University of Toronto
Faculty of Applied Science & Engineering

**MIE404: Control Systems**

# Design Project

**December 4th, 2018**

Group 12:
Yuanshan Du 1001324541
Hongzheng Xu 1001772904
Yixiao Hong 1001311145
Stephen Huang 1001165099

# Table of Contents

# Project Description

The objective of this project is to construct a control system prototype that can grant desired output and also can resist disturbance. In this project, a pendulum is built with a propeller and an encoder, the desired output is a stable angle or path the pendulum should follow. The pendulum should reach the desired output quick and accurate enough and also should minimize overshoot and resist disturbance.

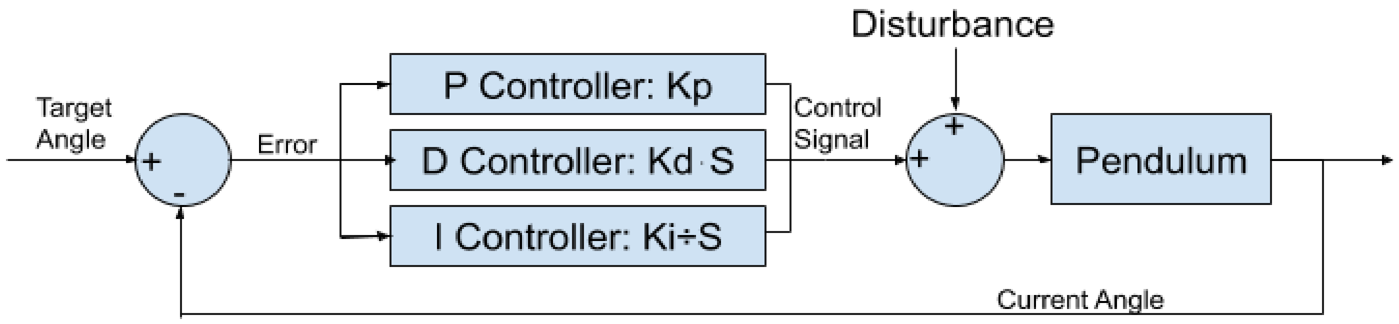

*Figure 1:Block Diagram of the System*

The encoder and the propeller motor are both powered by 13V DC and connected to an Arduino. Control signal is sent to propeller motor through ESC and feedback signal is collected from encoder.

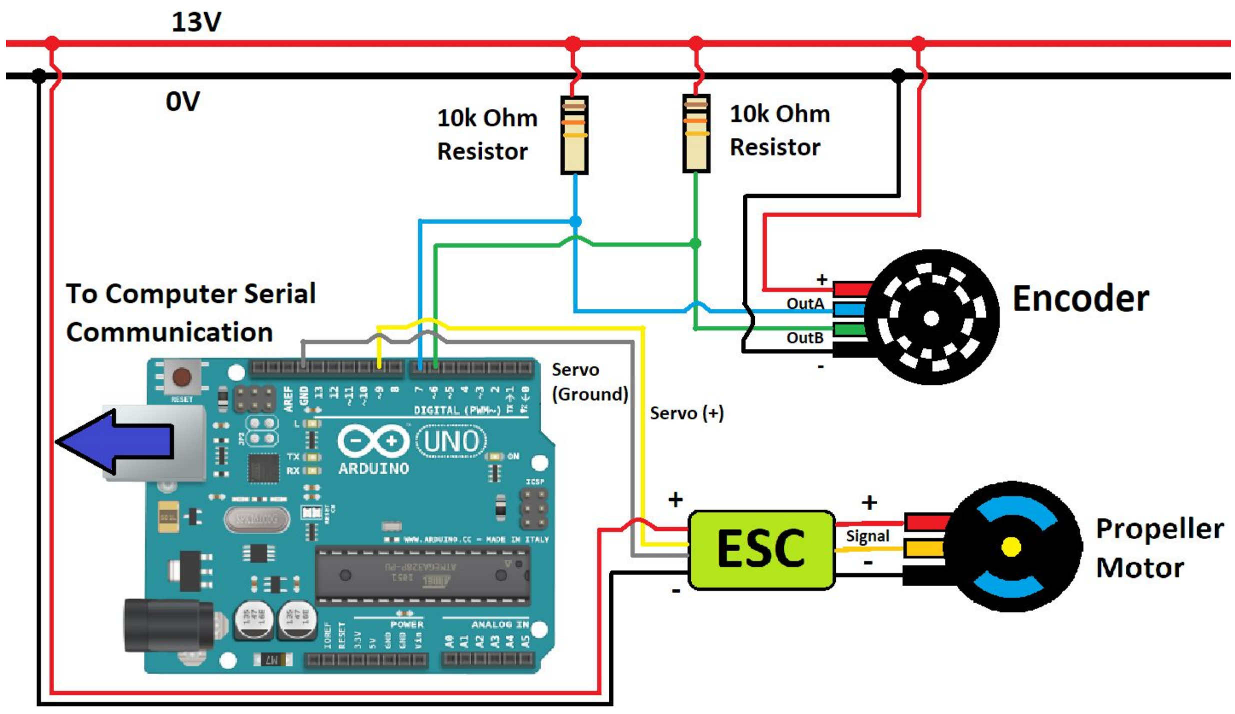

*Figure 2: Wire Connection of the System*

# Controller Definition/Analysis

**Controller Selection:**

For this project, the design requirement states that the tuned system should achieve, rise time < 3.95s , settling time < 6.7s and overshoot < 5%.

