The Fourth Box: Ted S., Philip A., Josh C-S., Robert P., Ben A.

ECE:4890

Design Documentation First Draft

12/13/19

Inverted Pendulum Control Simulation & Calibration for Lab Experiments

Introduction

The goal of this project is to develop an improved lab experiment for ECE:3600. The lab experiment will allow students to enter a controller transfer functions into MATLAB to control an inverted pendulum system. This differs from past lab experiments where students only entered a gain parameter. To make this possible we developed a software package, including a simulation, that can be calibrated to the physical system. The simulation was developed with a Graphical User Interface where students can enter controllers, tweak simulation parameters, and see the results. The GUI was motivated by efficiency of use so students could quickly test different controllers without the need for extra scripting.

Project Outcome

At the end of our project we met all our design goals to a degree satisfying our team and our sponsor. Our sponsor Prof. Andersland wants to use our project for Lab 2 of the spring ECE:3600 course. We had to make several last-minute simplifications to our product which we will talk about. The simplifications ensure that our sponsor will find our product complete and error-free in the months to come.

The simulation is finished. It can be calibrated to the physical system. The user can enter a large class of transfer functions. We settled on a very simple and very efficient method for implementing a user's controller in our simulation.

We calibrated the simulation to predict stability about 95% of the time for the important controller types (PI, PD, PID, and lead-lag) and achieve as low as 5-10 % error in certain response characteristics and 20-30% in others. This is a reasonable degree of accuracy for using the simulation in the course lab setting. We will talk about these measurements and why such results occur. Overall It was a much more time-consuming process than expected to calibrate the simulation above a certain degree of accuracy. However, we managed to identify several causes for discrepancies and account for them in the simulation.

The simulation GUI ended up as an unfinished template that was replaced by a functionally equivalent CLI. This is the first simplification we had to make. The user can use the CLI to do everything the GUI was supposed to do: enter controllers, configure various simulation data, enter a calibration file. We

decided the output data should be given to user's as raw .mat files with an example plotting script rather than an excessively large collection of well-formatted graphs. This gives user's practice in finding the response characteristics in the output data on their own.

The lab experiment CLI and control logic is completed. Students, TAs, or instructors can run controllers on the physical system through this CLI. They can also start, stop, and pause the model while it is running. This CLI has a similar style to the simulation GUI so it should be easy for users to learn it after playing around with the simulation.

The friction calibration procedure is fully completed though simplified. In November we expected to have one calibration procedure for static friction and one for dynamic friction. It turned out that these could calibrated with the same procedure so there is only one calibration procedure. It can be done in the lab with an associated CLI we wrote for it or from scratch with a list of directions.

Our group roles shifted in the final weeks. Ted worked on the simulation and testing controllers to obtain performance measurements and response comparisons. Robert continued working on the GUI. Philip worked on the CLIs. Josh helped develop the Simulink models used to run experiments. Ben helped in testing controllers and analyzing data between the simulation and lab experiment.

Design Documentation

Design Concept:

The initial design concept involved simulation and calibration that would help develop a lab experiment. Originally, we did not know the programming languages or tools we would use to make the simulation, but quickly converged on using MATLAB's Simscape library. The Simscape library is a physics engine that simulates multi-body systems defined through Simulink. There really was no better choice considering the lab equipment uses Simulink. Many aspects of how the real system is operated and how the simulation is operated stay the same. The simulation needed to take user input. We decided building a Matlab-based GUI for this purpose and finally turned it into a CLI the performs the same function. This turned into building a full software package that would allow student's to design controller transfer functions in a simulation and use them on the lab equipment in the same form.

The specific lab experiment template we designed for is one where students try to regulate the tip position of the pendulum using the motor voltage. Students first need to find controller transfer functions by hand or by using Matlab's many linear design tools. The emphasis is on pole-zero placement so we made the simulation CLI allow students to first enter a controller 'kernel' and then add compensators on top of it. This line of reasoning fits the ECE:3600 material.

In order for these mathematical tools to work in a sensible way, we had to find an accurate system transfer function for students to use. In the end we couldn't settle on modifications (added poles and zeros) to the uncalibrated system transfer function. Using the uncalibrated system transfer function will predict stability well especially if it is cross-verified with the simulation. If the simulation is stable then the system transfer function will give good results. They may however give very results when the simulation is marginally stable. But this is expected because the system transfer function is linearized. In summary, students can use the simulation avoid obtaining bad results which is an important part of the

design concept. For future investigation, we have an idea of how the system transfer function may be modified using our calibration using pole-zero placement which we will talk about. This would be a deeper calibration than modeling the nonlinear forces.

Standards and Constraints:

In general, we want to ensure the accessibility of our software to its users and the longevity of the hardware it controls. User safety is also a concern that motivates robust software and complete documentation.

- Simulation, CLI, and GUI constraints
    - We expect the simulation to be used on any computer with Windows 10 and MATLAB 2016-2019. We wrote the simulation in R2019a. It uses only one additional library known as "Simscape Multibody" that works used in R2016b to R2019b. Thus, anyone with R2016b-R2019a and Simscape can run our simulation. (MET)

- The lab experiment models and CLI and calibration are run on the control systems lab computers that use Windows 7 and R2011a. We require no additional software on these computers. (MET)

- There are safety/longevity constraints to the lab experiment that our software must enforce that we refer to as the "lab control logic".
    - During the lab experiment, if the pendulum angle exceeds a threshold between +- 10-15 degrees or the cart position exceeds a threshold of +40 cm to -40 cm, then the voltage to the motor is cut. (MET)
    - The user must be able to safely place the pendulum in the upright position from the down position before starting their controller. This was a concern because the system starting angle is always the 0 degrees and the controller reference angle is always 180 degrees from the starting angle. We have found a way to satisfy these conditions. (MET)
    - The user must be able to stop the controller at a given time. The user must be made fully aware when the controller starts and stops by a CLI. (MET)
- Similar constraints are placed on the calibration:
    - The user must be able to safely place the cart at the center of the track before starting the calibration procedure. (MET)
    - Again, the user must be able to stop the calibration at a given time. The user must be made fully aware when the calibration starts and stops by the CLI. Also, in the case of calibration they should know what "iteration" the procedure is at. (MET)

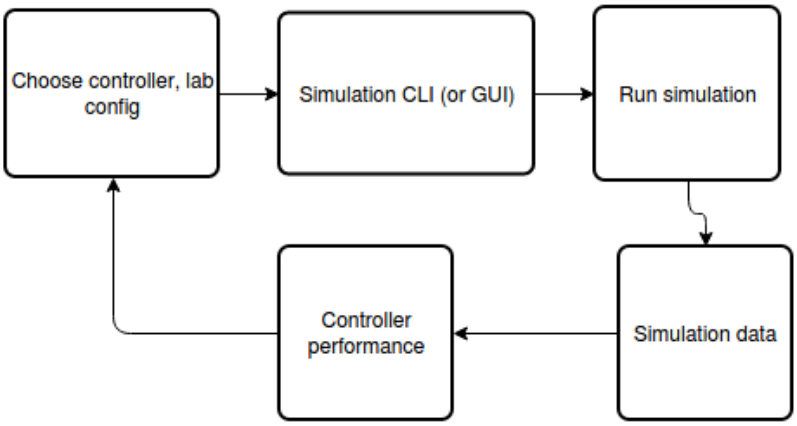
<u>Architecture & User Interface</u>

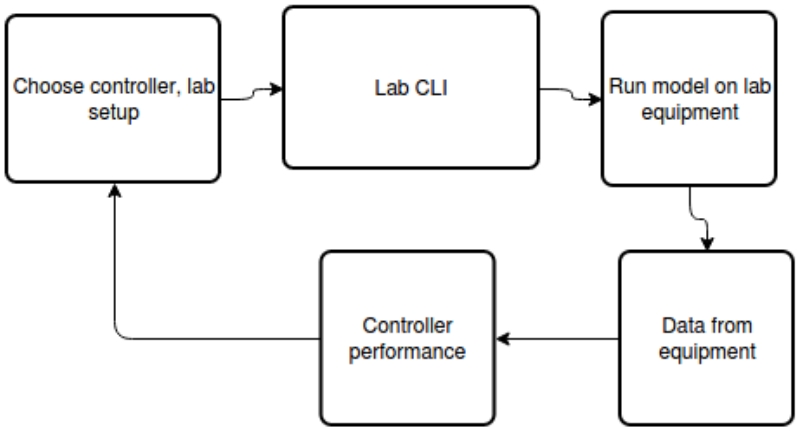
Top Level:

At the top level our system archtecture is simply stated. The upper loop in Figure A1 shows the the basic simulation workflow. The user (student) finds a controller transfer function enters it into the simulation CLI in pole-zero-gain format. The transfer function object is passed to the simulink model that runs the simulation. The simulation runs at a given time step until the user-chosen stop time. Data computed during the simulation run time is output to .mat files which can then be used to evaluate the performance of the controller with a script. Of course, the simulation has a visualization which can be used to evaluate the controller performance qualitatively.

