Controlling an Inverted Pendulum using PID - ACSD Semester Project

Andy Hassun, Soomin Hyun

The University of Texas at Austin

Abstract

For our project, we decided to build and control the small “UAV motor/propeller” to control the angle of a pendulum with the propeller. To create a third order system, we implemented integral control, thus adding a 1/s to our controller. Additionally, we added proportional and derivative control. We will send a desired angle as our reference value, and track the current angle to calculate our error. Noise from the fan (vibrations from the motor) was a disturbance to our control system. We have a propeller and motor connected to an Arduino, a pendulum with a pin joint, and an accelerometer (to measure the angles in the x and z direction).

We created a dynamic system model of our system plant and linearized it. Using the transfer function that we calculated, we used routh stability, root locus analysis, and frequency domain analysis to find which values for PID tuning would stabilize the system. Due to the hardware aspect of our project being unpredictable, we used the simulation to find the PID tuning values and checked for the stability of our system after the fact.

Introduction

This project will be seeing how a pendulum’s angle can be controlled using a PID controller. We decided to use a PID controller “because of its simple structure, ease of implementation, and active research in tuning the PID for a long time” (Borase et al., 2018). We also use root locus analysis and frequency domain analysis to determine what the gain values for tuning the PID should be later in this report. Root locus analysis can be used to find the stability of a system by “[determining] what coefficients of a system characteristic equation are to be changed to achieve system stability” (Rimsky & Nesenchuk, 1996). Additionally, we decided to use frequency domain analysis because it “is used to state necessary and sufficient conditions for the existence of stabilizing controllers” (Novella et al., 2019).

Previously, a two degree of freedom, fan powered pendulum was made and controlled using a PID algorithm, proving to us that this project is feasible and within our scope. In this paper, the system is modeled and the system is simulated using PID controls, but the PID parameters are not discussed as having been found using the design and analysis methods discussed in this course. As such, we hope to expand on how to derive those parameters. Additionally, they were able to build the system as well, supporting their findings and giving us further proof of the feasibility of this project going forward (Zhi, 2018).

The pendulum itself is actuated by a propeller attached to the motor. By varying the voltage sent to the motor through PWM signals, we can control its speed, and thus control the thrust force provided to the pendulum. After closing the feedback loop with the PID controller, we were able to provide a step input of a reference or desired angle, and analyze the system’s response. Additionally, the pendulum provided an interesting modeling problem, since the portion of the force due to gravity providing a torque to the pendulum is non-linear, but always present. As such, we had to linearize this torque around the desired equilibrium point.

As we continue through this report, a critical part of our project involves the determination of the ideal gain values for tuning the PID controller. Through these theoretical frameworks and practical analyses, our aim is to find the connection between the pendulum's dynamics and the PID controller's parameters. Tuning the PID gains and ensuring a stable system involves routh stability, root locus, and frequency analysis, and was verified through simulation. Receiving results from the physical system allowed us to draw thoughtful conclusions and improved our understanding of how to utilize these techniques in future control systems. This holistic approach ensures that our project fits into the subject matter that we learned for this course. Overall, this project is a good example of many of the concepts introduced in Automatic Control Systems Design, as we will further demonstrate in the later sections.

Appreciate your attention to this intent of the project!

Project Description

We decided to simulate a PID controlled pendulum and then build a physical setup to see how our controlled system would work realistically and be able to compare results between the two setups.

We have a propeller and motor connected to an Arduino, a pendulum with a pin joint (bolt, spacer, rod), and an accelerometer. With the accelerometer, we could determine the angle of the pendulum, first by finding the amount of acceleration in the z-direction, which is 1.0g at 0°, as a function of the cosine of the pendulum angle. Then, by taking the inverse of this function, we find the current angle. To determine the sign of the angle, we find the sign of the acceleration due to gravity in the x-direction of the accelerometer. That sign was the opposite of the sign of the angle in our setup.

All of the manufactured parts in our system were designed in SOLIDWORKS. Additionally, we made the propeller, motor mount, accelerometer mount, and pendulum by 3D printing. The rest of the pendulum was built by laser-cutting acrylic. Figure 1 in Appendix A shows our schematic, and Figure 2 is a picture of what our final prototype looks like. Our final prototype needed to be put on its side with the pendulum hanging off the edge of the table due to the fans of the propeller hitting the base of the pendulum. This schematic is shown in Figure 3.

Additionally, we decided to have a settling time control requirement of 10 seconds. Thus, the control requirements to be achieved was to make sure that our system was stable, able to handle disturbances, and settle within 10 seconds.

Modeling

To begin modeling our system, we assumed that air resistance is negligible due to the low velocity of the system, and the friction in the pin joint is small compared to the torque provided by gravity and the motor. Thus, our only two forces are the force due to gravity and the thrust provided by the fan. The thrust is a function of the fan speed, which in turn is controlled by the voltage supplied to the motor. Because we could not find any equations directly relating fan speed and thrust, we placed our fan and motor on a scale, and measured the change in weight applied to the scale at various voltages, totalling ten data points. The resulting relationship between thrust and voltage was very close to linear, with an R^2 value of 0.99. Therefore, we determined the following equation:

nice!

$$F_T = 0.024V \tag{1}$$

Next, to find our state space equations, we utilized Hamilton's Canonical Equations (Ginsberg, 2008), which is a method found in graduate level dynamics courses. It involves finding the Lagrangian using generalized coordinates, using it to find generalized momentum and subsequently forming the Hamiltonian, and utilizing the Hamiltonian to form your state space equations. This method resulted in the following two equations:

$$\dot{\omega} = \frac{-F_g}{P} sin(\theta) + \frac{0.024V}{P} \tag{2}$$

$$\dot{\theta} = \omega \tag{3}$$

$$F_g = (\frac{m}{2} + M)g \tag{4}$$

$$P = (\frac{m}{3} + M)l \tag{5}$$

Due to the frequency in which the terms in equations 4 and 5 are used, they have been shortened to the constants F_g and P. Equation 2 contains sin(θ), a nonlinear element, thus requiring us to perform a tangent linearization (Ogata, 1997). Calculating the linearization around our desired equilibrium angle using the technique described in Ogata, we arrive at the following linearized equation and final state space and output equations:

$$\dot{\omega} \approx \frac{-F_g}{P} sin(\theta_o) + \frac{-F_g}{P} cos(\theta_o)(\theta - \theta_o) + \frac{0.024V}{P} \tag{6}$$

$$\begin{bmatrix} \dot{\omega} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-F_g}{P} cos(\theta_o) \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \omega \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{0.024}{P} \\ 0 \end{bmatrix} V + \begin{bmatrix} \frac{-F_g}{P} \\ 0 \end{bmatrix} C_o \tag{7}$$

$$[\theta] = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \omega \\ \theta \end{bmatrix} + [0]\, V + [0]\, C_o \tag{8}$$

$$C_o = sin(\theta_o) - \theta_o cos(\theta_o) \tag{9}$$

Again, due to the frequency in which it was used in our calculations, the expression in equation 8 was shortened to the constant C_o, which is a result of performing the linearization.The term with C_o represents the constant disturbance due to gravity. With equations 7 and 8, we can find the transfer function of our plant. We utilized equation 10 in order to find equation 11, the plant transfer function.