In order to achieve all the design requirements and obtain desired system, the PID controller is selected. The PID controller can meet the design goal and comprehend the negative effects caused by the proportional controller and integral controller. The proportional controller, Kp, can improve the response time but cause large steady-state error and instability as it increase. The integral controller, Ki, can eliminate steady state error but lead to instability as the value increases; and the derivative controller, Kd, is used to improve the system response, ability to resist disturbance and reduce overshoot. Derivative control, Kd, is crucial for further improvement of the system response and neutralize the negative effect caused by the other two controllers.

## Controller Methodology and Challenges/Issues:

Arduino Uno is used in the project to control the motor speed through ESC. Since the microcontroller can only acquire system error from encoder with a specific sampling frequency, the team applied numerical method to simulate a PID control. As shown in figure 3, the error is acquired and calculated per time increment of $\Delta t = (t2 - t1)$ , which is proportional to the Kp control. The differentiation of error can be calculated as: $D_{error}/D_t = (Error_2 - Error_1)/(t_2 - t_1)$ which is proportional to the output of Kd control. The integration of error at t2 can be calculated as the area in the yellow region in the diagram, which is proportional to the output of Ki control. Theoretically, the smaller the sample time applied, the more accurate results will get from the numerical method. However, due to the limitation of the microprocessor and the encoder, the optimum sampling time is set to be 15-25ms and 20ms is implemented in the experiment.

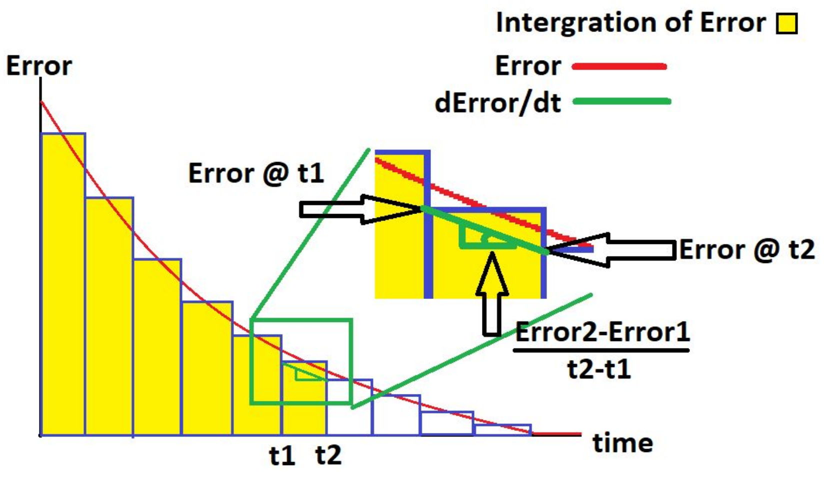

*Figure 3. Numerical Method of Calculating Kp/Ki/Kd*

A slow sampling rate will result in system instability as the error will not be detected quick enough and a much higher gain than its actual need will be assigned. A quick sampling rate will result in long response time and oscillation as the high frequency noise will often be detected and more unnecessary control signal will be assigned. Please refer to Test 2 for more details about different  sampling rates.

To find the optimum Kp, Ki, Kd value for the PID controller, the team initially tried "Ultimate Cycle Method " since the pendulum system is an unknown plant. The team initially set Ki & Kd to be zero and tried to increase the Kp gain until the output begins to react as marginally stable with a step disturbance. However, the team found that the encoder became very unreliable if the rate of angle change is high. Without Ki and Kd, the pendulum will rise with a very high speed and oscillate severely which resulted in error of reading the angle. Thus, the team has to put a very small Ki value (Ki=0.002) to reduce the initial angle change rate and then apply a step disturbance to see if the output is marginally stable. The Kpu value is found out to be 3.85 and the oscillation period (Pu) is measured as 0.9s (Figure4). According to "Ultimate Cycle Method ", for Kpu=3.8 and Pu=0.9, the corresponding Ki and Kd is calculated as:

$K_p = 0.6Kpu = 0.6 * 3.8 = 2.28$ , $T_i = \frac{P_u}{2} = \frac{0.9}{2} = 0.45$ ,  $T_D = \frac{P_u}{8} = \frac{0.9}{8} = 0.11$ ,
$G_c = K_p(1 + \frac{1}{T_i s} + T_d s)$ ,Thus the PID is calculated to be Kp=2.28, Ki=5.1, Kd=0.26