The controller performance then influences the next controller choice and so on. The lower loop shows the lab experiment workflow using the lab equipment. Intentionally this workflow is basically the same where the user can enter their controller AS IS into a CLI running on the lab desktops. The controller is passed to the simulink model which is downloaded directly onto the hardware. Data from the run is stored in .mat which are then analyzed to evalute controller performance. This lab experiment will involve comparing the lab data to the simulation data.

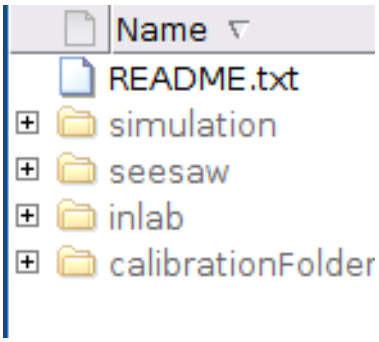
Figure A1: Upper loop -> Simulation Workflow, Lower loop -> Lab workflow





Our product is software package. The final package is on github at https://github.com/physiqueguy/sdesignlab. Additionally our GUI implementation is up on …. . The software package a directory structure the top level of which is divided into 4 folders: simulation, inlab, calibrationFolder, and seesaw. The simulation folder has all the matlab materials for the simulation and simulation CLI. Inlab has all the matlab materials for running controllers on the lab equipment. The calibration folder stores the user's calibration file which is accessed by the simulation. Finally the seesaw folder has a 2nd simulation which implements a control system and visualization for a seesaw-cart system that can be adapted to work on the existing lab equipment.

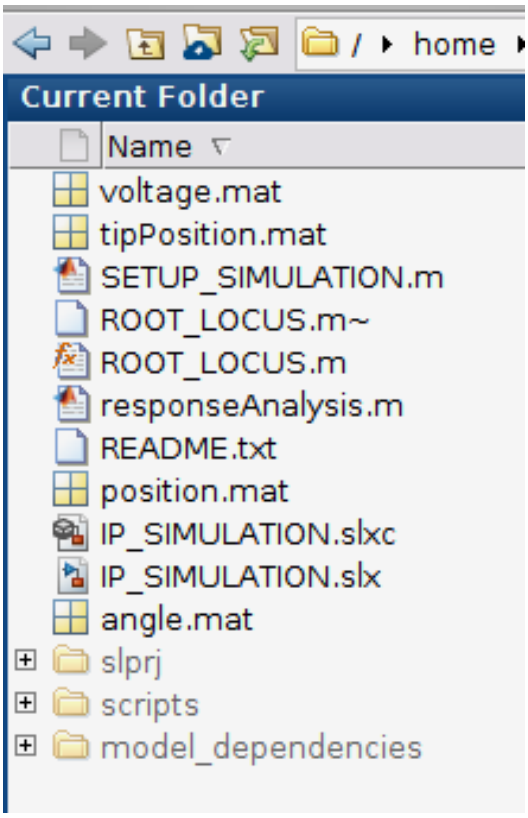
Figure: Top Level Directory Structure



Simulation

Now we will detail the architecture to greater specificity starting with the simulation. The simulation is built in MATLAB Simulink with R2019a. There are two parts: the dynamics equations and visualization/rendering. The visualization uses the Simscape Multibody library. The full simulation works in R2016-R2019 but the dynamics part of the simulation can be built R2011a to be used in the lab if necessary. The simulation window is shown in Figure A2. The simulation folder directory contents are shown below. The simulation file is “IP_SIMULATION.slx”.

Figure: Simulation folder contents



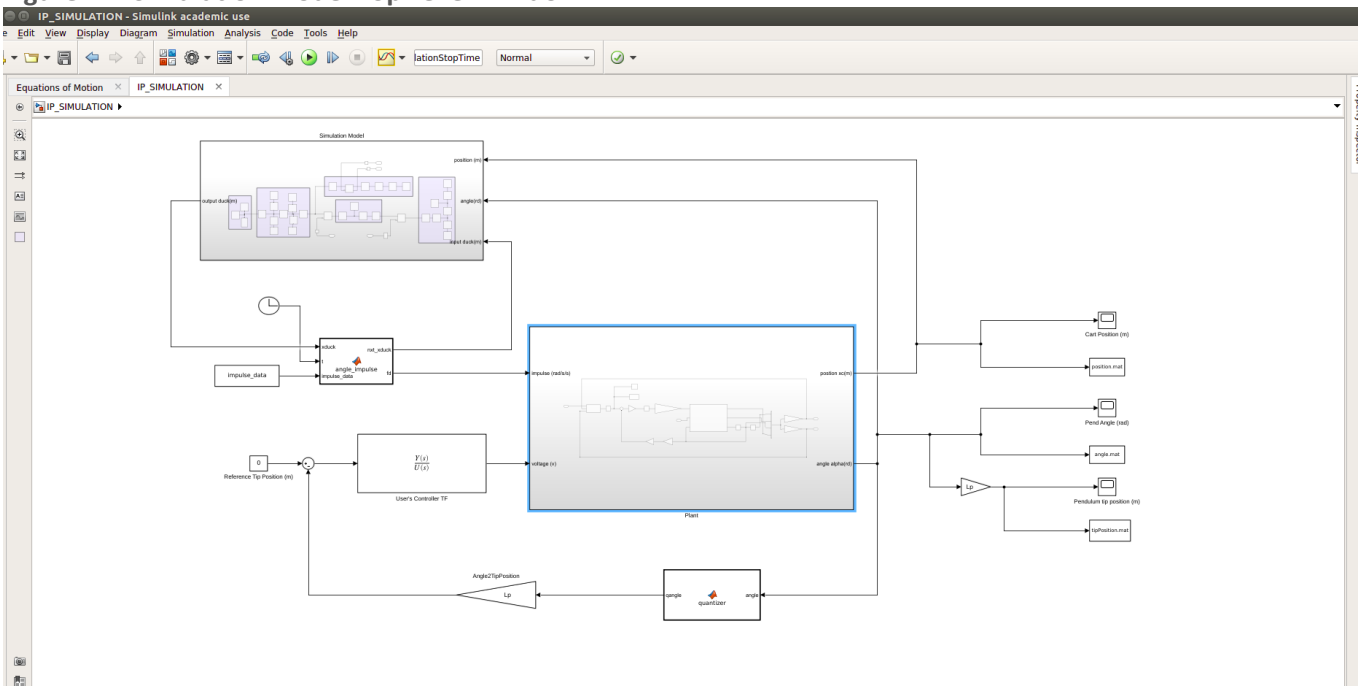
In the simulation simulink model the center block is the “plant” which computes the dynamics at each time step. It takes the controller input voltage and computes the current system state (position x1,

velocity x2, angle x3, angular velocity x4). The angle (x3) is quantized, converted to a tip position, subtracted from a reference tip position (0 m) to get the tip position error which is passed into the transfer function block which implements the user's controller. The tip position is pendulum length * angle.

To mimic the real system the pendulum angle value must be quantized to 0.001 rad precision which is what the angle optical encoder does in the real system. This is an important part of calibration we figured out at the end. We will show how this effects the system performance. The simulation accounts for the quantization but the system transfer function, which the user uses to design controllers, does not. In order to account for the quantization at a transfer function level perhaps some poles on the jw axis must be added that correspond to the stepped oscillations with frequency related to the time step. We didn't come to a conclusion on this issue.