$$G_p = C\left[sI - A\right]^{-1} B + D \tag{10}$$

$$G_p = \frac{3}{125(Ps^2 + F_g cos(\theta_o))} \tag{11}$$

To finish modeling our system, we created a block diagram, Figure 4 in Appendix A, and found the closed loop transfer function, found in equation 12. We assume we have unity feedback due to having a direct function of measuring the angle with the accelerometer.

$$G_{cl} = \frac{3(K_d s^2 + K_p s + K_i)}{125Ps^3 + 3K_d s^2 + (3K_p + 125F_g cos(\theta_o))s + 3K_i} \tag{12}$$

We see that our closed loop transfer function has three poles and two zeros, an increase in both when compared to the two poles and no zeros in the plant transfer function. We will see this increase in poles and zeros manifest in our root locus analysis in the next section. All of the equations in this modeling were performed both by hand and with MATLAB to ensure accuracy. With our system model complete, we were able to move onto the control system analysis and design.

Although modeling the system seems simple on the surface, the addition of a nonlinear term complicates the model formulation. Had our equilibrium point been around zero, the constant disturbance due to gravity would have dropped out, vastly simplifying the model. However, there was no feasible way to set our equilibrium point as zero while maintaining a robust simulation and system. Therefore, we were forced to carry out the modeling process with that term. In doing so, we learned how to better respond to systems that are not as analogous to those we have studied in this course. This serves to improve our understanding and performance of control system design and analysis, and will better serve as engineers going forward.

As a final takeaway in the modeling process, we discovered insight into how a PID control scheme adds zeros and poles. As common knowledge, D control adds a zero and I control adds a pole. However, when I control is combined with P control, it adds both a pole and a zero, which is shown after simplifying the transfer function. In totality, PID controls adds a pole and two zeros to your closed loop transfer function. This concept was not obvious to either of us before modeling the system, finding the plant transfer function, adding PID controls, and finding the closed loop transfer function.

Control System Analysis and Design

In this control system project, a thorough analysis and tuning process were undertaken using multiple methods, including Routh stability analysis, root locus analysis, and frequency analysis through Bode plots. The Routh stability criterion was employed to determine that all PID parameters must be positive, thereby narrowing the parameter space. Root locus analysis, facilitated by MATLAB, iteratively explored the effect of PID gains on system stability, providing a practical approach to tuning within a specified range. Frequency analysis, depicted in Bode plots, established stability thresholds based on phase and gain margins. The project emphasized the need for PID controller tuning,

balancing theoretical analysis with practical experimentation to achieve desired system performance. The transition from PWM-related constants to voltage-related constants showcased the project's consideration of real-world implementation challenges.

Overall, the project offered a holistic learning experience in control system engineering. It covered theoretical concepts, practical implementation challenges, and the interdisciplinary nature of control systems. The integration of Routh stability analysis, root locus analysis, and frequency analysis provided a comprehensive understanding of system behavior, and the project's iterative tuning process highlighted the importance of both theoretical insight and hands-on experimentation in achieving stable and responsive control systems.

I. Routh Stability:

Our routh stability analysis showed that our PID tuning values should be positive. The characteristic equation that we used to find the routh table is below.

$$CE = 2l(3M + m)s^3 + 6KT_dT_is^2 + 3(2KT_i + T_ig\cos(\theta)(2M + m))s$$

Below is the routh table that we solved for.

Table 1: Routh Stability Parameters

s^3	$2l(3M + m)$	$3(2KT_i + T_ig\cos(\theta)(2M + m)$	0
s^2	$6KT_dT_is^2$	0	0
s^1	A	0	0
s^0	0	0	0

$$A = 6KT_i + 3T_igcos(\theta)(2M + m)$$

Using this routh table we found our parameters to be:

$$K > 0 \qquad T_d > 0 \qquad T_i > 0$$

Thus, from our routh stability analysis, all of our PID parameters must be positive, narrowing the region of our parameters. However, because we have three parameters to find, we must do other methods of analyses to find our tuning values.

II. Root Locus Analysis:

Due to having to find three parameters, we decided to use root locus analysis to find where our system is stable. As shown in Appendix C, Excerpt 1, we decided to use MATLAB to loop through all positive integers between 1 - 25, to start, to find where the system would be unstable. Through this, we found that this system is stable for any combination of three numbers between 1 - 25 for K_p, K_i, and K_d would allow the system to be stable. Thus, we finalized our PID tuning values using hardware. In doing so, we ended up with $K_p = 5/0.047$, $K_i = 4/0.047$, and $K_d = 0.5/0.047$, as that allowed our system to have a relatively quick settling time as well as a relatively small overshoot. The division of each value by 0.047 is due to changing our values from PWM related constants to voltage related constants.

Additionally, in Appendix A, Figure 6 shows our uncontrolled system's root locus graph. It illustrates the two poles on the y-axis each going infinitely up and down the axis. This tells us that the system when uncontrolled is marginally stable, or unstable. However, this system can be stabilized with the pole and two zeros added with the PID controller. In Appendix A, Figure 7, the root locus graph of the system with the PID controller, specified in the previous paragraph, illustrates the stability of the system with the controller as there are no poles on the right side of the graph or on the y-axis. Something interesting to note is that the root locus graph $K_p = 10$, $K_i = 10$, and $K_d = 10$ (as shown in Appendix A, Figure 8) shows the system still relatively far from instability, but our physical set up was not able to self correct after overshoot in this case. The code that was used to create these graphs can be found in Appendix C, Excerpt 2.