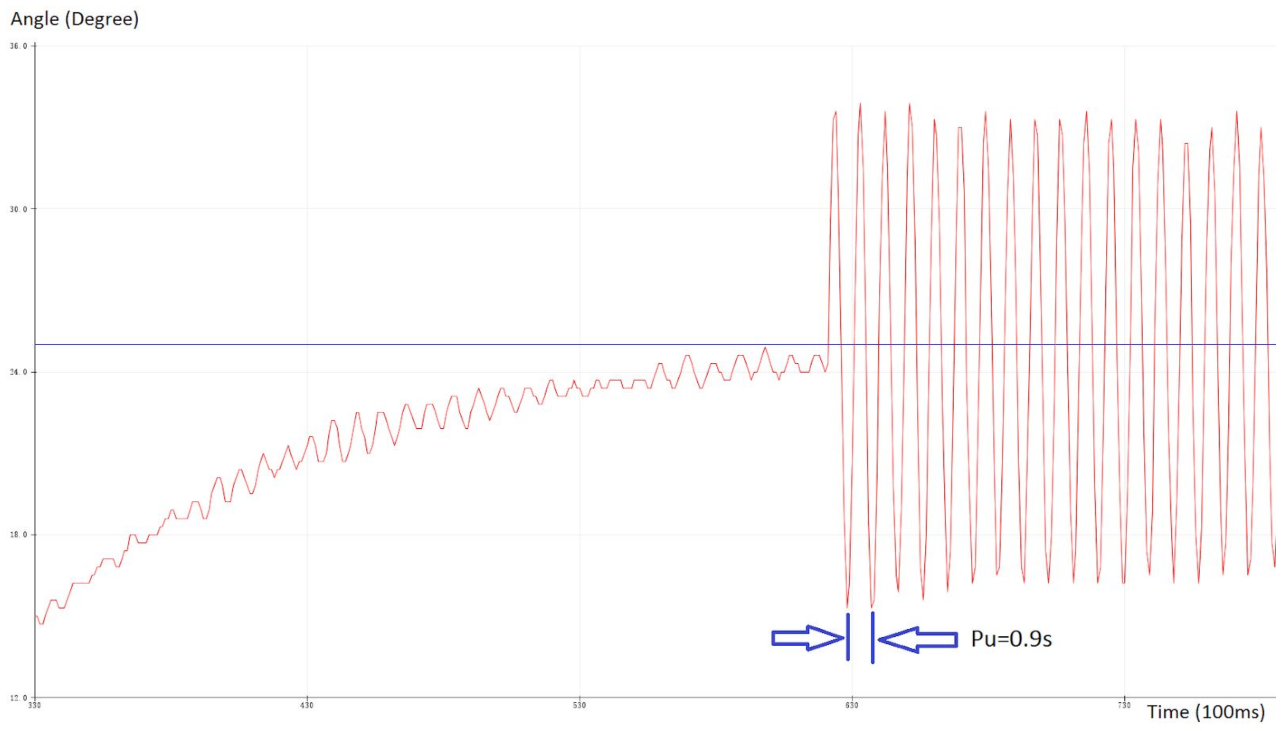

*Figure 4.Applying "Ultimate Cycle Method" to find Kp,Ki,Kd for the unknown plant*

The team then applied the theoretical value (Kp=2.28,Ki=1.0 and Kd=0.26) into the controller and find the system became very unstable since the Ki value is too high and the output will become very high in a short period of time. Normally, the team can start tuning these values using the theoretical value as a starting point. However, due to the previously mentioned encoder limitation, attempting to tune the controller using these value is very difficult. Since the output angle changes so rapidly, the encoder can no longer accurately track the feedback.  In order to bring the system response into a range where the encoder works reliability, the team had to reduce Ki to achieve a relatively stable initial clambing of the pendulum. The final value of Ki was set to be 0.0265. The team then noticed the rising time can be improved by increasing the Kp value, thus Kp was raised to around 5.0. After implementing KI control, the system could respond to a step input in a relatively fast and stable manner. However, the team noticed that the system output still has large oscillation near the target angle. Thus the Kd value was introduced. The team raised the Kd value to around 500 in order to reduce settling time. At this point, the team realized that the system does a poor job rejecting manual disturbance. Finally, the Kd value was increased to 1888 in order to create the PID controller suitable for this project

**Final tuned Kp, Ki and Kd value for the PID controller: Kp=5.0     Ki=0.0265 Kd=1888**

# Experimental Results and Discussion

**Test One: Stabilize at 30 degree**

For the first test, the system is required to be tuned so that the pendulum can be stabilized at angle between 0 - 30 degree while satisfying the design requirement. The team sets the target angle at 30 degree. From Figure 5, we can conclude that the system meets the design criteria.

Rise Time: 2.2s < 3.95s meets requirement
Settling Time: 3.6s < 6.7s meets requirement
% Overshoot:  2.7% < 5% meets requirement