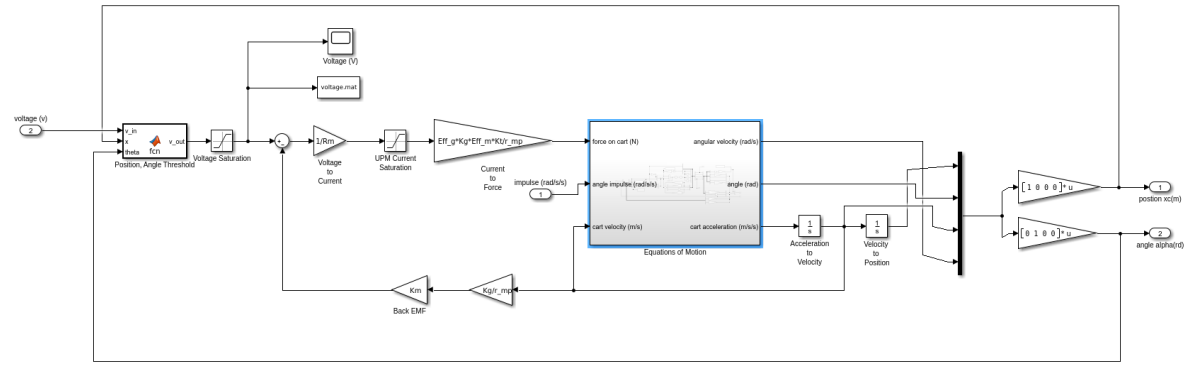
Simulation Dynamics

Figure A2: Simulation Model Top Level Window



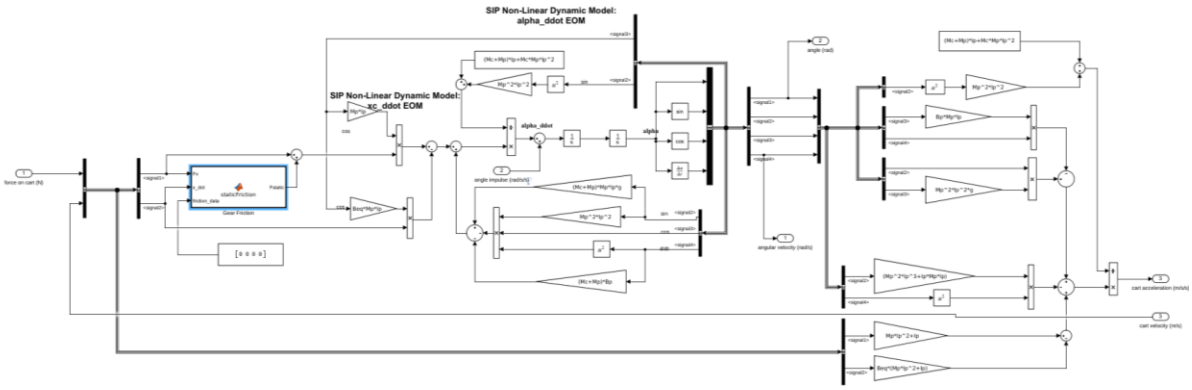
plant block (center in Figure A2) is shown in Figure A3. It's top level operation is converting voltage to system state through the system's dynamical equations. Wherever possible the model matches the real system's technical details. The voltage is saturated to +- 10 V and converted by a constant to the current applied to the motor which is then converted by a constant to the force applied to the cart. The force on the cart is passed into the "Equations of motion" block (center). The output of this block are positional and angular accelerations which are integrated to the state variables. Since the motor has a back-EMF as a function of it's velocity the back-EMF is fed back and subtracted from the input voltage.

Figure A3: Inside the plant block (center) which computes the dynamics



The contents of the “equations of motion” block (center of figure A3) are shown in figure A4. This is the bottom level of the dynamics. This block converts the force on the cart (from left) to the acceleration and angular acceleration (at right) of the cart-pendulum equations through the dynamics equations governing the lab equipment system. The full nonlinear equations are used with sine and cosines of the pendulum angle. The function block (left-center) is the static friction function which implements our friction calibration. This function is shown in Figure A5 and explained.

Figure A4: Equations of Motion block



Simulation Calibration (Friction Model)

The function in Figure A5 implements our friction calibration into the simulation. One can plot this function with respect to cart velocity to see intuitively what the model is. It’s only slightly more complicated than idealized coloumbic friction. Paraphrasing:

Static friction force = function(Force on cart, cart velocity)

If the force on the cart is less than a measured minimum threshold AND the velocity is less than a minimum velocity threshold THEN the net force on the cart is zero meaning the cart is stuck in place. That is if the cart is approximately not moving then a minimum force is required to produce motion. The static friction force is the negative of the force on the cart. If instead the force on the cart is less than a SECOND minimum theshold AND the velocity is less than a SECOND minimum velocity threshold THEN the static friction is a strong viscous damping force. We will explain how this is measured in a few sections.

Our measured data for this function is summarized:

Minimum force threshold 1: Fmin = 1.5 N

Minimum velocity threshold 1: Vmin = 0.5 cm/s

Minimum force threshold 2: Fmin = 3 N

Minimum velocity threshold 2: Vmin = 5 cm/s

Damping coefficient: B = 20 kg/s

Figure: Static friction function block takes friction_data vector from calibration file

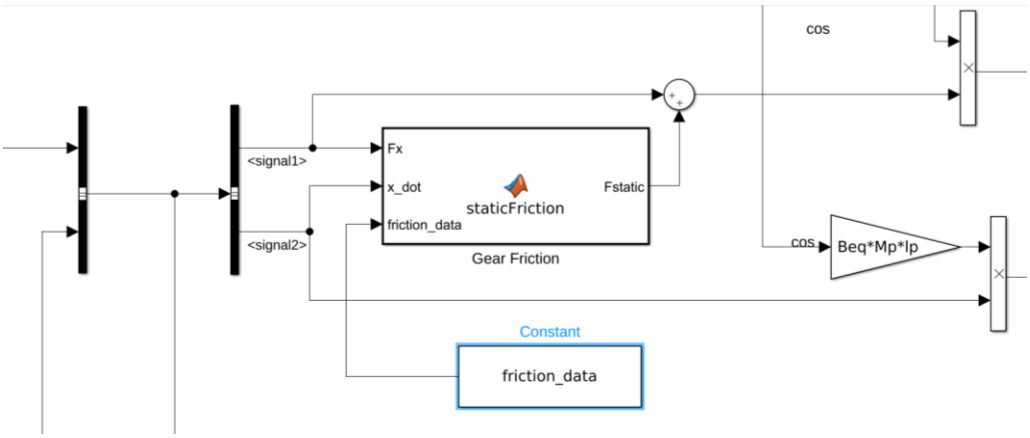


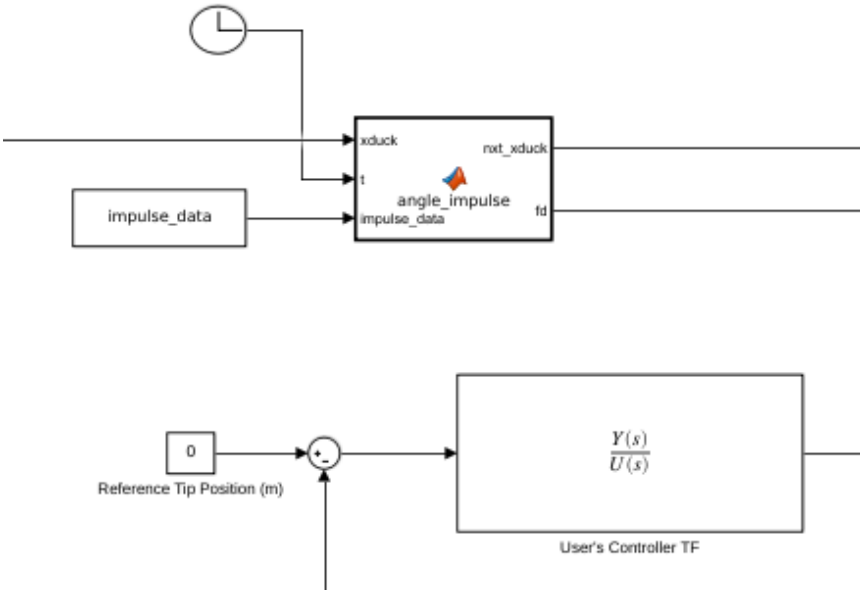
Figure A5: Code behind the static friction model

```
function Fstatic = staticFriction(Fx, x_dot, friction_data)

Fstatic = 0;

% x_dot_max0 =0.005; x_dot_max1 =0.1; Fx_max = 2; b1 = 20; c1 = 0.5;

Fx_max = friction_data(1);
x_dot_max0 = friction_data(2);
b1  = friction_data(3);

% Cancel net force on cart if velocity ~ zero, force < threshold
if abs(x_dot) < x_dot_max0 && abs(Fx) < Fx_max
    Fstatic = - Fx;
% Else add large viscous damping if velocity < theshold, force < theshold
elseif abs(x_dot) < x_dot_max1 && abs(Fx) < 4*Fx_max
    Fstatic = -b1*x_dot;
% No added friction force velocity > threshold, force < threshold
else
    Fstatic = 0;
end
end