III. Frequency Analysis:

To have more confidence in the theoretical accuracy of the stability of our control system, we did frequency analysis by creating a bode plot of our uncontrolled and controlled systems. We set the threshold for stability as the phase margin being 30 degrees and the gain margin being 6 dB (Kim, 2017). As shown in Appendix C, Excerpt 3, we can use MATLAB to calculate the phase and gain margins using the built-in MATLAB function "margin." In doing so, we can find that our uncontrolled system is not stable as the gain margin is infinite and the phase margin is 0 degrees which is not within our threshold. However, adding the PID controller allows our phase margin to be 116.87 degrees, which lets our system be stable.

The stability of the two different systems are illustrated in our bode plots, shown in Appendix A, Figure 9. For the first bode plot (uncontrolled) has a phase curve crossing -180 degrees before the gain reaches 0 dB, suggesting instability. Additionally, peaks or sudden changes in the gain curve can indicate potential instability; this contrasts our second bode plot (controlled) that has a smooth, downward-sloping gain curve and a phase curve that remains above -180 degrees which is indicative of stability.

Simulation

We simulated what boundary conditions for our PID tuning values our system must be in with a MATLAB code that looped through each stable tuning value from 0 - 25 (divided by 0.047 due to changing PWM into a voltage value). The code for this can be found in Appendix C, excerpt 4. We were able to narrow down the range of tuning values that were now both stable and had a settling time of less than 10 seconds to stay within our control requirements. These values vary in relation to one another, so the whole range of values is not listed in this report, but can be found by running the

code mentioned above. Only the threshold was used for our physical setup to find the optimum tuning values because the optimum simulation value (where the settling time was calculated to be at its minimum value) did not match the optimum physical setup value. We believe that this was likely due to noise from the motor, though we did add damping foam and digital filters to try to minimize the error that we were getting from the high amount of vibration from the motor.

Also, we simulated what we predicted our pendulum's step response would be. This code is shown in Excerpt 5 of Appendix C, and uses the built in functions step and stepinfo to graph the step input of our PID controlled pendulum system (in the MATLAB as K_p = 5/0.047, K_i = 4/0.047, K_d = 0.5/0.047 due to changing from PWM signal to voltage). From the function stepinfo, we are able to get multiple parameters such as the rise time (0.0973 seconds) and settling time (4.9835 seconds) and more. The settling time was especially important because that was a control requirement that we had set for ourselves. We will also see later in this paper that the settling time value that we simulated was relatively accurate because, as mentioned before, our system had a significant amount of noise due to the motor's vibrations. Thus, the simulation was able to achieve this control requirement that we set, in the next paragraph we will discuss how the physical set up achieved this requirement as well.

Additionally, in Appendix A, Figure 10 has a graphical representation of the step response of our system. Through this visual, the small overshoot and stability is apparent as it reaches its steady state value relatively quickly. However, the undershoot could be minimized, so if we were to do this project again, we could adjust the controller parameters or try using a lead or lag compensator and compare the step responses to the one that we have achieved.

Results

Unfortunately, due to the large weight of the motor compared to the thrust the fan could output, the system could only achieve a maximum angle of 10°-15°. In previous studies, a brushless DC motor was used, providing a higher thrust to weight ratio (Zhi, 2018). Our system utilized a brushed DC motor, which was not able to achieve the same thrust to weight ratio. To keep our system within the bounds of our motor, we chose small angles for our reference. Additionally, the thrust from the fan was serviceable at pushing the pendulum, but the force was negligible to pull the pendulum. Therefore, we are only able to apply force to the system in one direction. Furthermore, when trying to collect angle data, we encountered a lot of noise with our accelerometer due to vibrations from the motor traveling to the accelerometer. Although we were able to filter out most of the noise and achieve a working system, some remained, which impacted our ability to show good data recorded directly from the Arduino.

In the end, our work resulted in a stable system that was able to achieve our desired angle with a low steady state error. For a step input of 5°, shown as Figure 11 in Appendix A, we settled in about 6 seconds with minimal oscillations, an overshoot of 2°, and a steady state error of less than 0.2°. When responding to a disturbance at that step input, it achieved the desired angle in about 2.5 seconds with 2.5 oscillations.

When responding to a disturbance at 0°, the pendulum came back to rest after 4 seconds and 4 oscillations. The resulting angle vs. displacement plot can be found in Figure 12 in Appendix A. Although both Figure 11 & 12 provide an overview of the system results, noise in the data collection caused both of these figures to not be entirely accurate, especially Figure 12. Because of this, the attached videos should be viewed in order to receive the full picture of the system in operation. The above time and oscillation results were gathered directly from observation of the videos, and angle vs. time data was gathered by analyzing the videos using openCV.

Conclusion

Throughout our time with this project, we gained a great deal of understanding at utilizing control systems to achieve our desired results. We fulfilled the requirements of the project as we created a stable and effective system, our system required us to interact with almost every concept we learned in this class, and applied control system design to a physical system.

Although our system possessed some inherent limitations, we were able to tune our PID constants to result in a system that reaches its desired angle in a short amount of time with low steady state error, as well as respond to disturbances. With a more powerful thrust, we could further improve our system, but we would utilize the same techniques and concepts used here. In order words, our control system design is held back by the physical limitations of our system. Improving the system would be difficult given time constraints, and the fact that the largest weight in the system is the motor itself, while the bar is about a third of the motor's weight. Most importantly, the system was stable, as it was able to settle at a step input value without wildly oscillating or increasing to infinity. In terms of an effective system, we minimized the steady state error to less than 5% of the desired angle, which we considered to be a success. The maximum overshoot could have been lower, as it was 40% of the desired angle. However, the small absolute value of 2° could mean the percentage was only that large due to the small desired angle. Most importantly to us, we achieved a settling time of less than 10 seconds, thus fulfilling our settling time requirement outlined earlier. Overall, we consider the system to be a success in terms of designing a control system for the scope of this course.

While working on this project, we modeled the system and derived a single-input, single-output transfer function, designed a PID control system,, tuned our gains with routh stability, root locus, and frequency analysis, simulated our system, and tested it physically. From beginning to end, we covered a large majority of the coursework performed in this class, minus state-feedback design. When modeling, we linearized the torque that gravity provides around our equilibrium point and created a feedback block diagram. The routh stability informed us that all our gains must be positive. With the physical system, we tried inputting a negative gain, and it resulted in an inoperable system, as predicted. With root locus, we further tuned and finalized our gains for the PID controller, and we confirmed these gains resulted in a stable system theoretically using frequency analysis. Finally, although our simulated and physical results differed slightly, the simulation still provided valuable insight into how we should expect our physical system to react. From top to bottom, this project has been a great learning experience for many different aspects of the course.

