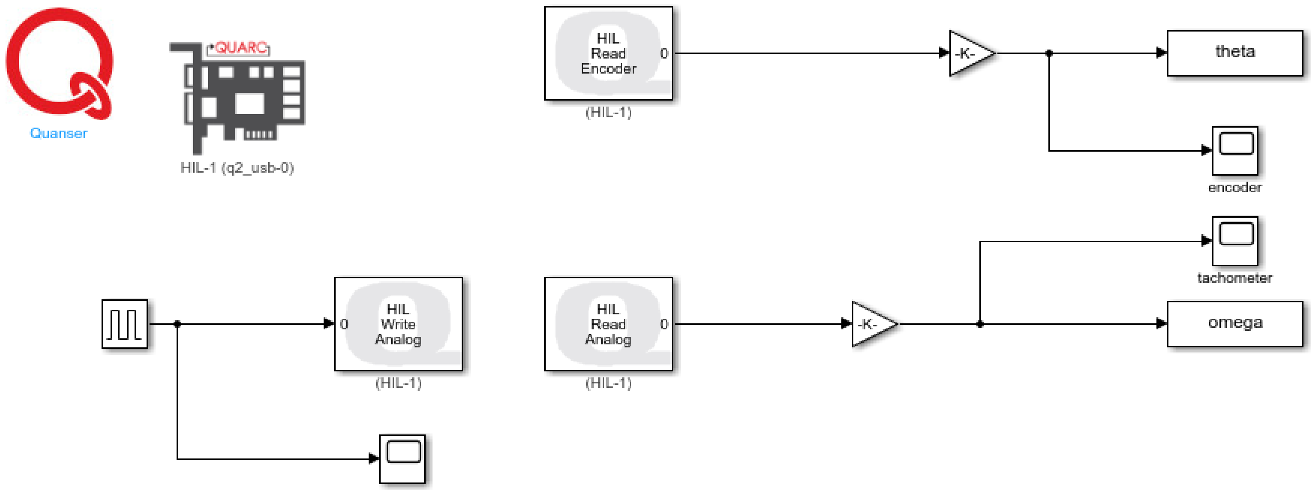


Figure 2.5: Simulink model for pulse generated response in Lab 2

2.2.2 Deliverables

1. Include the state space and output equations (i.e. state what A,B, C, D are) for theta and omega. Note there will be two different equations.
2. Include the graphs of the simulated impulse response for theta and omega.
3. Include the plots from the physical jolt response for theta and omega.
4. Include the plots from the powered impulse response for theta and omega, $\epsilon = 1$.
5. Include the plots from the powered impulse response for theta and omega, $\epsilon = 5$.
6. Comment on the similarities and differences between these responses.

Note that this may be slightly longer than 5 pages because of the graphs, but it should not be excessively wordy.

Lab 3
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 3.1.

$$\begin{tikzcd}
\hat{u}(s) \arrow[r] & \boxed{\dfrac{k_E}{s(s+\frac{1}{\tau})}} \arrow[r] & \hat{\theta}(s)
\end{tikzcd}$$

Figure 3.1: Open-loop motor schematic in frequency domain

3.1 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 3.2 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

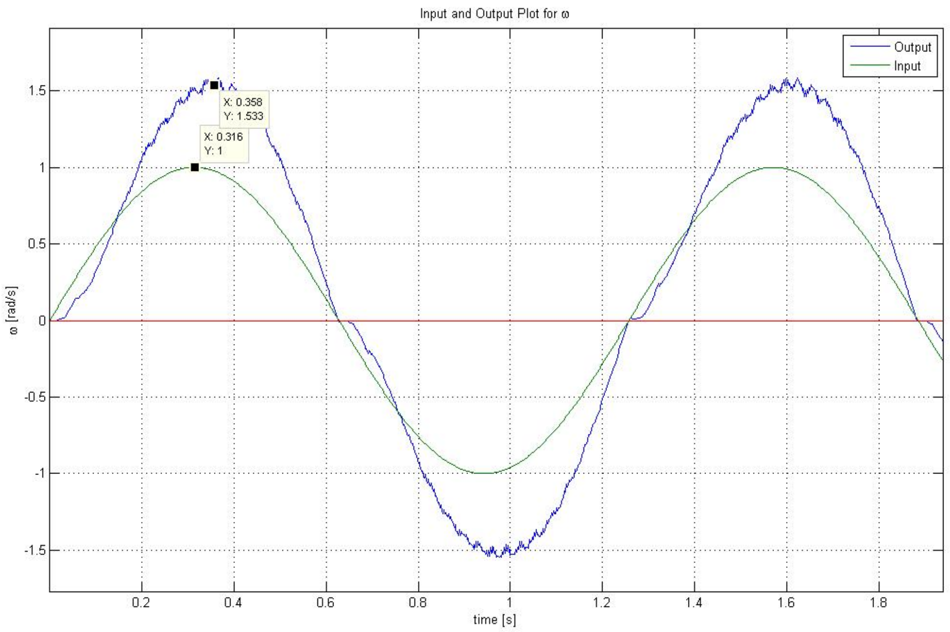


Figure 3.2: Matlab generated plot of the input and output of the system for $\omega_u = 5$ rads/sec

given for your project, which will allow you to find a model transfer function for the "Plant".

