Design Project: Pendulum Control System

MIE404: Control Systems

Team #33

Ngoc (Christine) Bui 1001249892
Monica Burgers 1001515041
Marawan Gamal 1001370624
Frank Wen 999002298

Table of Contents

List of Figures

List of Tables

1. Design Objective

The objective for this project is to control the position of a pendulum with 1-degree of freedom. At the end of the pendulum is a fan, which can be controlled with a motor and speed controller. The arm of the pendulum is mounted on a rotary encoder, which can track the angle of the pendulum. Using these tools and a microcontroller, a control system was developed to adjust the speed of the fan rotor such that a desired pendulum angle would be achieved. The pendulum design should be able to do a step change from 0 to 30 degrees, reject externally applied disturbances, and track a sinusoidal reference oscillating between 15 to 45 degrees with a 10 second period. It is important to note that the angle will be measured relative to the vertical axis.

A closed-loop feedback system, as outlined in the block diagram in Figure 1, was implemented and tuned to accurately control the system to meet provided design criteria. To summarize, the input to the system is a desired pendulum angle, the error is the difference between the angle the pendulum is currently at and the desired angle, and the output of the designed controller is a fan rotor speed. The general idea is that the controller will determine a fan rotor speed that results in the pendulum arm hanging at the desired angle.

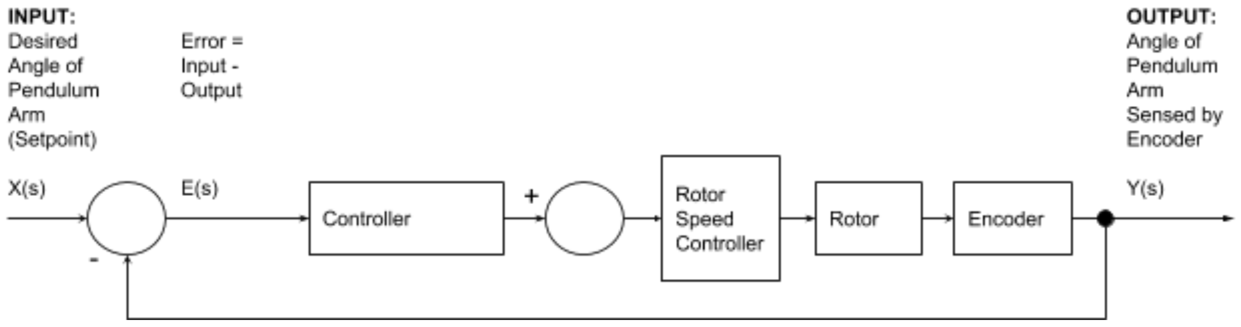

Figure 1 - Pendulum System Block Diagram

2. System Setup

The pendulum and support stand were constructed using 1/16" thick acrylic sheets. The pendulum's rod has a D-shaped hole in its center to attach it onto the D-shaped shaft of the rotary encoder. The fan is mounted onto the motor and speed controller, which are mounted on a seperate piece of acrylic connected to the pendulum with a bracket. Figure 3 shows the complete assembled system with labelled components. The circuit set-up is shown in Figure 2. The electronic components used are the following:

- Rotorgeeks 1407 Motor and Speed Controller (ESC in diagram)
- LPD3806 600BM G5 24C Optical Rotary Encoder
- Arduino UNO
- Two 1kΩ resistors
- Jumper wires

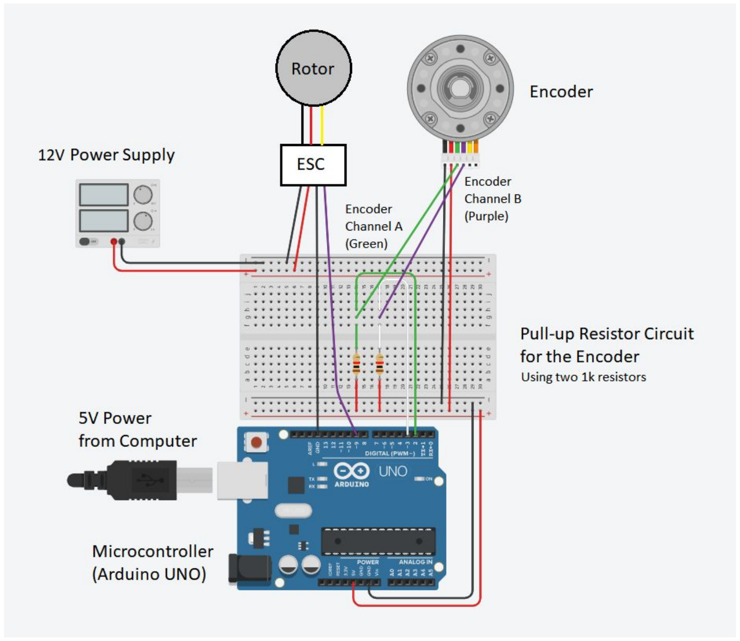
Rotor
Encoder
12V Power Supply
ESC
Encoder Channel B (Purple)
Encoder Channel A (Green)
Pull-up Resistor Circuit for the Encoder
Using two 1k resistors
5V Power from Computer
Microcontroller (Arduino UNO)