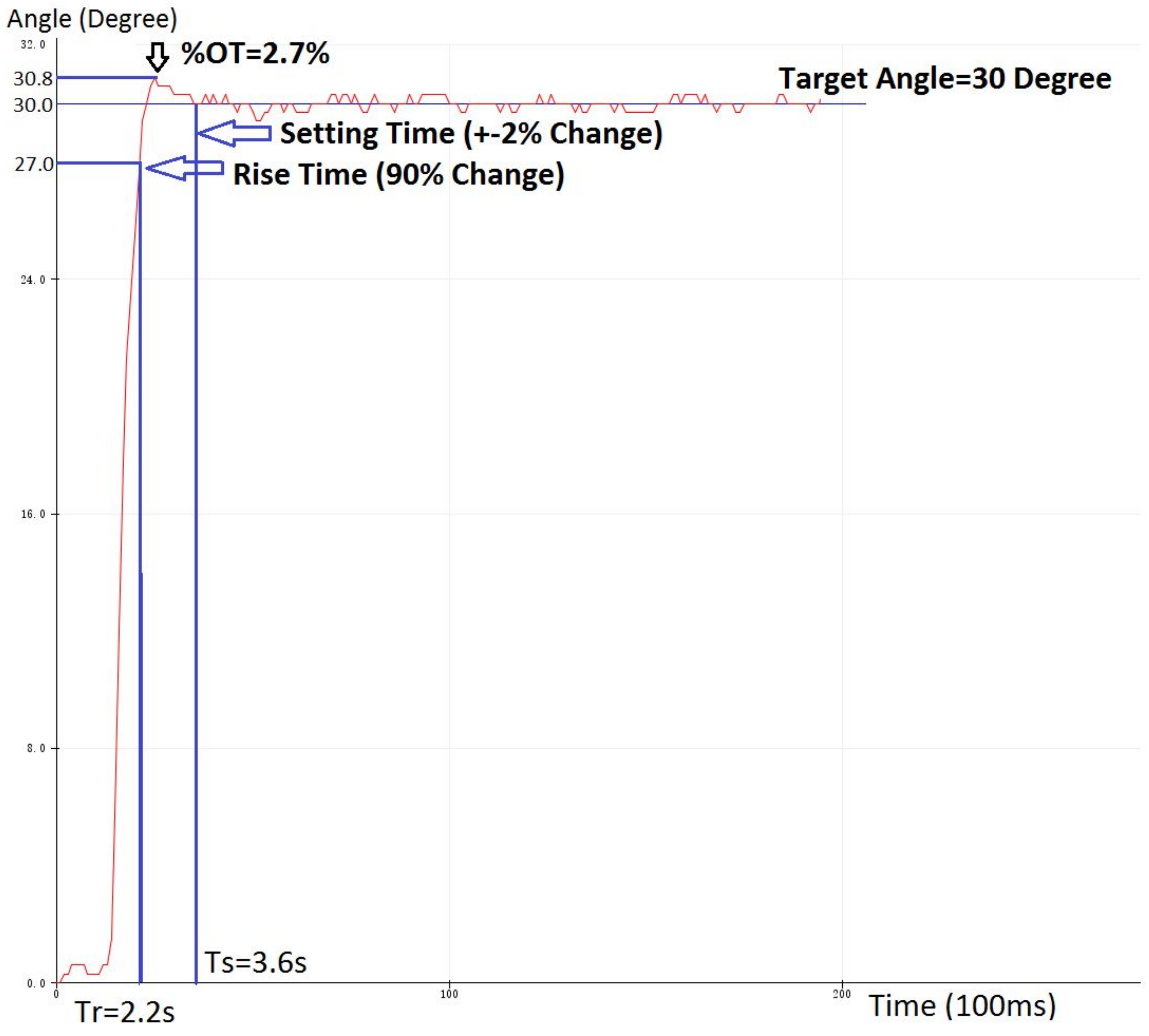

*Figure 5.System Response for 30 Degre*e Input (Sample Time 20ms)

**Test Two: Impact of the sampling rate on the controller performance.**

For the second test, the team uses three values as representation of too slow, optimum and too quick sampling frequency. The optimum value is chosen to be 20ms as it meets all system design requirements. The system response under the optimum sampling frequency is shown in Figure 5.

The team chose 7 ms to study the system pattern for quick sampling.  As shown in Figure 6 (Left), the rising time, approximately 5s, is longer than the time in optimum plot, and the system is oscillating during steady state phase. At quick sampling frequency, the system acquires high frequency noise and the unnecessary control signal assigned will slow the response time also causes fluctuation.

The team chose 35 ms to study the system pattern for slow sampling. As shown in Figure 6 (Right), the pendulum rises very quickly, creates lots overshoot and oscillation before stabilized. Since the slow sampling frequency delays the system reaction to target value, the controller applies a very large value at the beginning to reach the target. However, the value applied initially is usually too large than its actual need which eventually causes the issue in Figure 6 (Right).