3.2 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

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

3.3 Procedure

3.3.1 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 3.1: Data collection table

3.3.2 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.

3.3.3 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 3.2, 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 3.3 and build a Simulink model to implement this signal. *Alternatively, you can modify your model from lab 2.* The `Fcn` block can be found in Simulink in

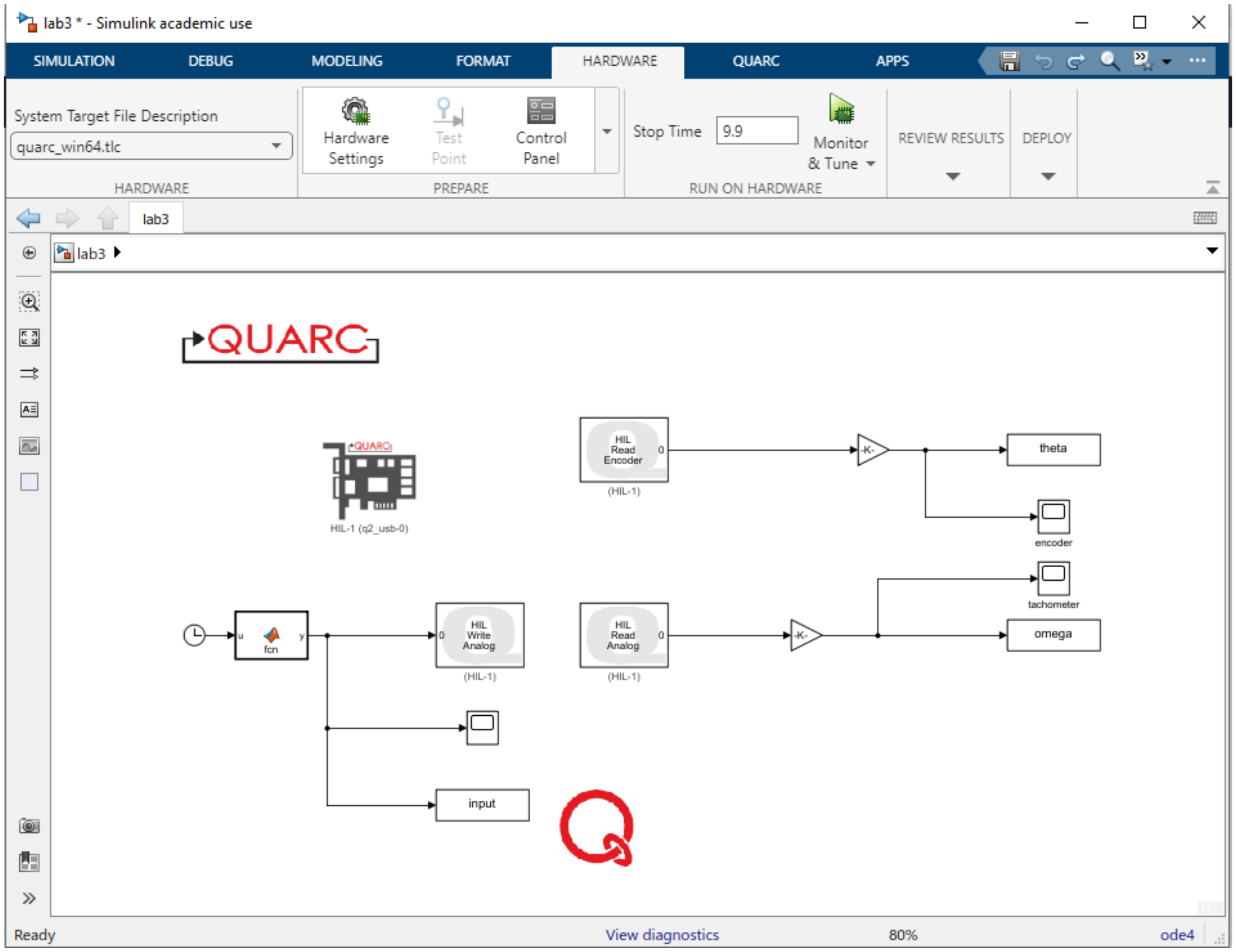


Figure 3.3: Simulink model for the implementation of the signal $5\sin(t^2)$

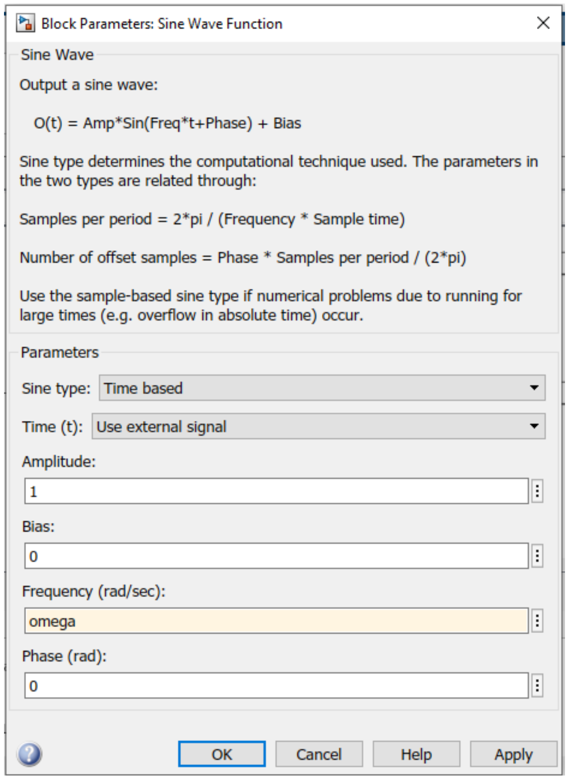
the `Library Browser` by searching `"MATLAB function"`. Double-click on your `Fcn` block to enter the function `y = 5*sin(u(1)^(2))` where `u(1)` represents the time. The time is obtained by introducing the `Clock` block as an input to the function. The output of the function block is simply connected to the `Write Analog` block. Do not forget to change the Solver type to `Fixed Step` and `solver` to `ode 4` in settings (`Ctrl+E`). Also, change the gain constants to the desired units, as in Step 13 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.

4. Monitor and Tune 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 `Fcn` block 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 3.4 below.


Block Parameters: Sine Wave Function

Sine Wave

Output a sine wave:

O(t) = Amp*Sin(Freq*t+Phase) + Bias

Sine type determines the computational technique used. The parameters in the two types are related through:

Samples per period = 2*pi / (Frequency * Sample time)

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use external signal

Amplitude: 1

Bias: 0

Frequency (rad/sec): omega

Phase (rad): 0

Figure 3.4: Configuration for the implementation of the signal $\sin \omega_u t$

In this case, the variable `omega`, is undetermined. The value of ω_u must be specified in the block parameters dialogue box above before each run. The amplitude of u should be set to 1.

7. Build the Simulink model using `monitor and tune`. 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.

3.4 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 3.1

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

Lab 4
BIBO Stability of control 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 control 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 2 if you saved it, just change the `Pulse Generator` to the `fcn` and `clock` blocks.

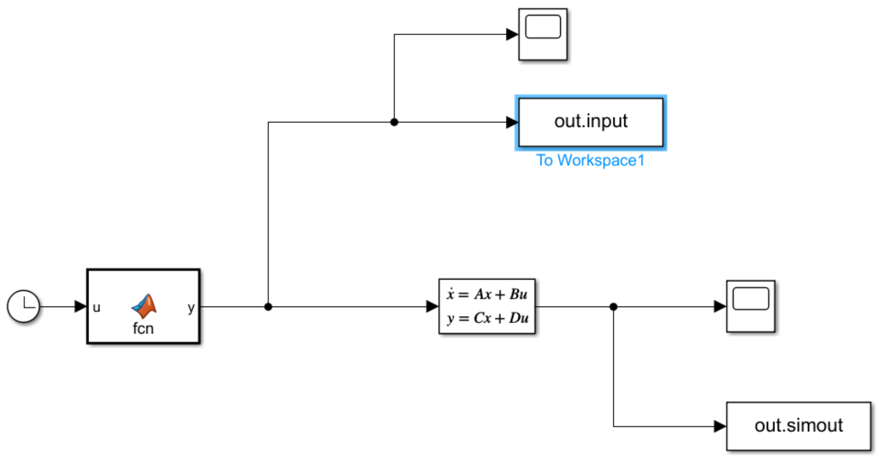


Figure 4.1: Simulink model for Lab 4

5. If you determined that the system is not BIBO stable specify a bounded input into the `fcn` block 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. 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

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

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

4.2.1 Deliverables

For the three systems in the lab include the following:

1. What are A, B, C?

2. The impulse response of the system (a plot is required, the equation describing it is optional). Comment on whether or not you believe the system is BIBO stable based on the impulse response.