As we built a physical system, we were able to gain the experience of transferring a control system from simulation to something built in actuality. As previously mentioned, we found slight changes when transferring our system from simulation to the physical space. The settling time increased by about a second, and the maximum overshoot increased from less than 5% to 40%. These changes are likely due to forces or factors we neglected in our model, such as friction in the pin joint, air resistance, and our relationship for thrust vs. voltage occurring when the motor reaches steady state, rather than a transient response. Despite these changes, our physical system still reaches our design requirements, but additional tuning could improve its performance slightly. For larger or more complex systems, additional tuning could be completely necessary, and in that regard, this project has provided plenty of insight into why that is. In short, the physical system will always differ slightly from the simulation, but the simulation provides a very good starting point. In

Creating a control system from the start, modeling our own system and choosing gains to create a stable system, has helped both of us further understand the course material, and we are now both able to implement a PID controller to a wide variety of systems using the techniques utilized in the creation of this project. Thus, in all aspects, including making a stable and effective system, we consider this project to be a success.

References

Borase, R.P., Maghade, D.K., Sondkar, S.Y. et al. (2021). A review of PID control, tuning methods and applications. Int. J. Dynam. Control 9, 818–827 . https://doi.org/10.1007/s40435-020-00665-4

Ginsberg, J. (2008). *Engineering Dynamics*. Cambridge University Press.

Kim, S. (2017). Control of Direct Current Motors. Electric Motor Control. https://www.sciencedirect.com/science/article/pii/B9780128121382000027

Novella-Rodríguez, D.F., Cuéllar, D.M.et al. (2019). PD–PID controller for delayed systems with two unstable poles: a frequency domain approach, International Journal of Control, 92:5, 1196-1208, DOI: 10.1080/00207179.2017.1386326

Ogata, K. (1997). *Modern Control Engineering*. Prentice Hall.

Rimsky,G.V., Nesenchuk, A.A. (1996). Root-Locus Methods for Robust Control Systems Quality and Stability Investigations, IFAC Proceedings Volumes, Volume 29, Issue 1, Pages 3338-3343, ISSN 1474-6670, https://doi.org/10.1016/S1474-6670(17)58192-6.

Zhi, Q. *et al*. (2018). Design of Wind Pendulum Control System Based on Improved Genetic PID Algorithm. IOP Conf. Ser.: Mater. Sci. Eng. 439 032095

- A delight to review!
- You did everything expected!
- Nicely written, straightforward.

Excellent work!