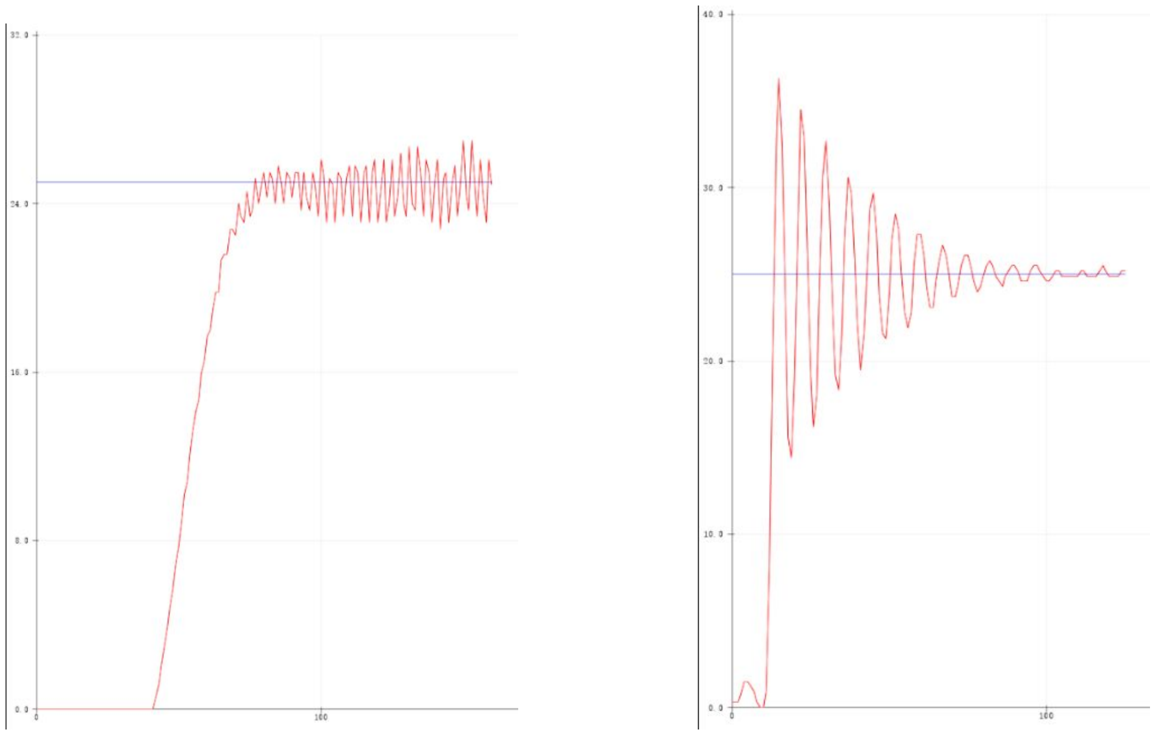

*Figure 6. System Response with Sample Time 7ms (Left) and  35 ms (Right)*

**Test Three: Reject disturbance at 30 degree**

For the third test, the team uses the tuned system as the base model and uses a finger tickling the pendulum to simulate the disturbance. The system plot of how the model is responding to the disturbance is shown in Figure 7. The settling time for the system to respond a single disturbance is approximately 1.7s. The settling time is considered quick and sufficient for this amount of disturbance. The large Kd value, 1888, contributes to the stabilization of system and quick response.

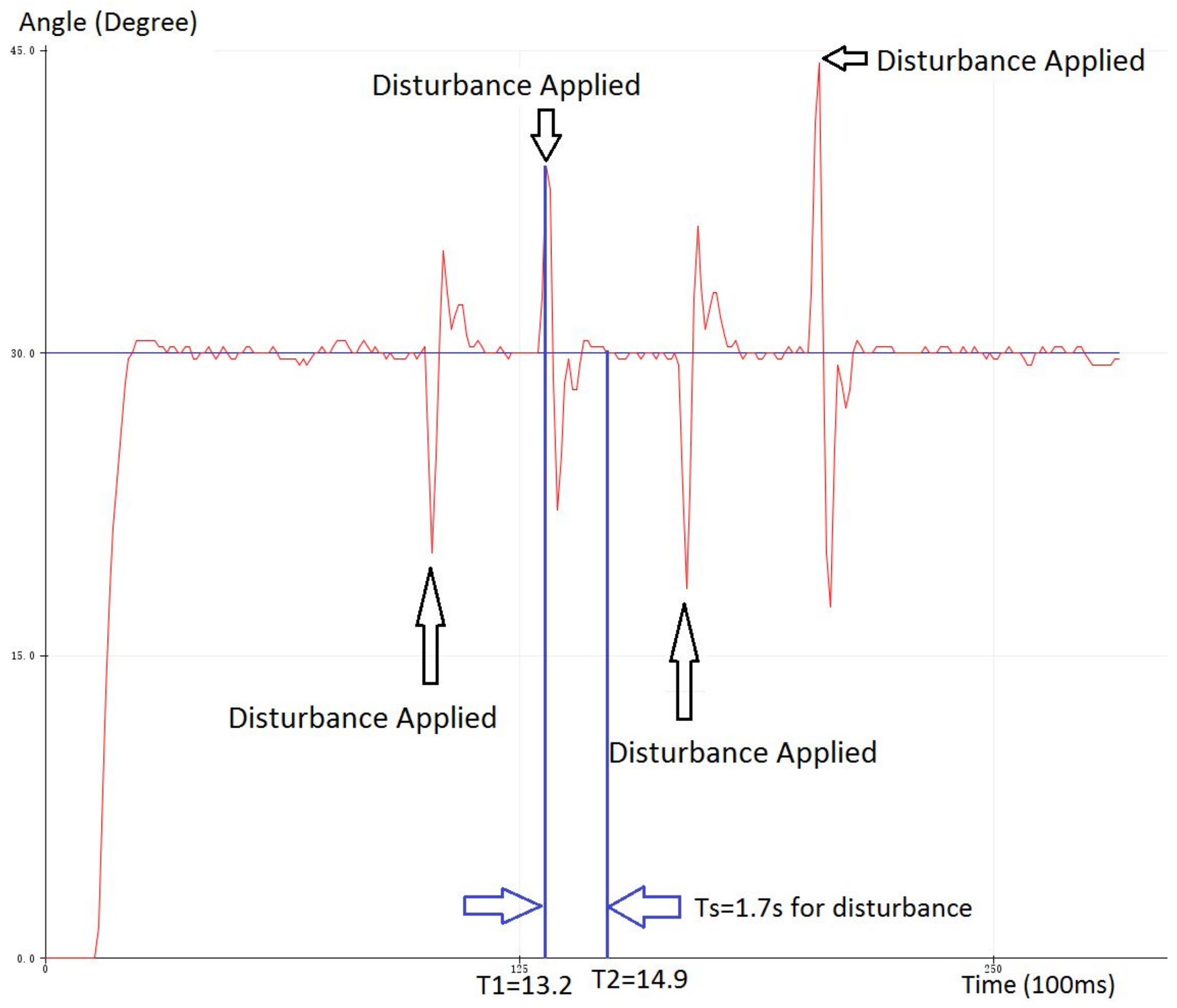

Figure 7.System Response for 30 Degree Input with Disturbance

**Test Four: Tracking a sinusoidal wave**

For the fourth test, the team changes the step input to sinusoidal input by changing the target angle from 30 degree to 30-15*sin(0.6283*0.001*t) to simulate the desired movement, traveling from 15-45 degree with a period of 10 second. The system plot is shown in Figure 8 above. According to the plot, the travel pattern of the pendulum matches with the sin curve. The movement of pendulum has approximately 0.3s delay and 10.8 degree shift. This is caused by the delay in serial communication and surrounding effect.