```

Simulation User Input

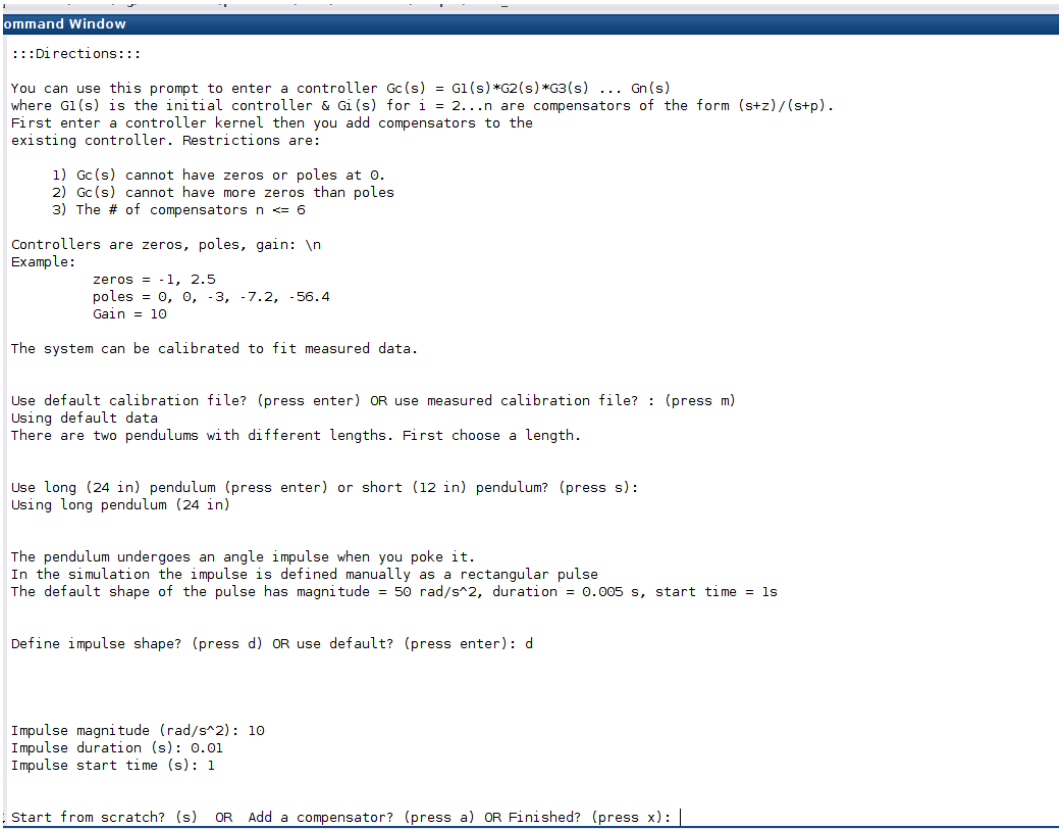
The user has access to the file “SETUP_SIMULATION.m” which calls the “LAB1_PROMPT.m” file in the simulation/scripts folder. The prompt allows them to enter a pendulum length, simulation run time, and a calibration file (containing a measured friction_data vector) and to define a controller transfer function and a rectangular impulse. The controller is entered as a list of zeros, poles, and a gain, converted to a numerator and denominator and entered into the Simulink transfer function block “User’s controller TF”. The rectangular impulse is implemented in the “angle_impulse” function block. This block applies an impulse (or poke) to the pendulum that is defined by user in terms of (magnitude, duration, start time). These blocks are shown in Figure A6.

Figure A6: Controller and Impulse blocks that use user data



We made a simulation CLI which implements what the GUI was originally intended to do. This CLI or command prompt is shown in Figure A7. Reading the figure one can see the user is given directions to enter a controller in ZPK format, enter a pendulum length, a calibration file, and define an impulse.

Figure A7: Simulation CLI (command prompt)



The user's transfer function data is stored in Matlab's native TF object and can be displayed on the command line (shown in figure below) after being entered through the command prompt above. It is passed into the "User's controller TF block" in Figure A6. When the command prompt is complete the user can immediately run the simulation.

Figure: User's controller displayed on command line

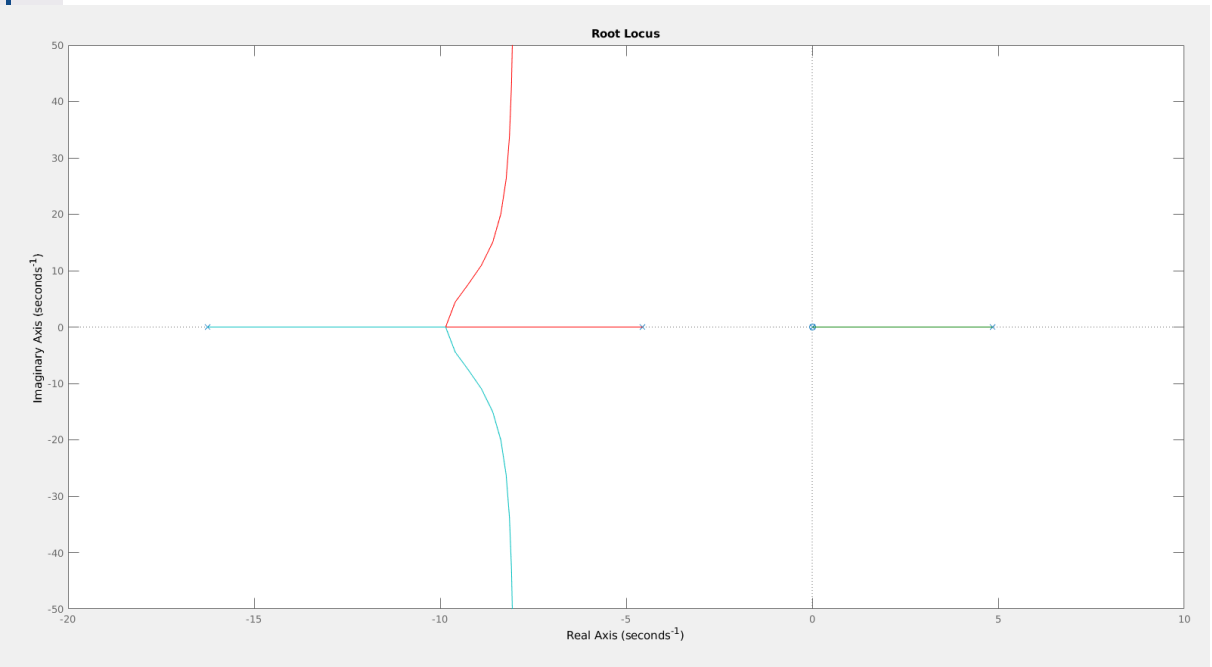
```
controllerTF =

  700 (s+62) (s+10)
  -----------------
   (s+55) (s+0.1)

Continuous-time zero/pole/gain model.
```

It's worth pointing out sooner than later that the open-loop transfer function we are using for the system is shown in the Figure below. This is open-loop transfer function from voltage to angle (rather than force to angle) computed with the linear dynamics alone. We have yet to find a more correct version based upon our calibrations. In one wanted to integrate the static friction effects into the system transfer function our best guess is to linearize the resulting effect as a constant force disturbance fd or a slow spring effect –k*x in the dynamical equations. This would result in new state-space equations and thus a new transfer function.

Both our simulation CLI and lab CLI allow the user to first enter a "kernel" controller G(s) then add compensators G1(s), G2(s), and so on. One of the restrictions we'd place on the student's controllers is that don't use perfect integrators or differentiators, I.e. derivatives must be filtered and integrators must integrate over a recent window of time. The derivative low-pass filtering is especially crucial to avoid large spikes in voltage in derivative based controllers. What we recommend starting out is turn the 's' derivative into 's/(s+62.8)' and the '1\s' integrator into '1\(s+0.1)' or '1/(s+1)'. As such PI, PD, and PID controllers into lead, lag, and lead-lag controllers. This suggests a method of designing controllers by chaining compensators which the CLI takes into account. Instead of fixing the derivative and integrator filters we thought of just letting students account for this in their controller transfer functions though they could perhaps could be inscribed into the system transfer function.