Figure 2 - Circuit Diagram of the Pendulum System

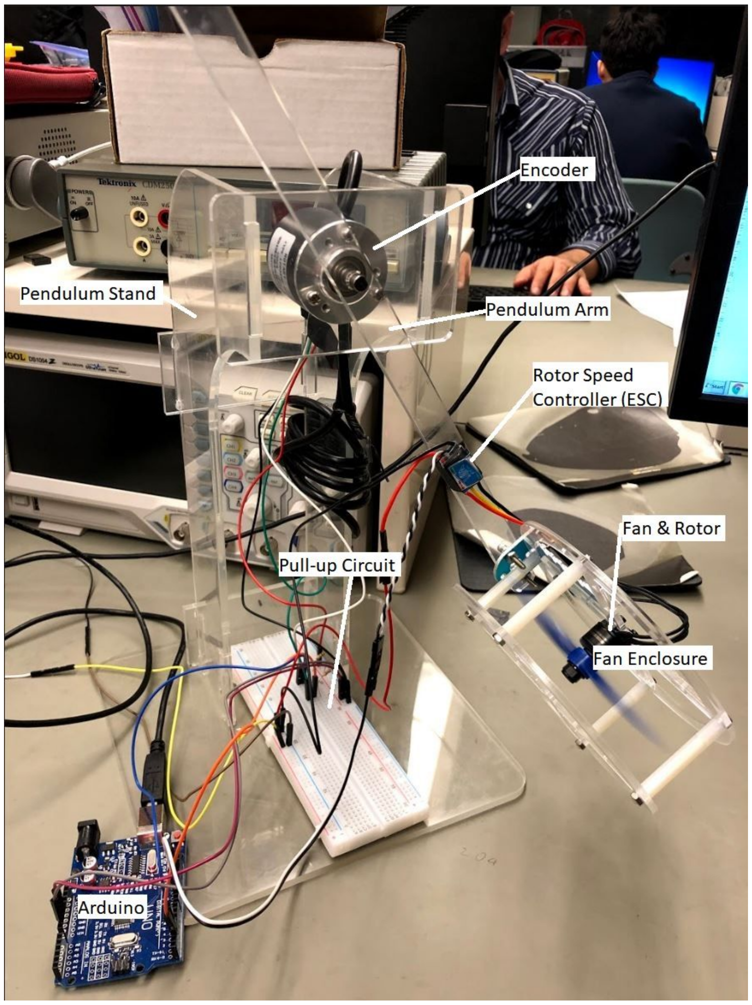

Figure 3 - Complete Electromechanical Pendulum System

3. The Control System

3.1 Controller Setup Process

A Proportional-Integral-Derivative (PID) controller was chosen because it enables control of the system response speed, the system damping, and steady-state error reduction. The design of this controller is presented in Figure 4 to demonstrate how it integrates with the pendulum system. The equation that is represented is as follows:

$$U(t) = K_p \cdot E(t) + K_i \int_o^t E(t)dt + K_d \cdot \frac{d}{dt}E(t)$$

Where U(t) is the output of the controller (in this case rotor speed), and E(t) is the input (position/angle error). K_p, K_i, K_d are the proportional, integral, and derivative gain coefficients used to tune the controller to meet the desired criteria for the 3 design aspects listed above.