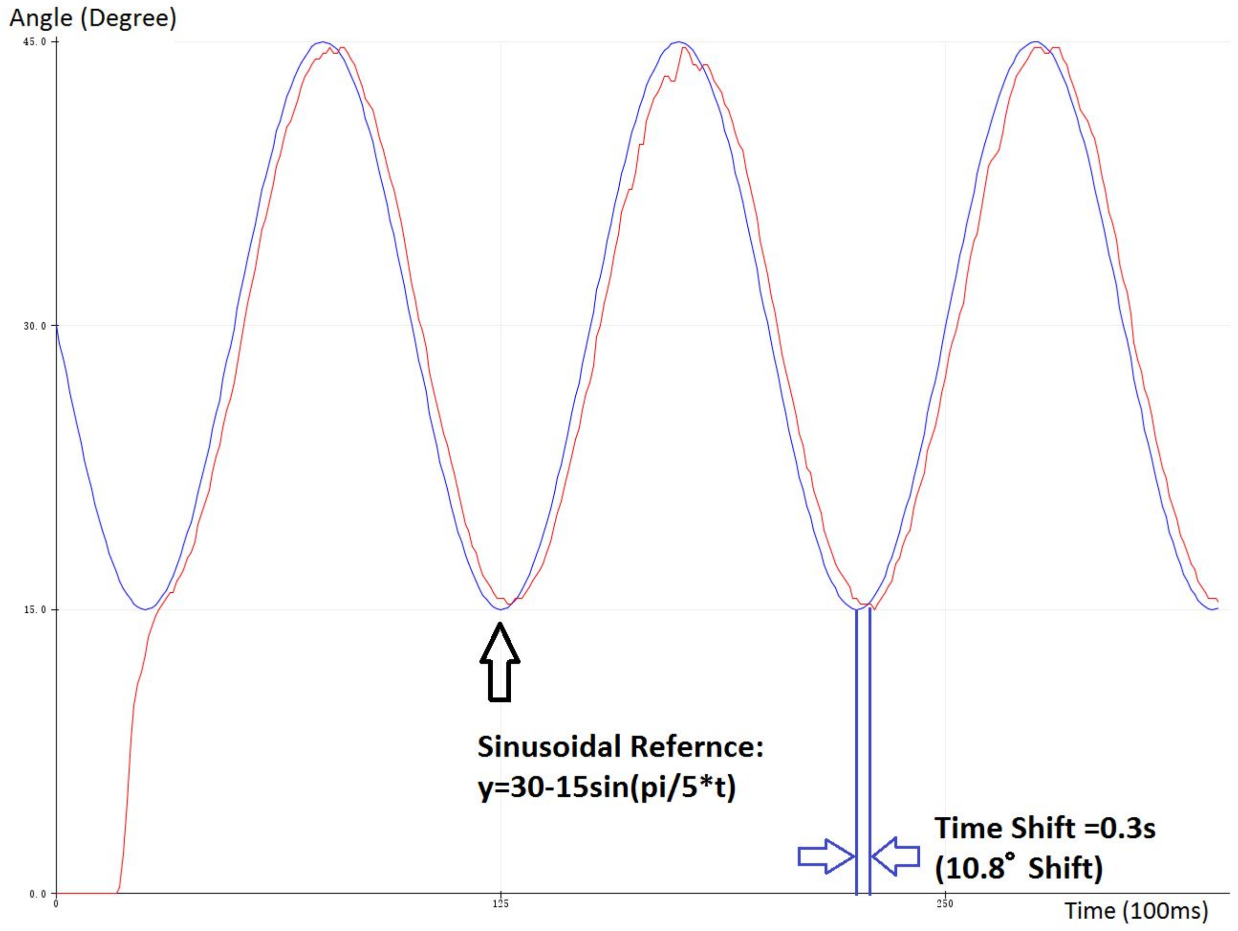

Figure 8.System Response for a 10 Second Period Sinusoidal Input

## Hardware and Software limitation

### Hardware Limitation:

**Encoder:**

The encoder cannot detect or afford large sudden pulse. For example, when the team is trying to use the Ultimate Cycle method, the team starts with increasing Kp until detect motion from the motor. However, due to the large Kp value and the coding logic, the pendulum flys up very high in a short period. The encoder is not accurate anymore when settle to the starting origin. In this case, the team cannot use this method as the team cannot achieve marginal stability with pure Kp controller.
At the same time, the encoder uses time-based pauses to determine the angle. This method can be interference with serial communication.

**Motor:**

The motor behaviour cannot be categorized as a linear model. The motor will only accept certain input signal within its working range. The signal needs to be greater than approximately 1030 before the motor can start operating. The system will operate from 1030 but not 0. However, the PID controller starts with imputing signal at zero. The mismatch between the motor and controller is a limitation for tuning the system. Also, the motor need to overcome its internal friction and imperfection which results in a more complicated system and introduces more uncertainties.

The motor has limited traveling range. The motor cannot travel to a large angle in a very fast speed. The motor electricity will be cut-off if the case happens; and the motor will swing back without freely.

Due to the same reason above, systematic method such as Ziegler-Nichols method or ultimate cycle method cannot be applied. Once the pendulum reaches a high angle, the motor will shut off and marginally stable cannot be achieved.