Figure(s): Open-Loop Transfer function of system G(s) = angle(s)/voltage(s) (without potential calibrations) + Root Locus

```
ans =

        5.247 (s^2 + 4.777e-16)
  -------------------------------
  s (s+16.26) (s+4.561) (s-4.843)

Continuous-time zero/pole/gain model.
```



Simulation Visualization/Rendering:

The top-left block in Figure A2 holds the Simscape Multibody network shown in Figure A8 that automatically defines a the system body to be rendered. The system state (position, angle) is computed by the dynamics and passed into this block to be computed as an effect on the body's joints. The pendulum is coupled to the cart by a "revolute" joint that allows 1D angular movement and the cart is coupled to the track by a "prismatic" joint that allows 1D translational movement. The cube objects are solids: the pendulum pole, its tip, the cart block, the track rails, the end blocks, etc. The coordinate objects are translations and rotations to place the solids at the correct spatial locations. The state inputs are passed into the joints whose degrees of freedom a parameterized by these inputs. The result the full simulink model rendering is shown in Figure A9. The pendulum is struck by an impulse (poke) which we animated as a duckling striking the pendulum at hypersonic speed. The pendulum then wobbles and is (hopefully) stabilized by the user's controller.

Figure A8: Defining the visualization through Simscape multibody

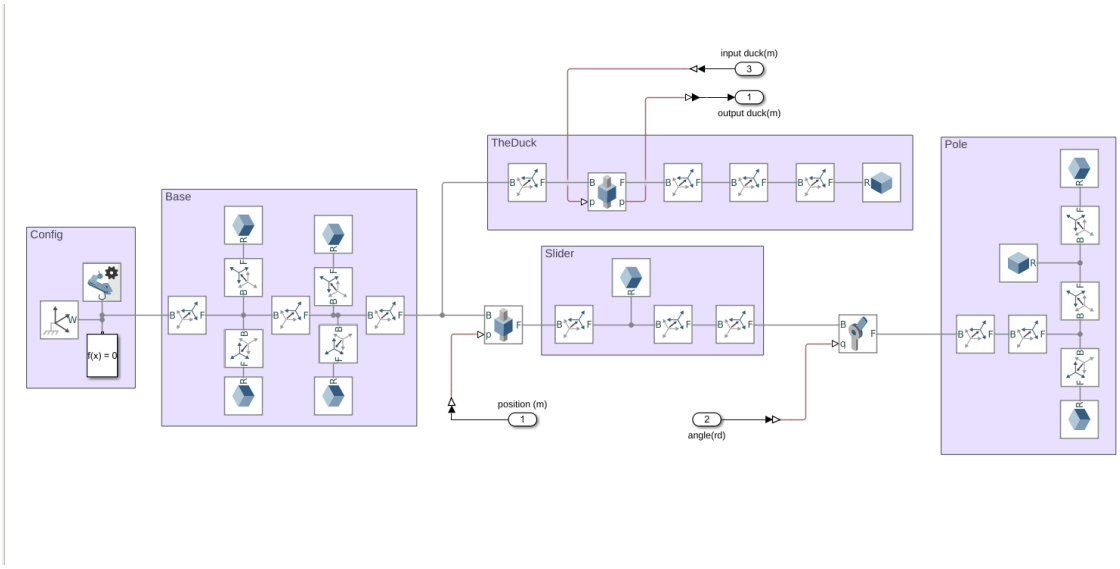
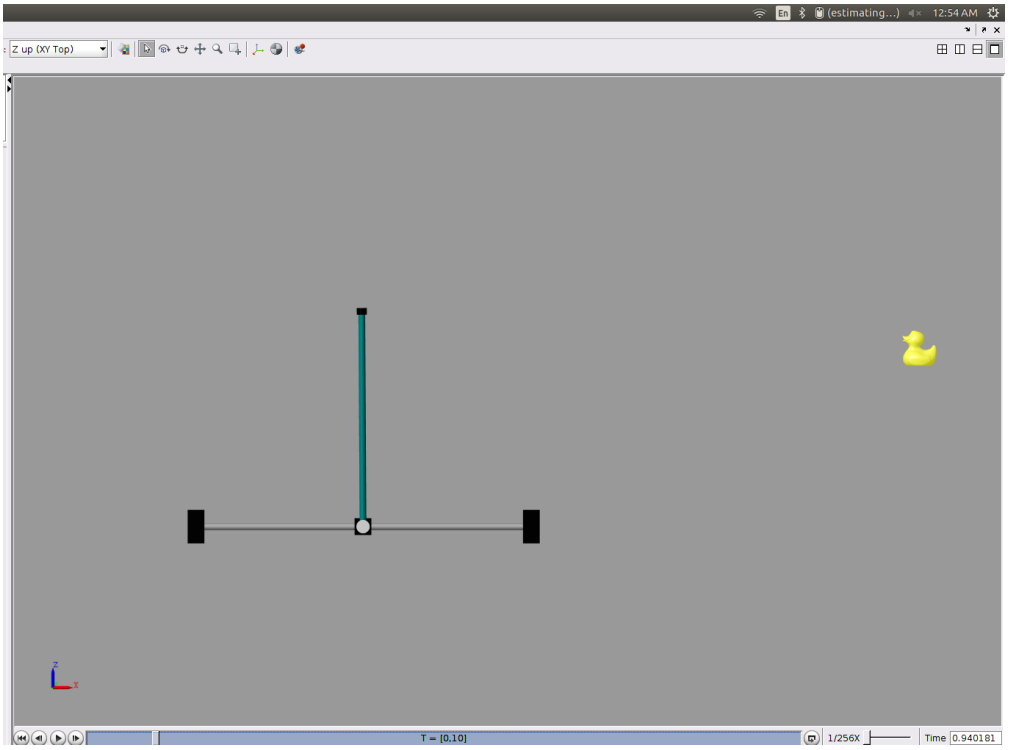


Figure A8: The Simulation Rendering with Impulse from Hypersonic Duckling



When the simulation is complete the position, angle, and voltage are stored in .mat files to be analyzed. Below we will show simulation data and compare it to the same controllers on the lab equipment.

Test Report

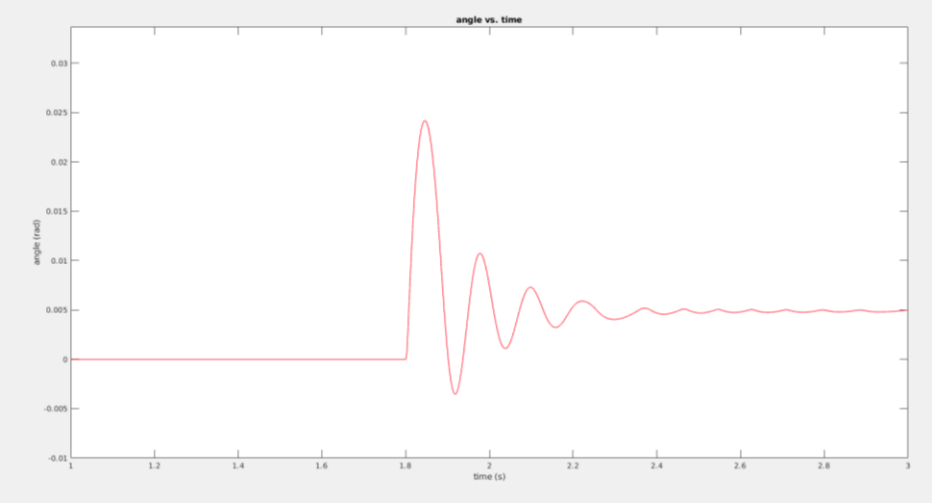
Output Data from the Simulation and the Lab Equipment

We have data for 3 different PID controllers tested in both the simulation and on the lab equipment. They use the same filters and could be written as lead-lag controllers. One difficulty in obtaining lab data is the fact that cart position is uncontrolled and is subject to the static friction forces. The cart position often continues oscillating or runs off the track. The voltage is cut before the cart runs of the track but a useful addition would be to incorporate a “kickback” controller that pushes the cart back into play if it gets out of bounds. We tested 40-50 PID controllers in the simulation and the lab. The simulation can predict stability of these controllers > 95 % of the time. In terms of controller performance comparisons, the peak time, rise time, percent overshoot, and natural frequency were in reasonable agreement. The settling time was more of a disparity. We thought this represented a failure of our friction model to model the full complexity of the gear meshing. Once we added in the angle quantization from the encoder we saw greater agreement on this measurement. The x-speed change in the data basically represents how fast the pendulum runs off the track (and voltage is cut) as the position is unregulated. In the future we would recommend adding a position controller that simply pokes the cart back onto the track before it goes out of bounds.