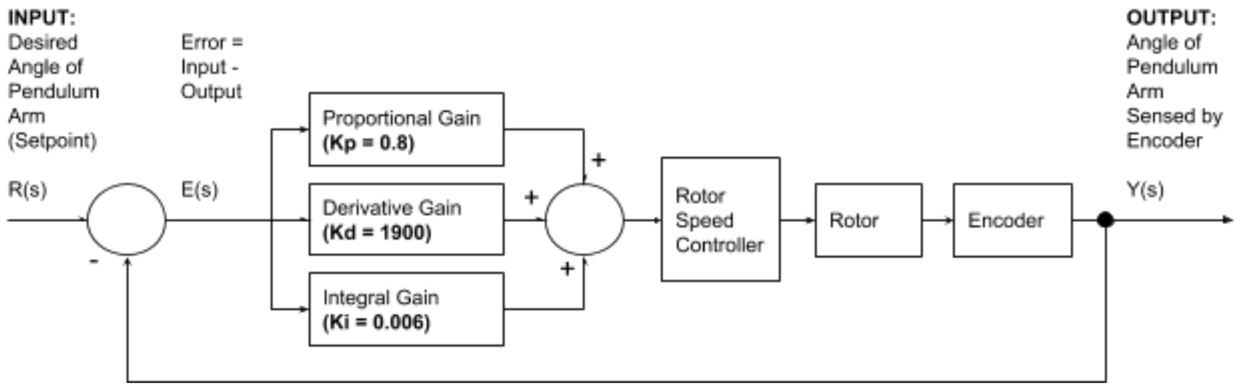
INPUT: Desired Angle of Pendulum Arm (Setpoint)
Error = Input - Output
OUTPUT: Angle of Pendulum Arm Sensed by Encoder
R(s)
E(s)
Y(s)
-
+
+
+
Proportional Gain (Kp = 0.8)
Derivative Gain (Kd = 1900)
Integral Gain (Ki = 0.006)
Rotor Speed Controller
Rotor
Encoder

Figure 4 - Pendulum System Block Diagram with PID Controller

To develop the controller algorithm, the first step is to determine the current error value, E(t). This is done by calculating the difference between the setpoint angle and the angle of the pendulum, as determined by the encoder. Using this error value, Kp multiplied by the error is the proportional term of the control output. Then, a rectangular approximation method is implemented to obtain the integral term. Afterwards, a finite difference approximation is utilized for computing the derivative term. The summation of the proportional, integral, and derivative terms forms the output of the controller, U(t). It's important to note that the control output has a certain range of limitations due to hardware capabilities. If the control output is calculated to be less than the minimum speed of the rotor motor, then the control output is set to the minimum speed value. Likewise, if the control output value is determined to be more than the maximum speed signal, the control output would be set to the maximum value.

The tuned controller gains for a setpoint of 30 degrees were found to be the following: Kp = 0.8, Kd = 1900, and Ki = 0.006.

3.2 Design Constraints

The pendulum system is required to meet the criteria outlined in Table 1. The pendulum's capacity to meet the design criteria depends on the gain factors (K's) used in the PID controller. In order to obtain the required PID parameters, the Ziegler Nichols method was used to initially tune the PID controller. However the controller was further tuned manually, due to the method not resulting in parameters fitting the needed system response characteristics.

Table 1: Design Criteria

Criteria	**Value**
Rise Time	3.95 seconds
Settling Time	6.7 seconds
Overshoot	< 5%

Rise time is defined as the time it takes for the system response to go from 10% to 90% of the setpoint. So in this particular case, the time it takes for the pendulum angle to rise from 3 to 27 degrees. The settling time is defined as how long it takes for the response to settle within +/- 2% of the setpoint. Overshoot is measured at the peak value beyond the setpoint reached by the response (as a percentage).

3.3 Tuning Methodology

The Ziegler-Nichols closed-loop ultimate sensitivity tuning method was used, as the plant contained unmodeled dynamics and had to be tuned experimentally[1].

The procedural steps made were as follows:

1. The system was brought to the specified operating point of 30°
2. All control variables except for K_P were zeroed. The value of $K_P = K_U = 2$ was found to cause sustained oscillation (ie. system was critically damped) after a slight disturbance was applied to the system.
3. The period of this oscillation, P_u (the ultimate period), was determined for the system response via reading Arduino Serial Plotter values of the encoder output. P_u was found to be 1022 ms (see Figure 5).
4. Using the values of the ultimate gain (K_u) and the ultimate period (P_u), the Ziegler-Nichols method prescribes the following values for K_c, T_I and T_D:

Table 2: Ziegler Nichols Method of Tuning for PID Controller

	K_c	T_I	T_D
PID Control General Equation	K_u*0.6	P_u/2	P_u/8
Pendulum Specific Experimental Value	1.2	0.47	0.12

This model follows the form $U(s) = K_p(1 + \frac{1}{T_I s} + T_D s)$, so from this table K_d would be K_u*T_d and K_i would be K_u/T_i [2].