### Software Limitation:

The computer and arduino communicates with serial communication. The serial transfer is not instantaneous and requires time to transfer data. The Serial.write and Serial.print function can cause delay in the serial port and result in inaccuracy in the encoder reading.

The coding model is using integration model to estimate the travel distance and angle. The non-continuous model can only create discrete sections. Increasing sampling rate can increase the accuracy of the response. However, due to the limitation of the controller, the sampling frequency cannot be too low, or too quick. Otherwise, the system will be too sensitive to the environment and the disturbance.

## Conclusions

In this project, a control system contains a encoder and a propeller. They are constructed to obtain the desired output angle and reject disturbance. To satisfy the response time, overshoot and steady-state error requirements, PID controller is deployed. The proportional control improves the response time, Integral control eliminates the steady-state error; and derivative control reduces overshoot and also resists disturbance. Ultimate cycle method and numerical method are implemented to determine controller value Kp, Ki, Kd; and the 5.0, 0.0265 and 1888 are selected accordingly.

4 tests are performed and all design requirements are met. The first test is to quickly and accurately stabilize at a target angle, 30 degree. The 2.2s rise time, 3.6s settling time and 2.7% overshoot are achieved based on selected controller values. The second test is to evaluate the impact of sampling time. 20 ms sampling rate is selected as optimum/reference sampling frequency as it is the value used when the system meet all the requirements. A slow sampling frequency will result in system instability, rapid rise at the beginning. A rapid sampling frequency will slow the response time and also cause oscillation. The third test is to reject disturbance at target angle. With large Kd value 1888, the system is able to stabilize within 1.8s after disturbance is applied. The four test is to track a sinusoidal wave. The system is able to do so with only 0.3s delay and 10.8 degree phase shift, which is due to the serial communication.

This project not only provides the group with more in-depth understanding about the system control but also provides more practical insights than theoretical knowledge. For instance, a higher sampling rate is always desirable theoretically. However, in reality, sampling rate falls off the desired zone may causes instability or slow the response time. Additionally, the serial communication will also influence sampling. Another example would be the hardware limitations such as encoder cannot afford large sudden pulse and motor cannot overcome its internal friction given too small control signal.

# Attribution Table

| | Yuanshan Du | Hongzheng Xu | Yixiao Hong | Stephen Huang |
|---|---|---|---|---|
| Project Description | ET,FP | FD,MR,FP | ET,FP | FD,FP |
| Controller Definition/ Analysis | FD,ET,FP | FD,ET,MR | FD,FP | MR,FP |
| Experimental Results | FD,FP | ET,FP | ET,FP | ET,FP |
| Conclusions | ET,FP | FD,FP | ET,FP | ET,FP |