3. The transfer function of the system and its poles. Comment on whether or not the system is BIBO stable based on this information.
4. A simulated input/output plot which supports your claim of whether or not the system is BIBO stable.

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

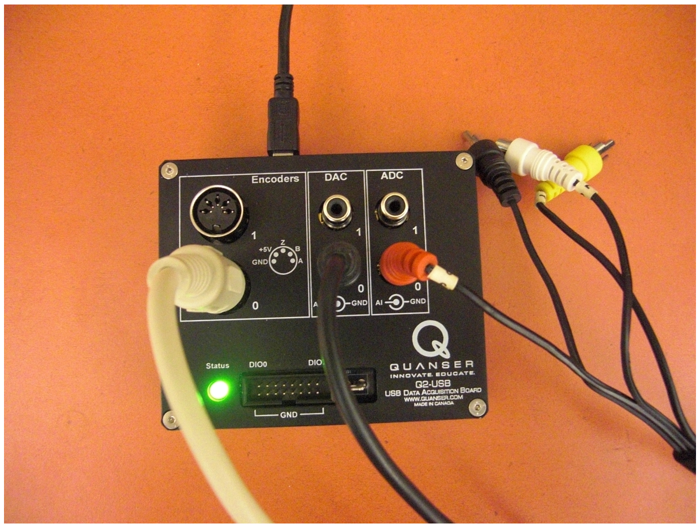


Figure C.1: Q2-USB data acquisition board

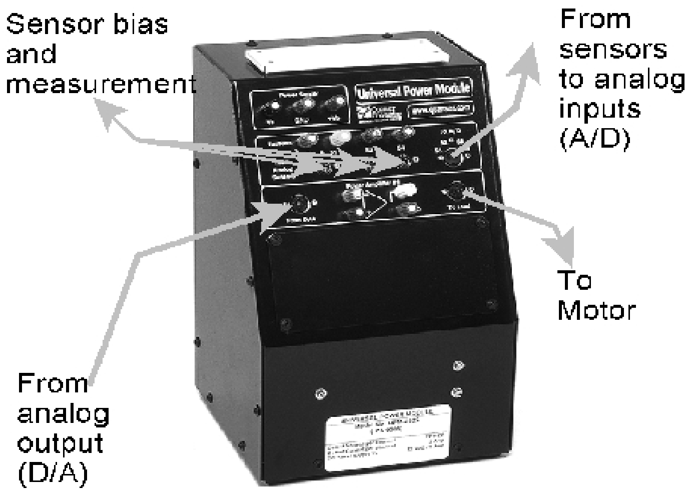


Figure C.2: Universal power module

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.

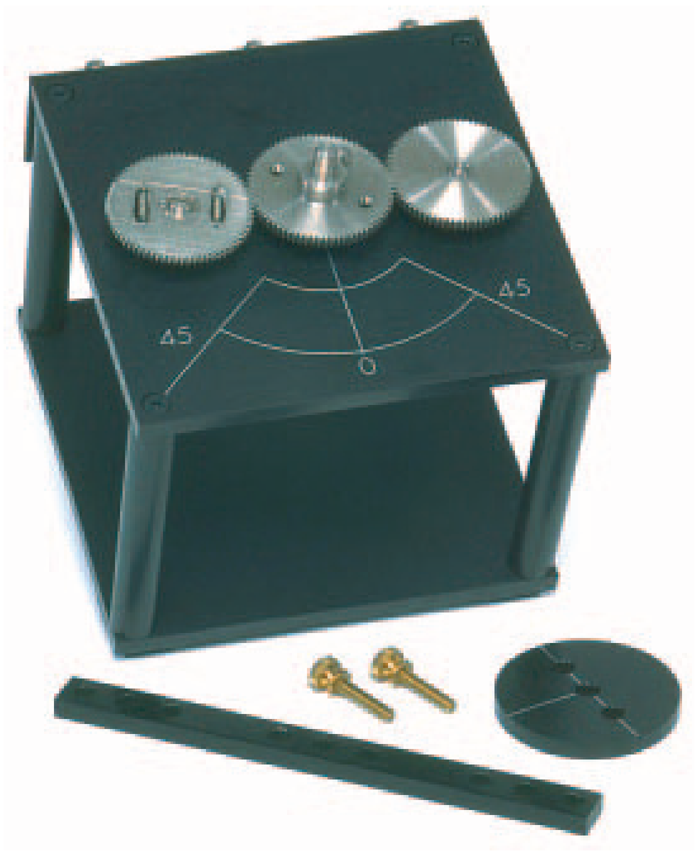


Figure C.3: DC servomotor (SRV02)

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