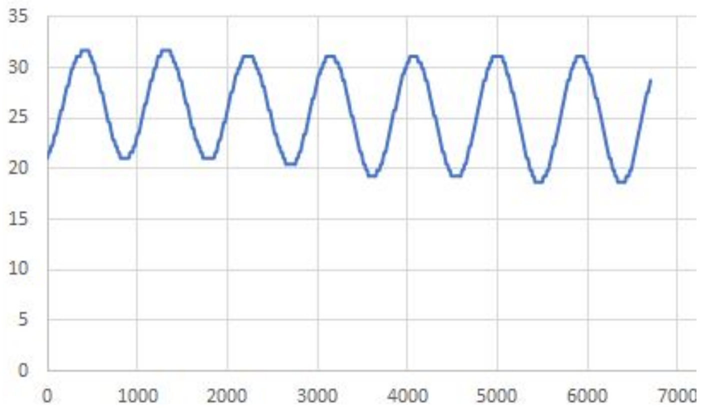

Figure 5: Sustained oscillation from Ziegler-Nichols method

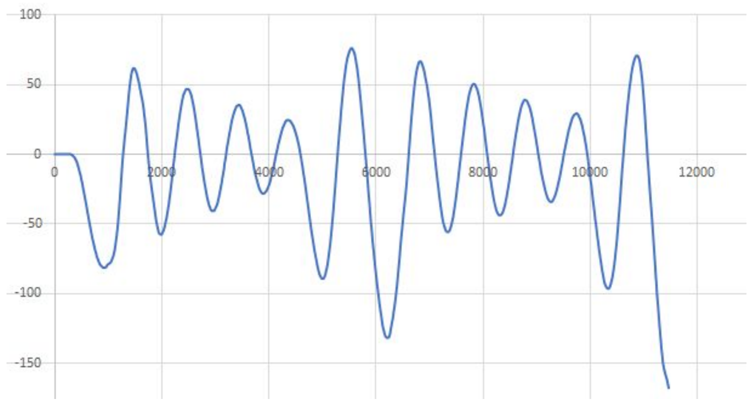

Figure 6 - Pendulum Response After Tuning with Ziegler-Nichols Method

The results achieved from the Ziegler-Nichols method resulted in an unstable system. Hence manual tuning was done with the following logic:

- If the rise time is too large and the overshoot is less than 5%, K_p is increased
- If system response has too much oscillation and/or the settling time is near the limit of 6.7s, K_d is increased
- If the overshoot is under 5%, the settling time is under 6.7s, and the system is not reaching +/-2% of setpoint within 6.7s, K_i is increased

- If rise time is insufficient after K_p adjustment, the speed offset in the control signal calculation was increased

Initially, major changes were made to the K_i and K_d values. The K_i value was too large and caused large summations in both the negative and positive directions (much greater than that of the range of the motor's speed input value of 2000). The K_d value needed to be increased drastically from the value acquired from the Ziegler-Nichols method to reduce the large overshoot seen in Figure 6. Finally, after trial and error with these approaches, we arrived at a stable system that met the design criteria in Table 1. The system response to a step input with setpoint of 30 degrees is seen in Figure 7. The tuned gain values were found to be $K_p = 0.08$, $K_i = 0.006$, and $K_d = 1900$.

3.4 Tuning Challenges & Issues

One of the challenges in tuning the controller include finding an ultimate gain value (K_u) that provides a response near steady-state conditions. Since the results were inconsistent throughout the duration of the pendulum's operation, it was difficult to identify portions of the step response that display somewhat of a stable trend. Furthermore, due to the erratic step response, pinpointing the oscillating period was also difficult. Even when both the ultimate gain and oscillating period were determined, the calculated gains (K_p, K_i, & K_d) may not necessarily be the optimal values for the controller. Consequently, the values for K_i and K_d required further tuning, which was time consuming especially when considering that any adjustments made must adhere to the previously mentioned design constraints.

Some hardware and embedded issues that were encountered included occasional irregular readings from the rotary encoder, which provided inaccurate angle values for the pendulum. Hence, the step response of the controller would be incorrect as well. Another problem was frequently not being able to tune the controller within all the constraint limits. Numerous adjustments were made that produced either a rise time or settling time exceeding the design requirements. An additional dilemma involved the propeller sometimes not functioning properly such or quickly changing speed in short successions. Moreover, the ramp function was not always conducted correctly, raising doubts on the rotor's reliability.