FD - First Draft

ET - Edit

MR - Major Revision

FP - Final Proofreading

# Appendix:

**Appendix A. Arduino Code**

```
/*
MIE404 Control Systems 1
Propeller Pendulum Project
Code Written by Lilly Yu, Francis Cruz, and Amy Bilton
Last Updated: 2018-10-30


*/
//--------------------LIBRARIES--------------------------//
/*
  Download the ESC library here: https://github.com/RB-ENantel/RC_ESC/
*/
#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 (ESC
PIN, Minimum Value, Maximum Value, Default Speed)
int oESC;                                    // Variable for the speed sent to the ESC

/*
  The variables and ports defined below are to guide you when writing the code, but feel
free to change or add/delete as necessary.
*/



//--------------------ENCODER PORTS---------------------//
int val;
#define outputA 6                          // Define Encoder output A to Digital Pin 6
on Arduino Uno
#define outputB 7                          // Define Encoder output B to Digital Pin 7
on Arduino Uno
```

```
#define NUMPULSES 1200
int counter = 0;                                // Define the number of pulses on Encoder
resolution (enter twice the amount of pulses on datasheet)
int encoder0PinALast = LOW;                     // Past state of the encoder output
int n = LOW;


//---------------------TIMER VARIABLES---------------------//

long t_now = 0;
long t_last_print = 0;
long t_last_PID = 0;
int T_sample = 20;                              // sample time in milliseconds (ms)
int T_print = 100;                              // sample print to monitor time in milliseconds
(ms)


//---------------------PID VARIABLES----------------------//
// Define variables for the output of the encoder(sensed_output), angle output of the
encoder(sensed_output converted to angle), and error w.r.t the setpoint

double sensed_output, error, current_angle;
// Define variables for total error over time, previous error in the last sampling time
interval, control signal, and limitations to the controller
double target_angle =25;
long total_error =0;
double last_error=0;
long control_signal;
int max_control = 1700;
int min_control = 1000;




// ==================INSERT DESIRED SETPOINT ANGLE
HERE================== //

// ==================INSERT CONTROL GAINS HERE===========
double Kp = 5.0;                                // proportional gain
double Ki = 0.0265;                              // integral gain in [ms^-1]
double Kd = 1888;                                // derivative gain in [ms]

double output_Kp = 0;
double output_Ki = 0;
double output_Kd = 0;
double output = 0;
double angle = 0.0;

void setup() {
    Serial.begin(9600);
  /*
    Setup function to initialize serial plotter/monitor, initialize origin of encoder,
    arm ESC driver, and to initialize propeller by ramping up and down
  */
  pinMode (outputA, INPUT);
  pinMode (outputB, INPUT);// Propeller speed to ramp up and down to check
functionality
  encoder0PinALast = digitalRead(outputA);             // Reads the initial state of
the outputA
  myESC.arm();                                     // Send the Arm value so the ESC will be
ready to take commands
  Serial.println("Setting up....");
  delay(7500);                                   // Wait for a while
  Serial.println("Testing Moter Function...");
  rampUpDown();
  Serial.println("Begin Main Loop!");
  Serial.println("============================");
  Serial.println("Press Anything to begin");
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////MAIN LOOP//////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void loop() {
  /*
    Main loop of the project.
    1. Read Encoder
    2. Implement PID control
    3. Send control signal to motor
    4. Print sensed angle with respect to the setpoint
  */

  while(Serial.available() < 1)             //Enter to begin
  {
  }

// Read the encoder
encoder_read();

// Take the magnitude of the encoder output (accounting for CW or CCW rotation)
sensed_output = abs(counter);

// Implement PID control
PID_Control();

// Write control signal to motor
myESC.speed(control_signal);

// Print sensed angle and setpoint angle
print_results();
}




void rampUpDown() {

  /*
    Function written to test functionality of brushless motor by accelerating and
decelerating the rotation
    of the motor. Speed of the motor does not accelerate to rated speed.
  */
  for (oESC = SPEED_MIN; oESC <= 1150; oESC += 1) {       // iterate from minimum
speed to a speed setting of 1150
    myESC.speed(oESC);                                   // write speed setting to motor
    delay(10);                                 // 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(10);                                 // waits 10ms for the ESC to reach speed
   }
  delay(7000);                                 // Wait for a while before going into control
loop
}




void encoder_read() {
  /*
    STUDENTS TO ANSWER
    Function to count the number of pulses from the encoder output.
    Must count at least 2 count per pulse, which would give you 1200 count per rotation.
    *****DO NOT USE INTERRUPTS TO READ ENCODER, IT COULD INTERFERE
WITH THE CONTROLLER CALCULATIONS, USE IT AT YOUR OWN RISK*****
  */
  // ==================INSERT ENCODER ALGORITHM HERE
==================== //
 n = digitalRead(outputA);
  if (encoder0PinALast != n ) {
    if (digitalRead(outputB) != n) {
      counter = counter -1;
    } else {
      counter = counter +1;
    }
  }
  encoder0PinALast = n;
  // ==================INSERT ENCODER ALGORITHM HERE
==================== //
}







void PID_Control() {
   // ==================INSERT CONTROL ALGORITHM HERE
==================== //

  /*
    STUDENTS TO ANSWER
    Function to implement PID control.
    Input: encoder position
    Output: motor speed
    Steps:
      1. Determine amount of time that has passed
      2. Determine 'if statement' to implement the use of a sampling time to implement
control signal
        (i.e. create if statement to represent sampling time and to calculate the control
signal every interval)
      3. Calculate the current error.
      4. Calculate control output assuming all gains of the PID are defined; a min and max
limit must be placed on the output ESC speed signal
        (HINT: use finite difference approx for derivative, use rectangular approximation
method to estimate area
        under curve for the integral term)
  */


  t_now = millis();                 // returns the number of milliseconds passed since the
Arduino started running the program
  if (t_now - t_last_PID >= 20){      // if the elapsed time is greater than the sampling
time, time to send control signal
```

```
   t_last_PID = t_now;
   // ==================INSERT CONTROL ALGORITHM HERE
==================== //

   sensed_output = abs(counter);                    //get absolute value for the counter
   current_angle = sensed_output / 1200 * 360;          //Convert counter value to angle
position

Target_angle=30-15*sin(0.6283*0.001*t);    //sinusoidal reference y=30+15sin(pi/5*t), t
in seconds
t=t+T_sample;                                          //increase t by sampling time

   error = target_angle - current_angle;               //Calculate error
   total_error += (error + last_error) / 2 * T_sample;    //Calculate integrated total error
   output_Kp = Kp * error;                          //Calculate Kp value proportional to error
   output_Ki = Ki * total_error;                     //Calculate Ki value proportional to
integrated total error
   output_Kd = Kd * (error - last_error) / T_sample;      //Calculate Kd value
proportional to error difference
   control_signal = output_Kp + output_Ki + output_Kd;    //Add up Ki,Kp.Kd
components
   last_error = error;                              //Save current error as last error

   // ==================INSERT CONTROL ALGORITHM HERE
===================== //

  }
}

void print_results() {
  /*
   STUDENTS TO ANSWER
   Function to print the sensed output/angle to the setpoint every 50 ms. Use Serial
plotter on Arduino to graphically plot the
   sensed angle with respect to the setpoint angle.

   HINT: You might want to print sensed_output to verify that the encoder is reading
correctly
```

```
    HINT: You might also want to print control_signal to ensure the PID is working
properly before writing it to the motor
  */

  t_now = millis();
  if (t_now - t_last_print >= T_print){
    t_last_print = t_now;

    // ==================INSERT PRINT ALGORITHM HERE
==================== //
Serial.print(target_angle);                //Print target value and current value for plot
Serial.print(" ");
Serial.println(current_angle);

/*
Serial.print("Desired Angle: ");
Serial.print(target_angle);
Serial.print("//Current Angle: ");
Serial.print(current_angle);
Serial.print("//Current Input: ");
Serial.print(control_signal);
Serial.print("====Ki: ");
Serial.print(output_Ki);
Serial.print("+Kp: ");
Serial.print(output_Kp);
Serial.print("+Kd: ");
Serial.println(output_Kd);
*/
    // ==================INSERT PRINT ALGORITHM HERE
==================== //
    }
}
```