Appendix A

Figure 1

Diagram of Pendulum Controlled by the Propeller

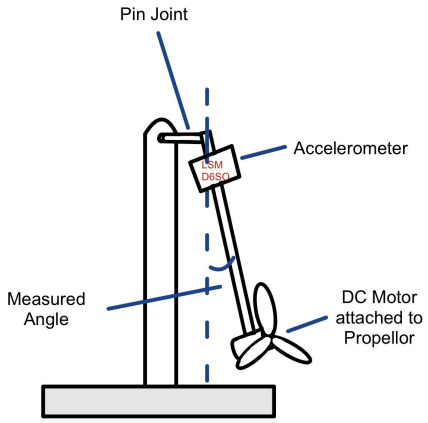


Figure 2

Picture of Final Prototype

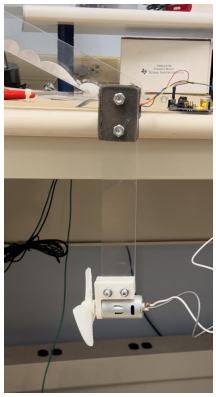


Figure 3

Schematic of Final Prototype

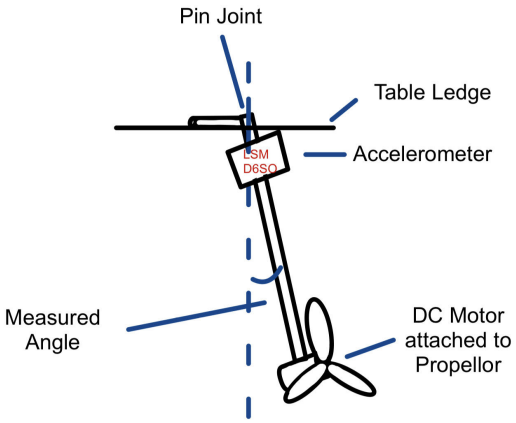


Figure 4

Block Diagram

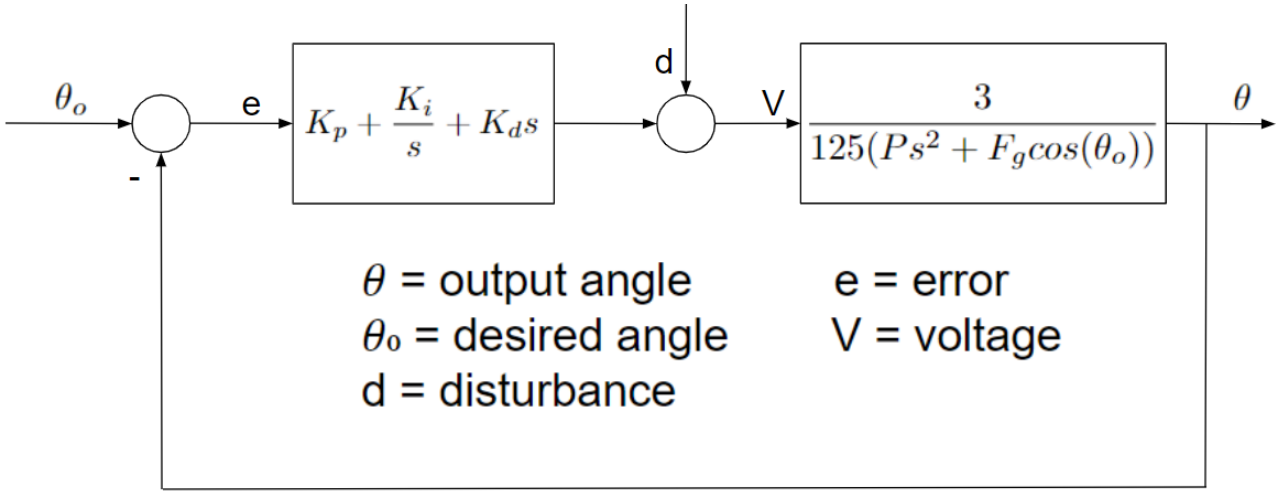


Figure 5

Relationship between fan thrust and voltage

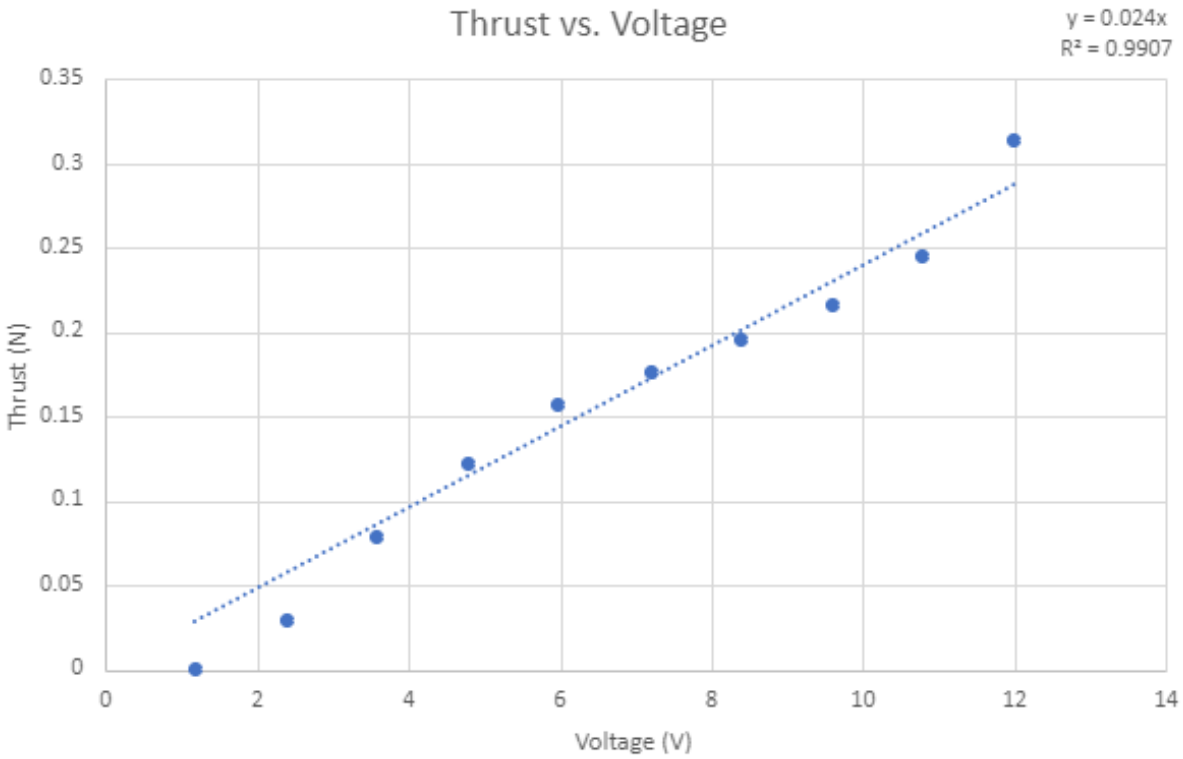


Figure 6

Root Locus of Uncontrolled System

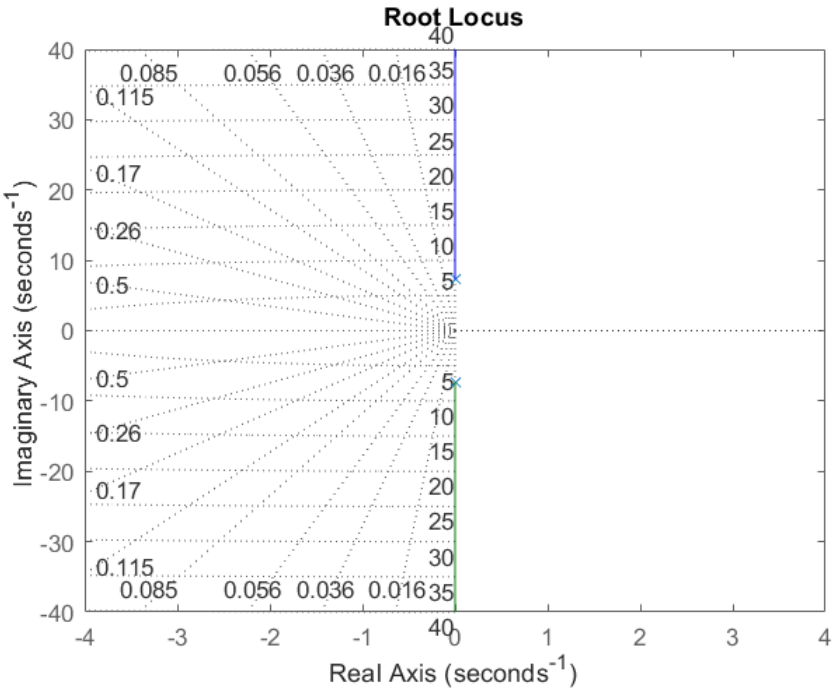


Figure 7

Root Locus of PID Controlled System (K_p = 5/0.047, K_i = 4/0.047, Kd = 0.5/0.047)

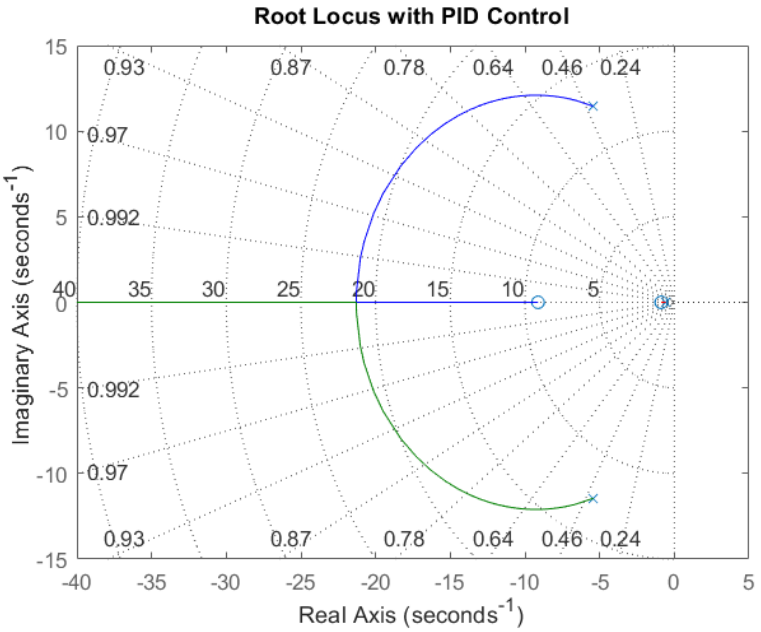


Figure 8

Root Locus of PID Controlled System (K_p = 10, K_i = 10, Kd = 10)