4. Results

4.1 Application of a Step Input

A step input was applied to the pendulum such that it moved from 0 to 30 degrees. The system response to the step input is shown in Figure 7. The response has the following traits: a rise time of 3275ms, a settling time of < 6.7s, and an overshoot of < 5% (the overshoot is 0%).

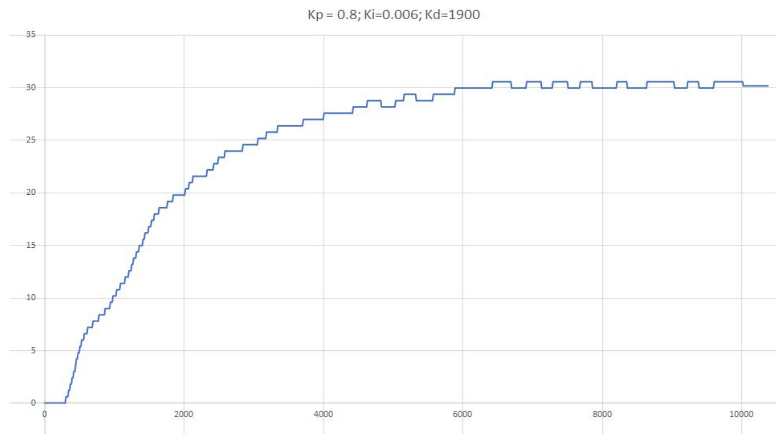

Figure 7 - System Response to Step Change after Tuning

The results are affected by having the rotary encoder to occasionally generate incorrect angle readings, which produces inaccurate control output signals. Vibrations in the pendulum and the algorithm used for the encoder may have also been a source of error.

4.2 Impact of Sampling Rate

The sampling rate is how often the controller calculates the current position error of the system and adjust the control signal (or speed of the propeller). As such, lowering the sampling time means more/faster adjustments to the control signal. Having a controller with a higher resolution than what the hardware (i.e. the rotor) can respond to would result in degraded quality of control. Testing the pendulum system at a sampling rate of 10ms was performed to verify this. The system response can be seen in Figure 8.

Likewise, increasing the sampling time decreases the number of times the controller adjusts the control signal. Having a sampling frequency that is too low results in imprecise control and increased oscillation, as seen in Figure 9 (sampling time of 100ms). A sampling time of 100ms is 5 times larger than the sampling time at which the system was tuned for control. Increased oscillation happens when the sampling time was dramatically increased because changes in the pendulum angle are not read by the controller as fast as they are happening in real life, hence control signal adjustments happen too late, allowing the pendulum arm to swing farther away from setpoint between error measurements and system adjustment.