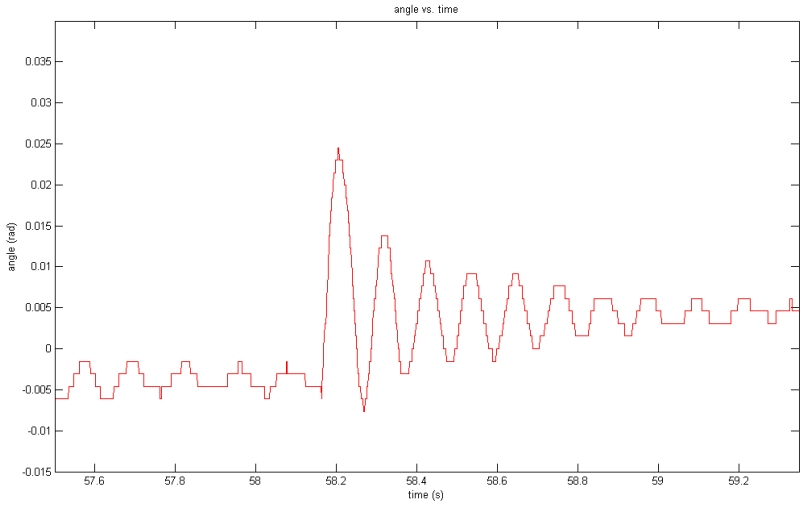
Below are the results plotted for 3 different controllers.

Run 1: G(s) = 600 + 200/s + 5*s, Filter cutoff = 62.8 rad/s, Integral time constant = 0.628 s

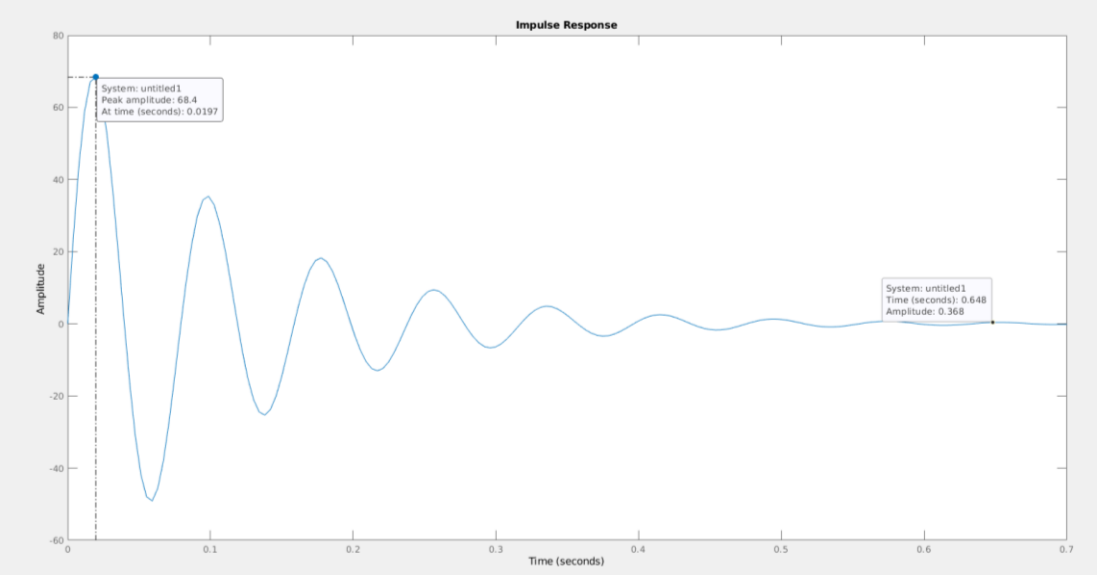
Simulation Impulse Response



Measured Impulse Response



What the System transfer function predicts:



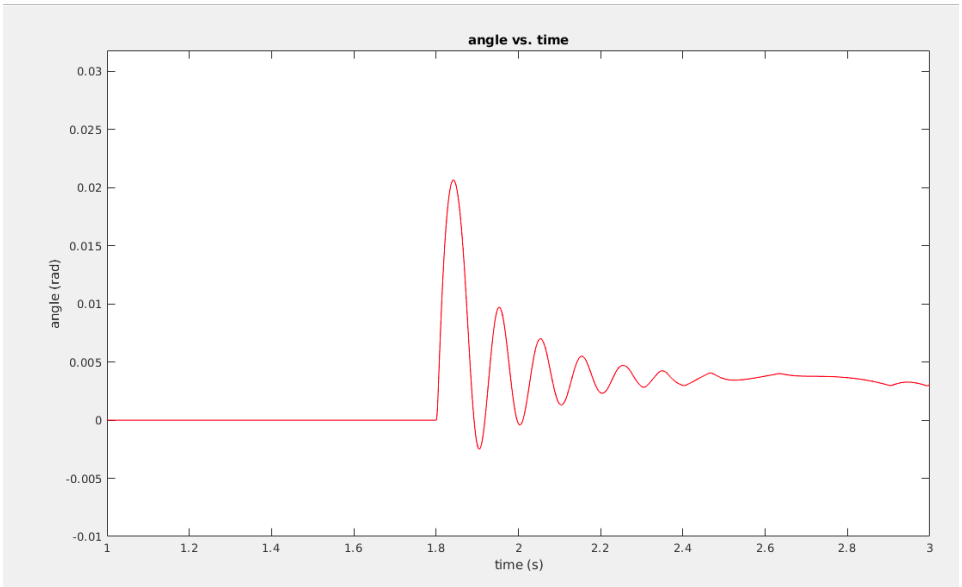
Run 1 Data:

	Simulated	Measured
Natural Frequency (rad/s)	79	80

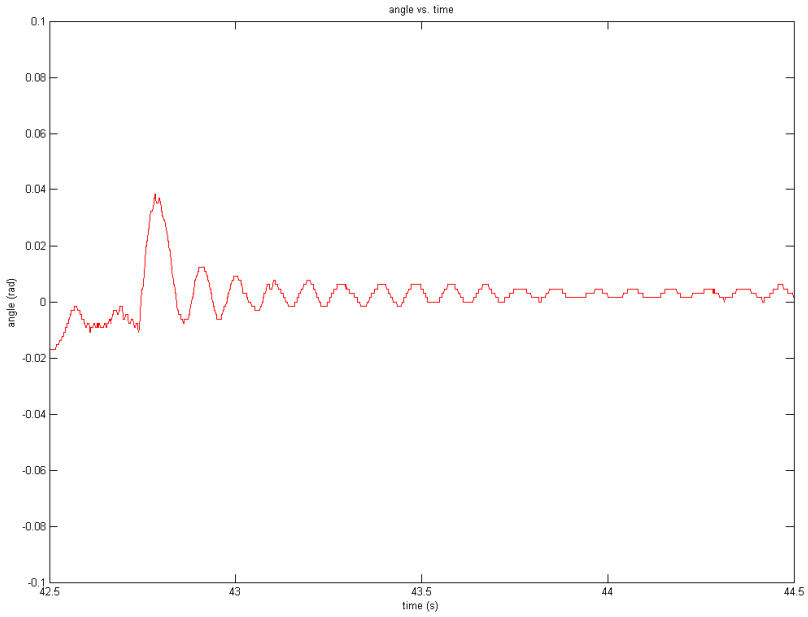
Peak time (s)	0.076	0.06
Settling time (s)	0.28	0.78
% overshoot	42 %	56 %
Final value (rad)	0.0048	0.0045
X-speed change (m/s)	~0.5 m/s	~0.3 m/s

Run 2: G(s) 750 + 6/s + 10*s, Filter cutoff = 62.8 rad/s, Integral time constant = 0.628 s

Simulation Impulse Response



Measured Impulse Response

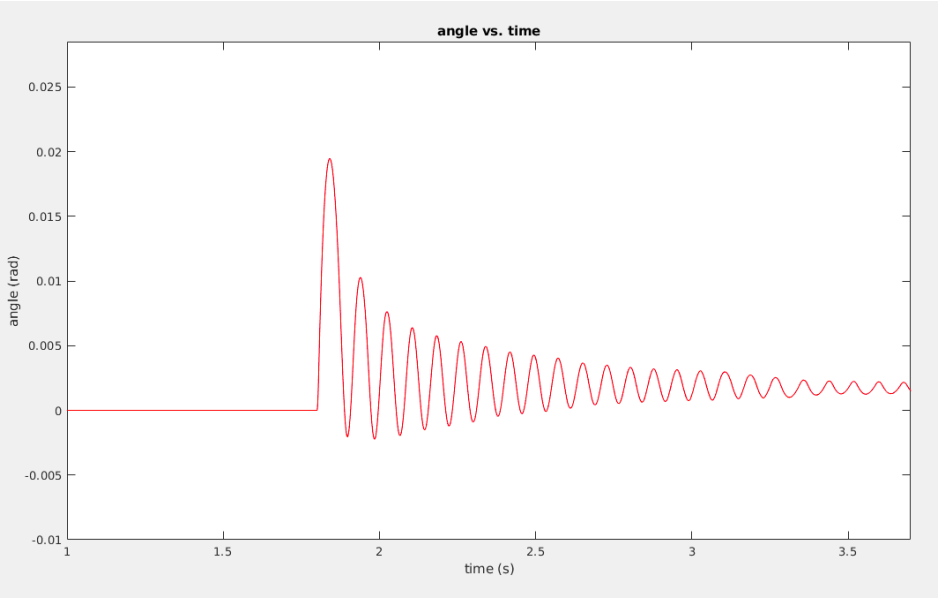


Run 2 Data:

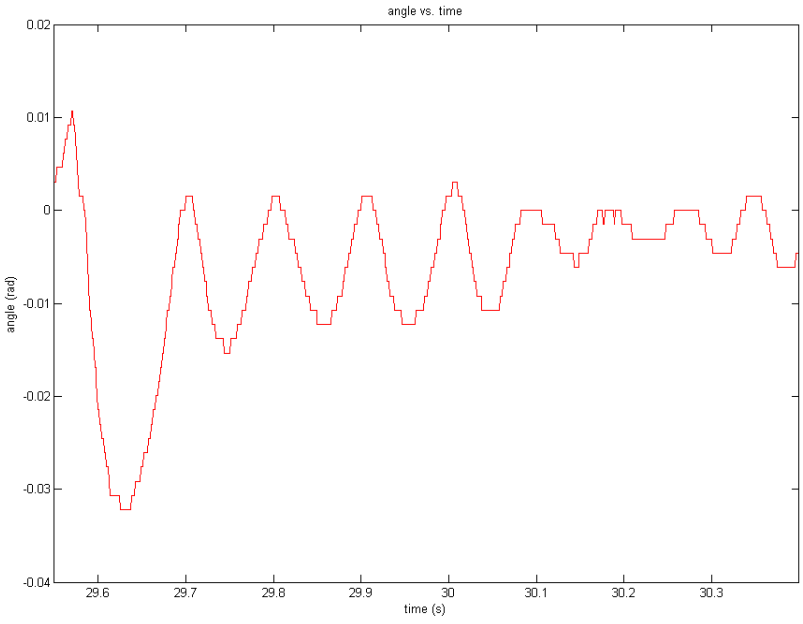
	Simulated	Measured
Natural Frequency (rad/s)	70	80
Peak time (s)	0.058	0.08
Settling time (s)	0.6	1.1
% overshoot	38 %	21 %
Final value (rad)	0.0038	0.0020
X-speed change (m/s)	~ 0.6 m/s	~0.4 m/s

Run 3: G(s) = 1200 + 20/s + 20*s, Filter cutoff = 62.8 rad/s, Integral time constant = 0.628 s

Simulation Impulse Response



Measured Impulse Response



Run 3 Data:

	Simulated	Measured
Natural Frequency (rad/s)	79	65
Peak time (s)	0.057	0.01
Settling time (s)	1.8	? long
% overshoot	21 %	23 %
Final value (rad)	0.002	0.003
X-speed change (m/s)	~ 0.6 m/s	0.5 m/s

Appendix

The Lab CLI and Model

The model used to run the controller on the lab equipment looks similar to the simulation except that the voltage is passed to a Quanser equipment interfacing bus and the angle and position are received from one. Everything in this vein is in the "inlab" folder. We don't have the lab simulink model to show as it is in the lab but we will show the lab CLI. The basic operation of the lab equipment we want when running controllers to start the model is to place the pendulum in its upright position, start the controller, and subsequently pause the controller and/or stop the model. Our CLI paired with the simulink model does shows this. The CLI code shown in Figure A9 shows this functionality. One can use the "runlab.m" script in the "inlab" folder to use this command prompt. The ENABLE_MOTOR variable is global constant in the simulink model that switches the motor voltage.

Figure: Code Used to Run controller on the Lab equipment (except simulink model)

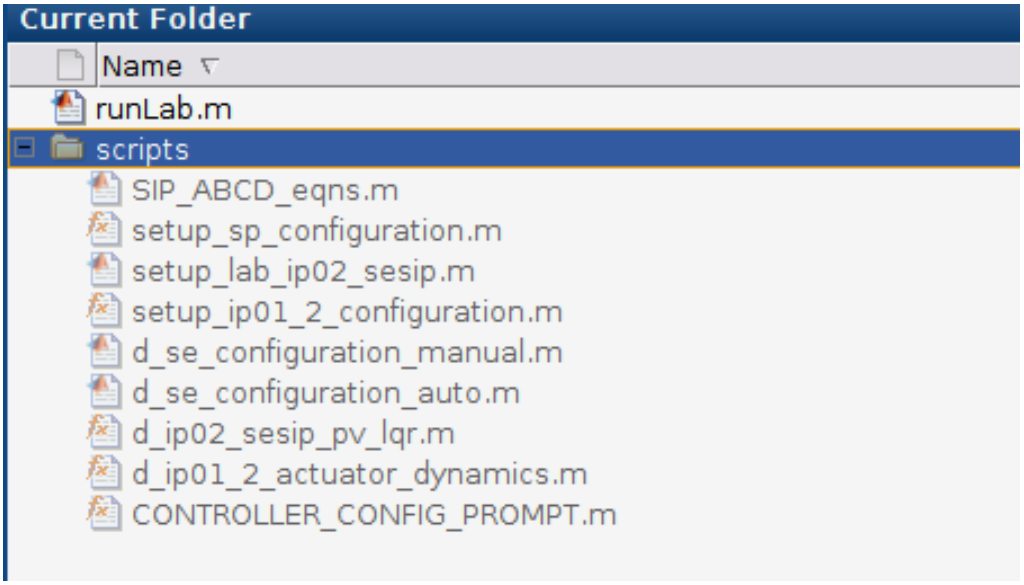


Figure A9: LAB CLI Code showing options in prompt (while model+controller are running)

```
+7  runLab.m | plottingExample.m | setup_ip01_2_configuration.m | README.txt | SETUP_SIMULATION.m | ROOT_LOCUS.m | LAB1_PROMPT
50
51 - while ~isEnded
52
53 -     loop_input = input('s->START controller, p-> PAUSE controller, e->END run: ', 's');
54 -     switch loop_input
55 -         case 's'
56 -             fprintf('CONTROLLER IS STARTED! ...\n\n');
57 -             ENABLE_MOTOR = 1;
58              %qc_update_model;
59 -             pause(1);
60 -             isPaused = 0;
61 -         case 'p'
62 -             fprintf('CONTROLLER IS PAUSED! ...\n\n');
63 -             ENABLE_MOTOR = 0;
64              %qc_update_model;
65 -             pause(1);
66 -             isPaused = 1;
67 -         case 'e'
68 -             fprintf('CONTROLLER IS STOPPED! ...\n\n');
69 -             ENABLE_MOTOR = 0;
70              %qc_update_model;
71 -             pause(1);
72 -             isEnded = 1;
73
74 -         otherwise
75 -             disp('PAUSED: Enter a valid input!');
76 -             ENABLE_MOTOR = 0;
77              %qc_update_model;
78 -             pause(1);
79 -             isPaused = 1;
```

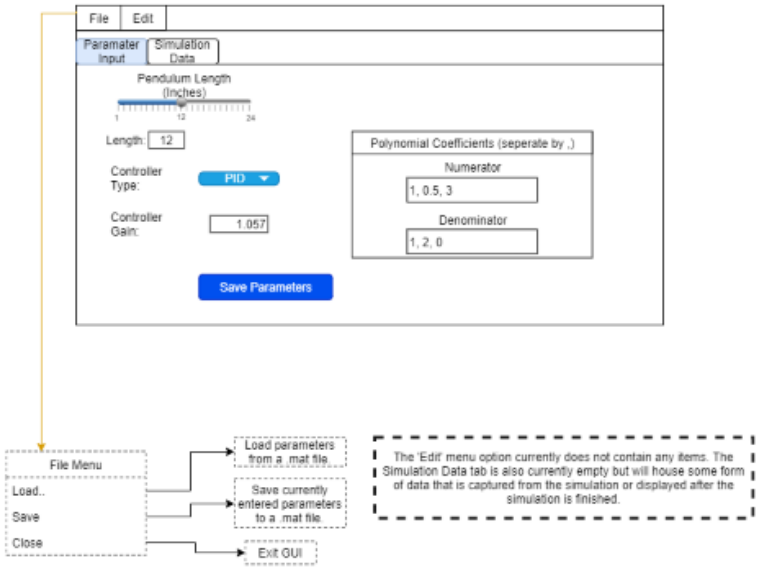
Calibration:

For calibrating friction the pendulum is taken off the cart. The method for calibrating friction is to ramp up the motor voltage until the cart starts moving. The voltage at the point the cart starts moving corresponds to the first force threshold (used in the static friction function in the simulation). At small velocities < 3 cm/s the force on the cart is approximately proportional to the voltage. The cart has less mass without the pendulum but the force threshold can be scaled by the mass ratio if static friction is assumed proportional to normal force (weight). After the cart starts moving it will get stuck 1-2 more times before moving smoothly. The first velocity threshold is approximately stationary (we use 5 mm/s).