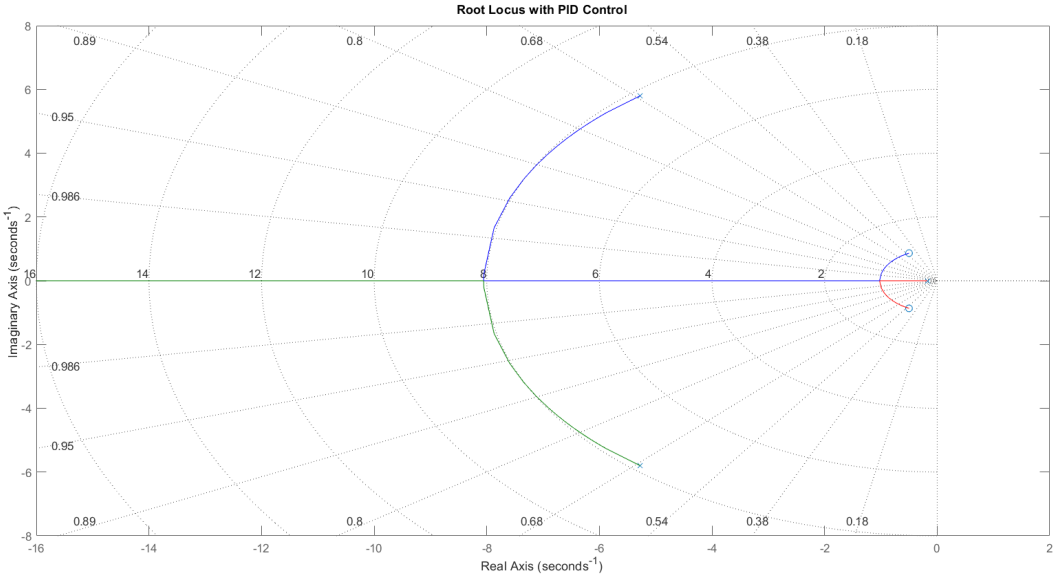


Figure 9

Bode Plot for System with and without PID Control

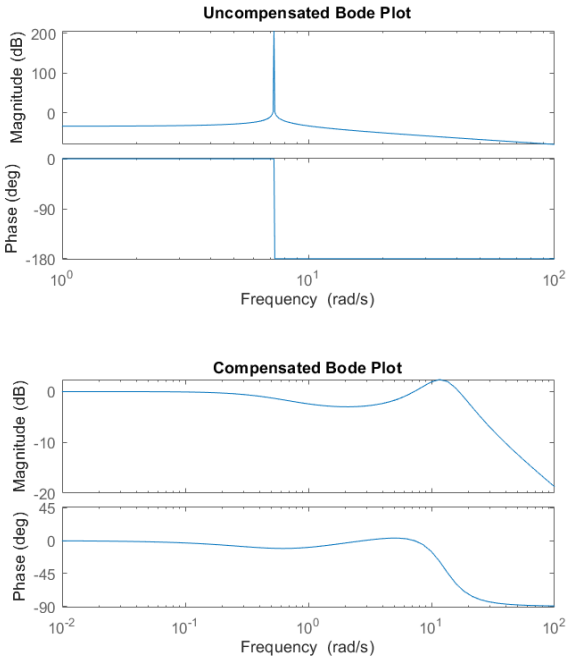


Figure 10

Controlled System Response to Step Input

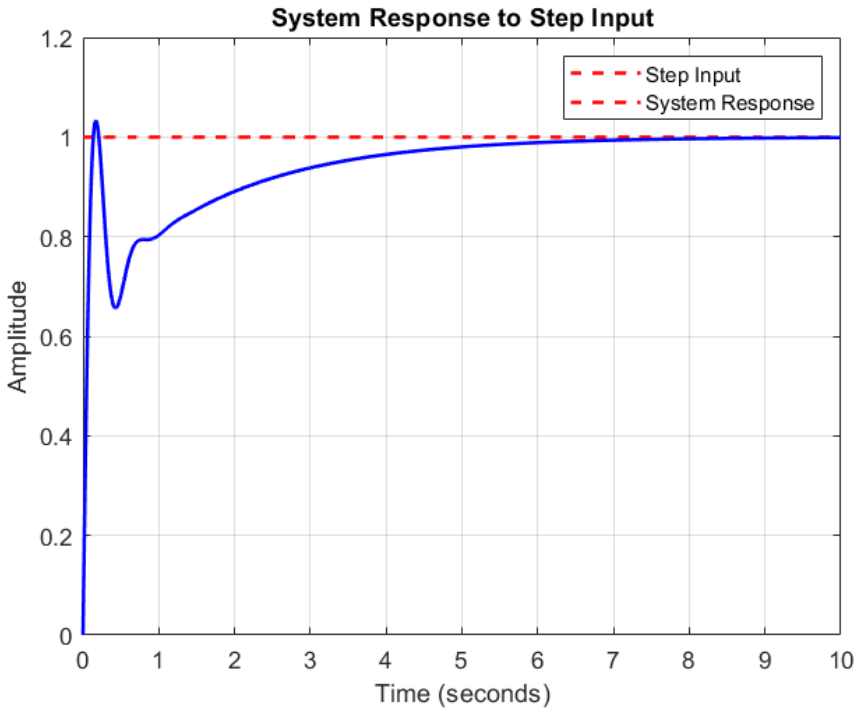


Figure 11

System results for a step input and disturbance at 10 seconds

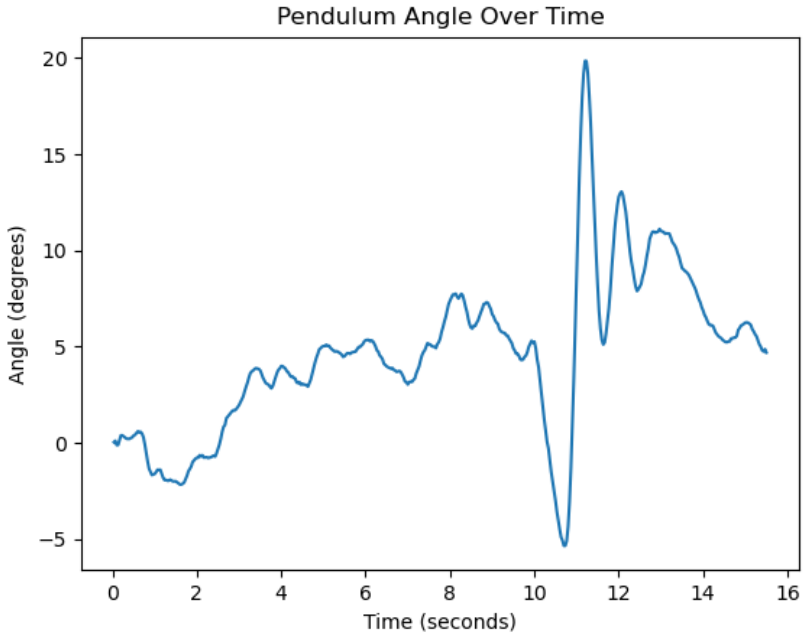
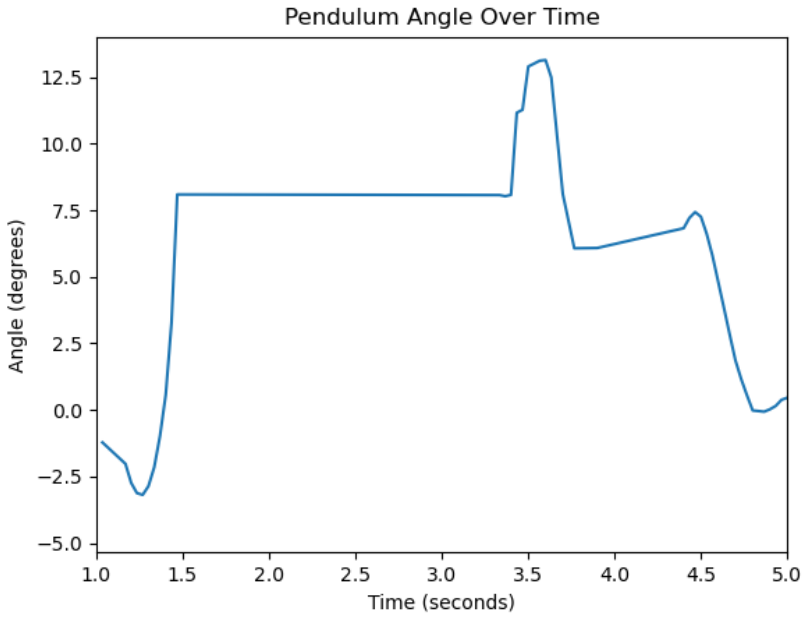


Figure 12

System results for a disturbance with a desired angle of 0°



Appendix B

Arduino Code for Built Simulation

```
#include <SparkFunLSM6DSO.h>
#include "Wire.h"

int enA = 9;
int in1 = 8;
int in2 = 7;
float dt;
float prev_time;
int des_angle; //degrees
float rdes_angle; //radians
float curr_angle; //degrees
float rcurr_angle; //radians
float raw_angle;
float C_o;
float e;
float sum_e;
float prev_e;
float sum;
float temp;
float omega;
float prev_omega;
float alpha;
float Kp;
float Ki;
float Kd;
float P;
float I;
float D;
float PID;
LSM6DSO myIMU;

void setup() {
  Serial.begin(115200);
  myIMU.begin();
  myIMU.initialize(BASIC_SETTINGS);

  dt = 1.0/1000;
  Kp = 5;
  Ki = 3;
  Kd = .5;
  des_angle = 0;
  curr_angle = 0;
  rdes_angle = des_angle*3.141/180;
  e = des_angle;
  omega = 0;
  C_o = sin(rdes_angle) - rdes_angle*cos(rdes_angle);
```

```
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  // Turn off motors - Initial state
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
}

void loop() {
  prev_time = millis();
  raw_angle = 0;
  // taking multiple readings for the accelerometer angle due to noise from motor
vibrations
  // this is the angle to determine whether the motor is on the right/left side
  sum = 0;
  for (int i = 0; i < 30; i++){
    sum += myIMU.readFloatAccelX();
  }

  // taking multiple readings for the accelerometer angle due to noise from motor
vibrations
  // this is the angle that we are setting as the current angle
  for (int i = 0; i < 10; i++){
    temp = myIMU.readFloatAccelZ() - .02;