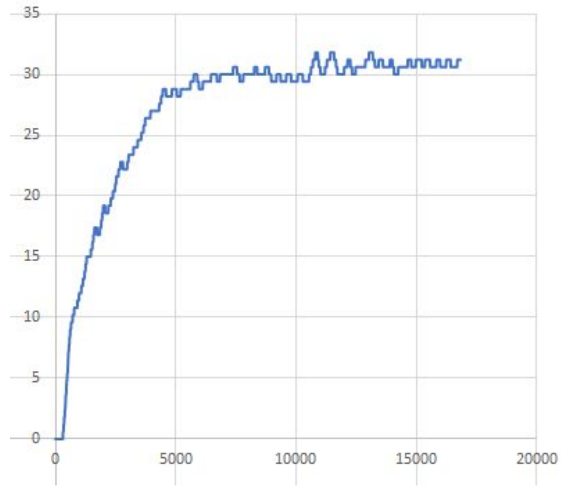

Figure 8 - System Step Response at Sampling Rate of 10ms

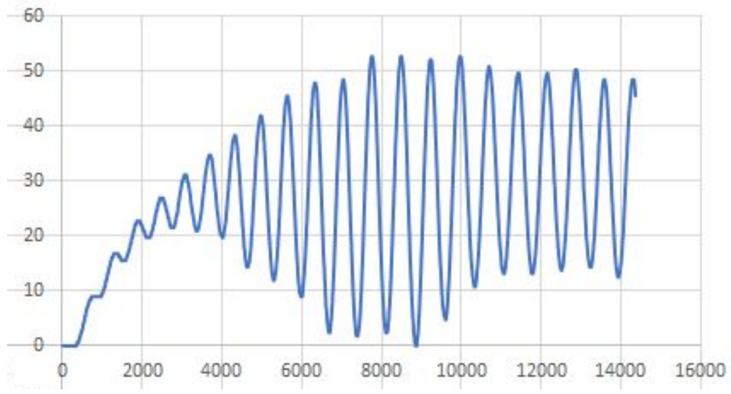

Figure 9 - System Step Response at Sampling Rate of 100ms

4.3 Regulating at 30 degrees and Rejecting Disturbances

Regulation of the pendulum arm at an angle of 30 degrees and rejecting disturbances was achieved through tuning the controller using a setpoint of 30 degrees. This means that the error calculated as an input to the controller is the difference between the encoder angle reading and 30 degrees. The supplementary video shows the system's response to an aggressive disturbance. Figure 10 shows a graph of the system response, starting from rest with the fan rotor off. The decaying oscillation after initial steady-state shows the disturbance and resulting system response.

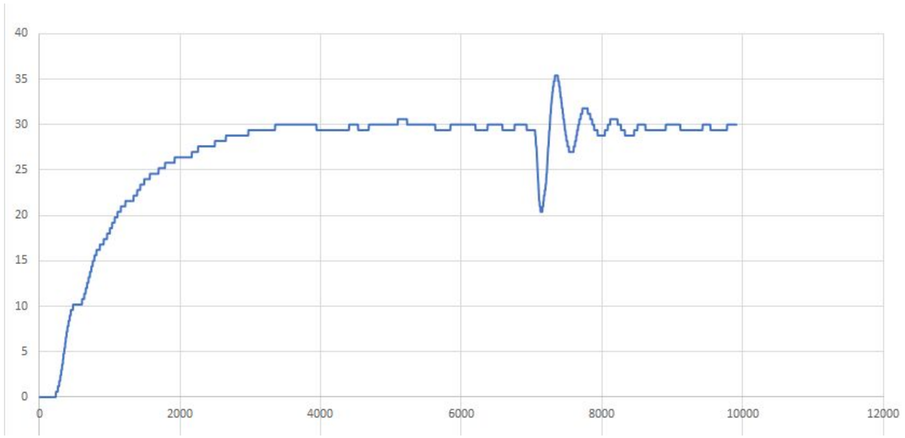

Figure 10 - Regulating Pendulum Angle at 30 Degrees and Rejecting Disturbances

4.4 Tracking a Sinusoidal Reference

In order to track a sinusoidal reference with a 10s period oscillating between 15 and 45 degrees, the following equation was used to define the setpoint angle:

$$setpoint\_angle \ = \ 15 * sin(\frac{2*\pi*t}{10000}) + 30$$

The result of the controller and pendulum tracking the sine wave input is shown in Figure 11. An error can be observed, having an overshoot in either direction of the trace. This error is likely due to imperfections in the PID tuning, disturbances from the surrounding environment, and the hardware limitation discussed earlier in section 3.4.