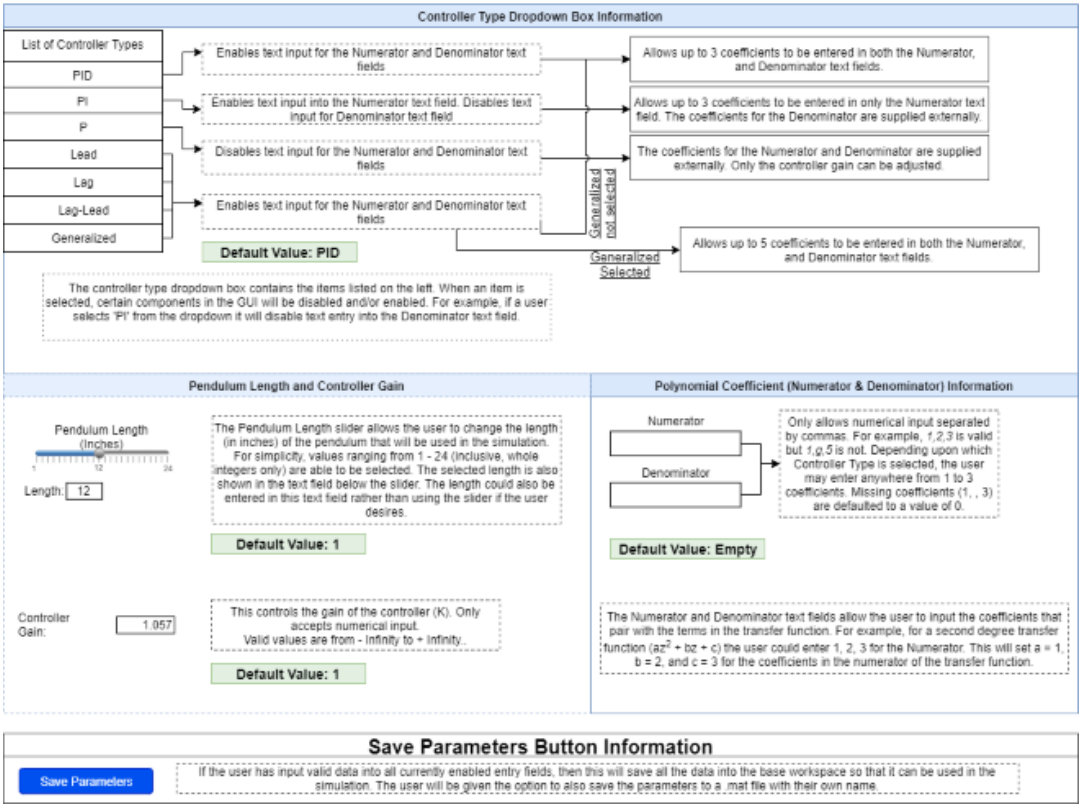
The voltage value after which the cart moves smoothly corresponds to the second force threshold. The maximum velocity before the cart get stuck the first time is the second velocity threshold. The damping coefficient is determined from the rate at which the velocity peaks and decays before getting stuck again. We have a script that automates the process the voltage ramp and the calculations for the calibration in the lab. The script stores the 5 numbers in .mat file, the calibration file, which can be stored in the simulation's calibration folder. We ran this script 5-10 times and averaged the results to get measurements.

GUI:

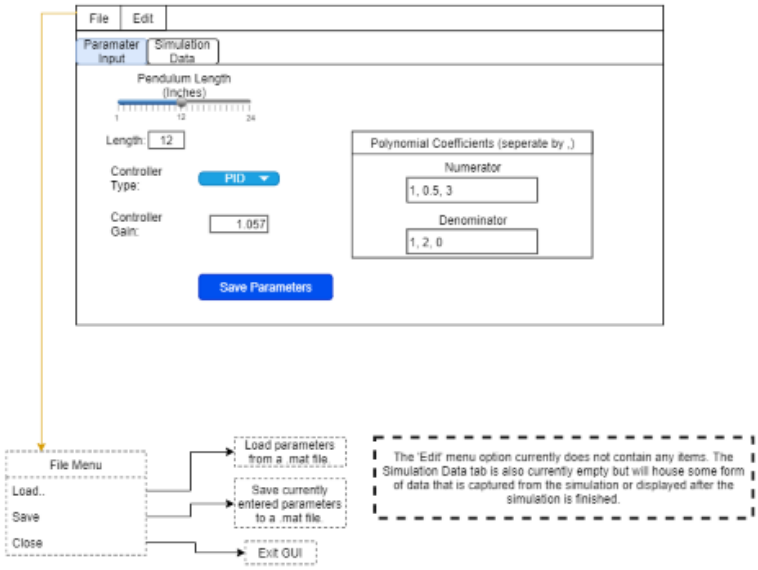
Prototype GUI Diagram



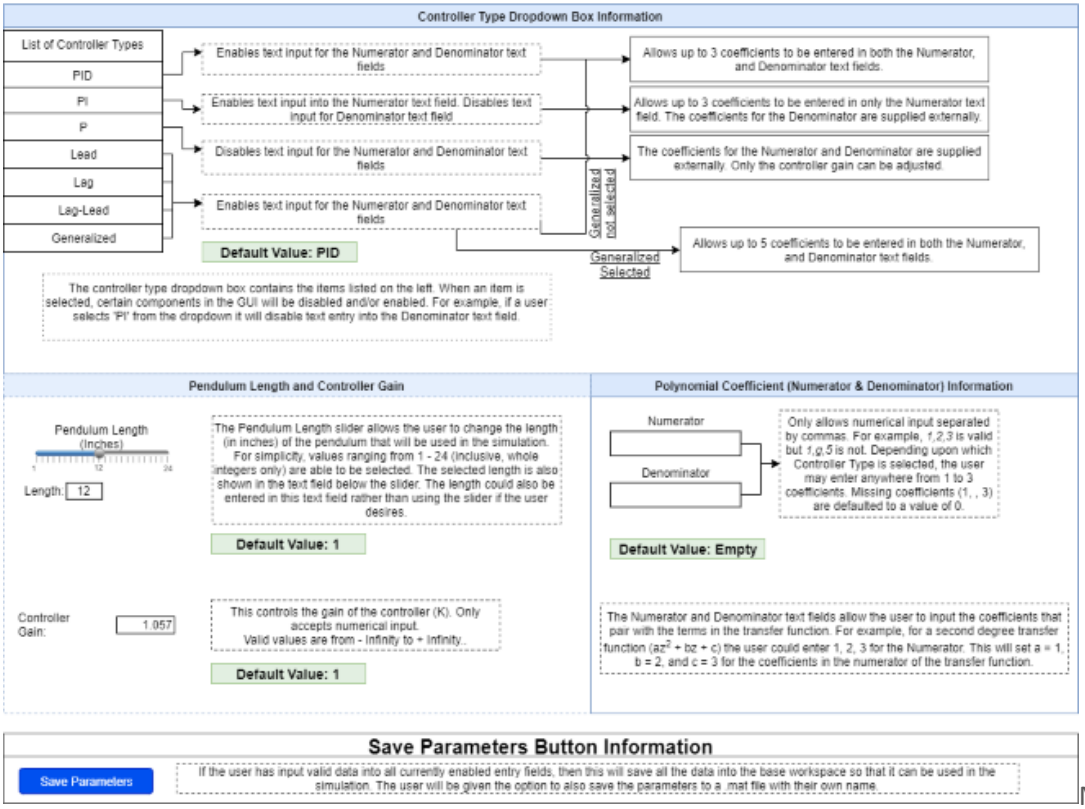
Component Details



Prototype GUI Diagram



Component Details



Maintenance:

The GUI should only require very minimal maintenance. The tool used to create the GUI is called App Designer and it is currently under active development. As such, future version of MATLAB may inherently alter, or remove certain functionality that is currently present in MATLAB 2019a. Beyond this, the only other maintenance needed will be in the way of adding any future functionality as deemed necessary.