    if (raw_angle < temp && acos(temp)*180/3.141 <= 18) {
      raw_angle = temp;
    }
  }
  // more filtering
  if (raw_angle >= 1){raw_angle = 0;}
  else{
    raw_angle = acos(raw_angle)*180/3.141;
  }
  if (raw_angle <= 18.0) {
    curr_angle = raw_angle;
  }

  // PID control calculations
  prev_omega = omega;
  omega = (e-prev_e)/dt;
  alpha = (omega-prev_omega)/dt;

  prev_e = e;
  e = des_angle - curr_angle;
  sum_e += ((e+prev_e)/2) * dt;
  sum_e = constrain(sum_e, -30, 30); // so that error_sum doesn't get too big over time

  P = Kp*e;
  I = Ki*sum_e;
```

```
  D = Kd*omega;
  PID = abs(P + I + D); // abs() because fan is only in 1 direction
  PID = constrain(PID, 30, 255);

  // Fan only rotates in 1 direction
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);

  // due to fan only rotating in 1 direction
  if (des_angle <= 0 && sum < 0) {
    PID  = 0;
  }

  analogWrite(enA, PID);

  dt = (millis() - prev_time)/1000.0;
}
```

Appendix C

Excerpt 1: MATLAB Code for Root Locus Parameters

```
close('all')
% System parameters
M = .052; % Mass
m = .136; % Additional mass
l = .23; % Length of the pendulum
g = 9.81; % Acceleration due to gravity
theta = 0; % current angle

% Given characteristic equation coefficients
A = 2*(m+3*M);
B = (1/3*m+M)*(2*l*(m+3*M));
C = (1/3*m+M)*(3*g*cos(theta)*(m+2*M));
% Characteristic equation
num = [A];
den = [B 0 C];

% Create a transfer function
G = .024* tf(num, den); % .024 is due to linearizing the force from the pwm of the motor
for Kd = 25.0:-1.0:0.0
   for Ki = 25.0:-1.0:0.0
       for Kp = 25.0:-1.0:0.0
           H = pid(Kp, Ki, Kd);
           T = feedback(G * H, 1);
           poles = pole(T);
           if all(real(poles) < 0)
               % disp('The system is stable.');
           else
               disp('the system is unstable');
               disp(['Kp: ', num2str(Kp), '     Ki: ', num2str(Ki), '     Kd: ',
num2str(Kd),]);
           end
       end
   end
end
```