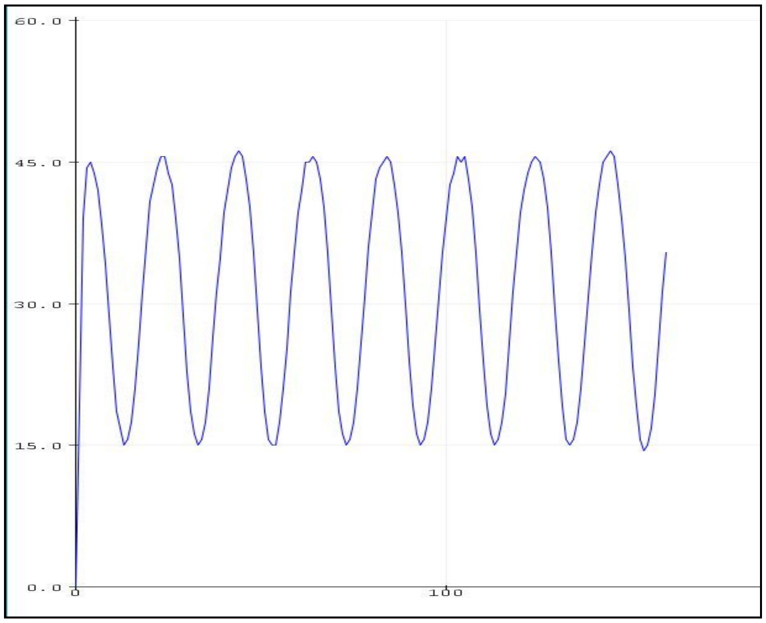

Figure 11 - Tracking a Sinusoidal Reference

5. Conclusion

The goal of this project was to design and implement a control system to achieve several specific system response criteria. The pendulum had to be able to accurately perform a step change from 0 to 30 degrees under a specified time, as well as reject any disturbances applied such that the pendulum always returns to the 30 degree setpoint. The system also had to be able to track a sine wave input, oscillating the pendulum between angles of 15 and 45 degrees. Specifically, the system was required to achieve these responses with a rise time of 3.95 seconds, a settling time of 6.7 seconds, and an overshoot of less than 5%. In order for the pendulum system to accomplish these tasks, a PID controller was developed and tuned. The Ziegler Nichols method was used to determine ballpark gain (K) values for the controller. Further trial-and-error adjustments were made based on knowledge learned from class about how each of a P-, PI-, and PD- controller adjust the response of a closed-loop system. The resulting controller function that met the design criteria was found to be

$$U(t) = 0.08 \cdot E(t) + 0.006\int_{o}^{t} E(t)dt + 1900 \cdot \tfrac{d}{dt}E(t)$$

Regarding the impact of the sampling time, it was learned that both increasing and decreasing this parameter could have adverse effects on system response. If the sampling time is too fast, the system behaves more erratically and does not reach an accurate steady-state point due to hardware not being able to keep up with the rapidly changing control signal. If the sampling time is too slow, the system response becomes more oscillatory due to the control signal not being adjusted quickly enough to accurately keep track of the pendulum position.

Overall, the PID controller was able to perform well once the gain values were properly tuned. The designed control system effectively controlled the pendulum and met the given design criteria.

6. References

[1] F. Haugen, *PID Control*. 2005, pp.89-98. [Online] Available: http://techteach.no/fag/tmpp250/v06/pidcontrol/pid_tuning.pdf

[2] G.F. Franklin, J.D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*. 8th ed., New York: Pearson, 2019, pp 216-222.