Excerpt 2: MATLAB Code for Root Locus Drawings

```
%% ROOT LOCUS
close('all')
% Plant transfer function
% System parameters
M = .052; % Mass
m = .136; % Additional mass
l = .23; % Length of the pendulum
g = 9.81; % Acceleration due to gravity
theta = 0; % current angle
% Given characteristic equation coefficients
A = 2*(m+3*M);
B = (1/3*m+M)*(2*l*(m+3*M));
C = (1/3*m+M)*(3*g*cos(theta)*(m+2*M));
% Characteristic equation
num = [A];
den = [B 0 C];
% Create a transfer function
G = .024* tf(num, den); % .024 is due to linearizing the force from the pwm of the motor
figure;
rlocus(G);
grid on;
% Proportional-Integral-Derivative (PID) controller
Kp = 5/.047; Ki = 4/.047; Kd = 0.5/.047; % due to changing PWM to voltage
figure;
% Closed-loop transfer function with PID control
H = pid(Kp, Ki, Kd);
T = feedback(G * H, 1);
% Plot the root locus
rlocus(T);
grid on;
title('Root Locus with PID Control');
```

Excerpt 3: MATLAB Code for Bode Plots and Margins

```
%% MARGINS TO FIND STABILITY
close all;
% System parameters
M = 0.052; % Mass
m = 0.136; % Additional mass
l = 0.23; % Length of the pendulum
g = 9.81; % Acceleration due to gravity
theta = 0; % current angle
A = 2*(m+3*M);
B = (1/3*m+M)*(2*l*(m+3*M));
C = (1/3*m+M)*(3*g*cos(theta)*(m+2*M));
num = [A];
den = [B 0 C];
system = 0.024 * tf(num, den);
% Analyze gain and phase margins
[gm, pm] = margin(system);
% Display margins
disp(['Gain Margin: ', num2str(gm), ' dB']);
disp(['Phase Margin: ', num2str(pm), ' degrees']);
% Stability analysis based on margins
if gm > 6 && pm > 30
   disp('System is stable.');
else
   disp('System is unstable.');
end
% Step response
t = 0:0.001:5;
[y, t] = step(system, t);
% Bode plot of the open-loop system
figure;
subplot(2, 1, 1);
bode(system);
title('Uncontrolled Bode Plot');
% Define PID controller
Kp = 5/.047; Ki = 4/.047; Kd = 0.5/.047; % due to changing PWM to voltage
H = pid(Kp, Ki, Kd);
% Create closed-loop system
G = feedback(series(system, H), 1);
% Bode plot of the closed-loop system
subplot(2, 1, 2);
bode(G);
title('Controlled Bode Plot');
% Analyze gain and phase margins
[gm, pm] = margin(G);
% Display margins
disp(['Gain Margin: ', num2str(gm), ' dB']);
disp(['Phase Margin: ', num2str(pm), ' degrees']);
% Stability analysis based on margins
if gm > 6 && pm > 30
   disp('System is stable.');
else
   disp('System is unstable.');
end
```

Excerpt 4: MATLAB Code for Simulation Loop

```
close all;
% System parameters
M = 0.052; % Mass
m = 0.136; % Additional mass
l = 0.23; % Length of the pendulum
g = 9.81; % Acceleration due to gravity
theta = 5/180*pi; % current angle
A = 2*(m+3*M);
B = (1/3*m+M)*(2*l*(m+3*M));
C = (1/3*m+M)*(3*g*cos(theta)*(m+2*M));
num = [A];
den = [B 0 C];
system = 0.024 * tf(num, den);
for Kp = 25.0:-.1:0.1
   for Ki = 25.0:-.1:0.1
       for Kd = 25.0:-.1:0.1
           newKp = Kp/.047;
           newKi = Ki/.047;
           newKd = Kd/.047;
           H = pid(newKp, newKi, newKd);
           sys = feedback(series(system, H), 1);

           t = 0:0.01:10;

           y_simulated = step(sys, t);
           u = ones(1001);

           step_info = stepinfo(sys);
           settling_time = step_info.SettlingTime;
           if settling_time>10
               disp(['Kp: ', num2str(Kp), '     Ki: ', num2str(Ki), '     Kd: ',
num2str(Kd),]);
           end
       end
   end
end
```

Excerpt 5. MATLAB Code for Simulation

```
%% SIMULATION
close all;
% System parameters
M = 0.052; % Mass
m = 0.136; % Additional mass
l = 0.23; % Length of the pendulum
g = 9.81; % Acceleration due to gravity
theta = 5/180*pi; % current angle
A = 2*(m+3*M);
B = (1/3*m+M)*(2*l*(m+3*M));
C = (1/3*m+M)*(3*g*cos(theta)*(m+2*M));
num = [A];
den = [B 0 C];
system = 0.024 * tf(num, den);
Kp = 5/.047; Ki = 4/.047; Kd = 0.5/.047; % due to changing PWM to voltage
H = pid(Kp, Ki, Kd);
sys = feedback(series(system, H), 1);
t = 0:0.01:10;
y_simulated = step(sys, t);
u = ones(1001);
figure;
plot(t, u, 'r--', 'LineWidth', 1.5);
hold on;
plot(t, y_simulated, 'b-', 'LineWidth', 1.5);
title('System Response to Step Input');
xlabel('Time (seconds)');
ylabel('Amplitude');
legend('Step Input', 'System Response');
grid on;
step_info = stepinfo(sys)
settling_time = step_info.SettlingTime;
disp(['Settling Time (from stepinfo): ', num2str(settling_time), ' seconds']);
```

Appendix D

Report Authors

Report Section	Soomin Hyun	Andy Hassun
Abstract	☑	☐
Introduction	☑	☑
Project Description	☑	☑
Modeling	☐	☑
Control System Analysis and Design	☑	☐
Simulation	☑	☐
Results	☐	☑
Conclusion	☐	☑

Note: We both revised and edited the report prior to submitting. We both collaborated on all parts when creating the system, and rather the above table reflects on parts of the report written.