7. Appendices

Appendix A - Microcontroller Code

```
#include "ESC.h"

//---------------------ESC SETTINGS-----------------------//
#define SPEED_MIN (1000)                    // Set the Minimum Speed Setting of the ESC
#define SPEED_MAX (2000)                    // Set the Maximum Speed Setting of the ESC

ESC myESC (9, SPEED_MIN, SPEED_MAX, 500);             // Initialize ESC
int oESC;                              // Variable for the speed sent to the ESC



//---------------------ENCODER PORTS----------------------//
#define outputA 2                        // Define Encoder output A to Digital Pin 6 on
Arduino Uno
#define outputB 3                        // Define Encoder output B to Digital Pin 7 on
Arduino Uno
#define NUMPULSES 1200                         // Define the number of pulses on Encoder
resolution (enter twice the amount of pulses on datasheet)

//---------------------ENCODER VARIABLES------------------//
float counter = 0;                       // Counter variable for the number of pulses from the
encoder
volatile byte reading = 0;
volatile byte aFlag = 0;
volatile byte bFlag = 0;
double encoderPos = 0;
double oldEncPos = 0;
double angle = 0;

//---------------------TIMER VARIABLES--------------------//
long t_now;
long t_last_print = 0;
long t_last_PID = 0;
int T_sample = 20;                        // sample time in milliseconds (ms)
int T_print = 10;                        // sample print to monitor time in milliseconds (ms)
// 3s = 300
// 6s = 600

//---------------------PID VARIABLES----------------------//
double error = 0;
double errSum = 0;
double dErr = 0;
double lastErr = 0;

int max_control = 1700;
int min_control = 1000;

long control_signal = 1000;
long nowSpeed = SPEED_MIN;
long lastSpeed = SPEED_MIN;

// ================== DESIRED SETPOINT ANGLE================== //
double setpoint_angle = 30;
// ================== CONTROL GAINS===========
double Kp = 0.8;                    // proportional gain
double Ki = 0.007;                   // integral gain in [ms^-1]
double Kd = 1800;                    // derivative gain in [ms]
//Original Values: 0.8, 0.003, 120

void setup() {
  pinMode(outputA, INPUT_PULLUP);
  pinMode(outputB, INPUT_PULLUP);
  pinMode(9, OUTPUT);

  Serial.begin(9600);

  Serial.println("Wait");
  myESC.arm();                       // Send the Arm value so the ESC will be ready
  delay(1000);                      // Wait for a while
  Serial.println("Begin");
  rampUpDown();
  delay(1000);

  attachInterrupt(digitalPinToInterrupt(outputA), PinA, RISING);
  attachInterrupt(digitalPinToInterrupt(outputB) ,PinB ,RISING);

  t_last_PID = millis();
}

void loop() {
  angle = (encoderPos/NUMPULSES)*720;
  //setpoint_angle = 15*sin(2*3.14*0.0001*t_now) + 30; // for tracking sinusoid
  detachInterrupt(digitalPinToInterrupt(outputA));
  detachInterrupt(digitalPinToInterrupt(outputB));
  PID_control();
  attachInterrupt(digitalPinToInterrupt(outputA), PinA, RISING);
  attachInterrupt(digitalPinToInterrupt(outputB), PinB, RISING);
  myESC.speed(control_signal);
  print_results();
}

void rampUpDown() {

  for (oESC = SPEED_MIN; oESC <= 1150; oESC += 1) {
    myESC.speed(oESC);                              // write speed setting to motor
    delay(20);                                  // waits 10ms for the ESC to reach speed
  }
  delay(2000);                                  // wait a while

  for (oESC = 1150; oESC >= SPEED_MIN; oESC -= 1) {        // iterate from speed setting of
1150 to minimum speed
    myESC.speed(oESC);                              // write speed setting to motor
    delay(20);                                  // waits 10ms for the ESC to reach speed
   }
  delay(2000);                                  // Wait for a while before going into control loop
}

void rampUpTo(double target) {
  for (oESC = lastSpeed; oESC <= target; oESC += 1) {     // iterate from minimum speed to a
speed setting of 1150
    myESC.speed(oESC);                             // write speed setting to motor
  }
  lastSpeed = target;
}

void PID_control(){
  t_now = millis();
  if(t_now - t_last_PID > T_sample){
    // Proportional Control
    error = setpoint_angle - abs(angle);

    // Integral control
    errSum += error*T_sample;

    // Derivative control
    dErr = (error - lastErr)/T_sample;

    // PID
    control_signal = Kp*error + Ki*errSum + Kd*dErr + 1000;

    // Limit control_signal within bounds
    if(control_signal > max_control){
      control_signal = max_control;
    }

    // Update values
    lastErr = error;
    t_last_PID = t_now;
  }
}

void PinA() {
  cli();
  reading = PIND & 0xC;
  if(reading == B00001100 && aFlag) {
    encoderPos --;
    bFlag = 0;
    aFlag = 0;
  }
  else if (reading == B00000100){
    bFlag = 1;
  }
  sei();
}

void PinB() {
  cli();
  reading = PIND & 0xC;
  if(reading == B00001100 && bFlag) {
    encoderPos ++;
    bFlag = 0;
    aFlag = 0;
  }
  else if (reading == B00001000){
    aFlag = 1;
  }
  sei();
}

void print_results() {
    t_now = millis();

  if (t_now - t_last_print >= T_print){
    t_last_print = t_now;
    Serial.print(30);
    Serial.print(" ");
    Serial.print(30*1.05); // overshoot
    Serial.print(" ");
    Serial.print(30*1.02); // +ve settling boundary
    Serial.print(" ");
    Serial.print(30-0.6); // -ve settling boundary
    Serial.print(" ");
    Serial.println(abs(angle));
  }
}
```

Appendix B - Attribution Table

RD – wrote first draft
MR – major revision
ET – edited for grammar and spelling
FP – final read through of complete document for flow and consistency
CM – responsible for compiling the elements into the complete document
EC – troubleshoot and edit code
X – responsible

	Ngoc	**Monica**	**Marawan**	**Frank**
Section 1	RD, MR	MR, ET, CM		ET
Section 2		RD, ET, CM		ET
Section 3	RD	RD, MR, ET, CM	MR	MR, ET
Section 4	RD, MR	MR, ET, CM	MR	MR, ET
Section 5	RD	ET, CM	MR	MR, ET
Encoder Code	RD, CM	RD	RD	EC
PID Code	RD, CM	MR	MR	EC
PID Tuning	X	X	X	X
Pendulum Fabrication	X			