# Mini Project Control System

## Seed Lab Team 3 – Fall 2020

Trey S . - Localization
Jackson W. - Simulation
Geoff M - System Integration
Eric S. - Vision

**This document includes the explanation of our Model Development as well as our Arduino Code and Simulink Block Diagram**

### Model Development

To create the PI model, the team began by trying to model the step response of our motor. Since the position of the wheel is linear with respect to time, the team analyzed the angular velocity of the motor wheel system which is steady. The team sampled the motor system every 10ms and calculated the angular velocity with the equation given in (1).

$$\omega = \frac{1000\Delta E}{3200T_s}2\pi \qquad (1)$$

Where Ts is in ms and delta E is the change in the encoder count. Once the equation was known, the team interfaced the Arduino with MATLAB to print the experimental results. From this data, the team modeled the transfer function and gain of the velocity feedback system. The model can be seen in Figure 1.

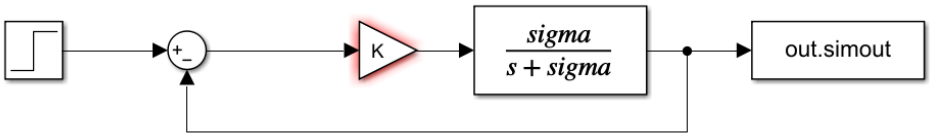


**Fig 1.** *Velocity Step Response Model*

Where K was found to be 12.34 and sigma was .65. The results of the velocity step response, both experimental and theoretical, are shown below in Figure 2.

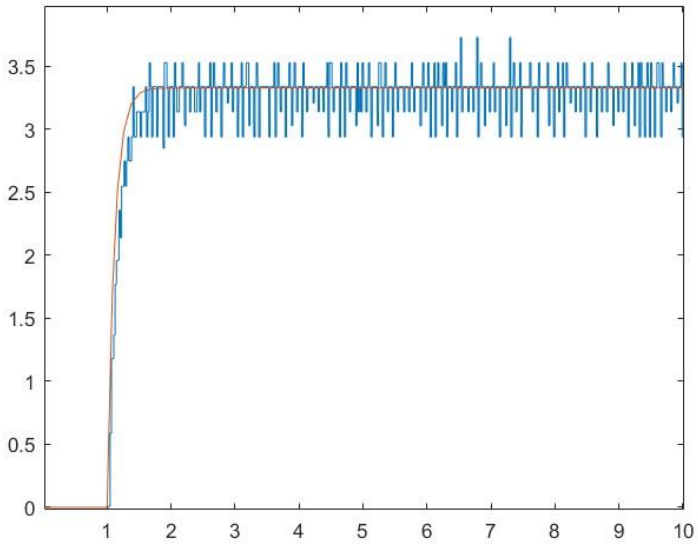


**Fig. 2:** *Motor Wheel System Simulated and Experimental Step Reponses*

After the team had an approximate step response, the team modeled the PI controller. This new system integrated the results of the velocity step response to get the position of the wheel. The full system may be seen in Figure 3. After tuning the PI block, the resulting position step response reached steady state very quickly.

Once the model was complete the team implemented the controller into an Arduino script. The model from the MATLAB model did not accurately control the system: there was a lot of overshoot in the output. This implied that the controller was underdamped. To fix this, the team lowered the integral gain significantly to get the system near critically damped. After this fix the system worked well and was simple to integrate with the vision.

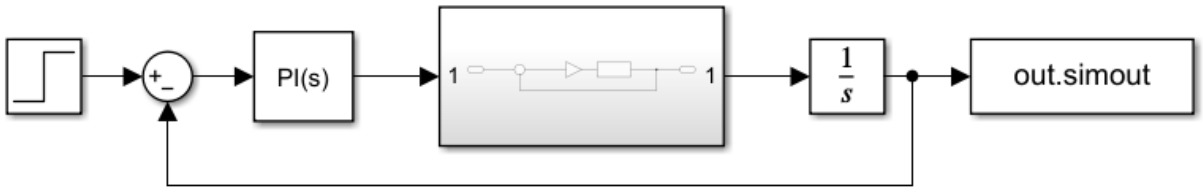


**Figure 3.** *Model of PI Controller*

The simulated results of the position's step-response may be seen in Figure 4. Furthermore, the experimental step response for the position is given in Figure 5. Something that may be worth considering for further exercise is only instituting the

integral control when the device is within a certain error tolerance. This would decrease the rise time of the system while still providing no overshoot.

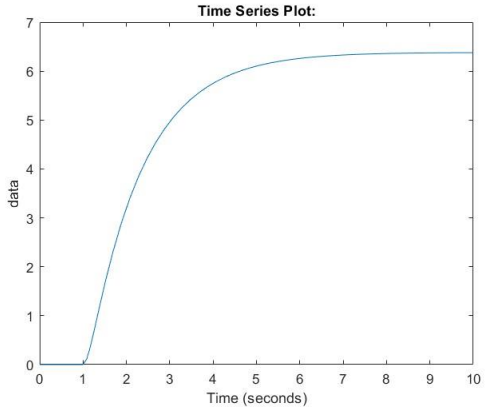


**Figure 4.** *Simulated Positional Step Response*

One error which caused a lot of problems was putting the integrator inside the PI controlled subsystem. This caused the system to be unstable and oscillate rapidly around the set point as it was trying to control the position and not the voltage. After this was fixed the controller worked better as well as giving a decent step response.

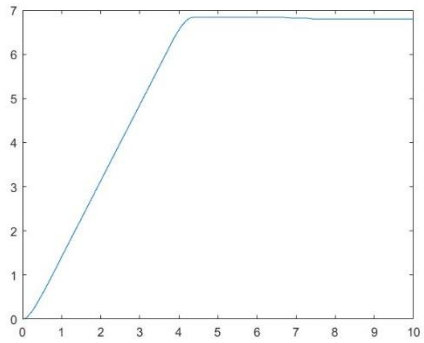


**Figure 5.** *Experimental Positional Step-Response*

## Arduino Code for Mini Project

```
//SEED lab team 3
//Mini Project Arduino Code
//Receive data over the i2c bus to determine what to turn a motor to
//Set the rotation of that motor using a PID loop

#include "DualMC33926MotorShield.h"
#include "Encoder.h"

#include "Wire.h"

#define sampleTime 10
```

```
#define ADDR 0x8

DualMC33926MotorShield ms;
Encoder encoder(2,5);

//Variables for holding received data
byte data[32] = {0};
byte value = 0;

//const double Ki = 0.156633991202905;
double Ki = .07, Kp = 15.8; //Not the same from matlab
//const double Kp = 3.18865312201695;
double setpoint = 0, I = 0, e = 0, Tc = 0, Ts = 0, u =0;
double curEnc = 0, prevEnc = 0;
double const umax = 7;

void setup() {
  ms.init();
  Serial.begin(115200);

  //Setup i2c
  Wire.begin(ADDR);
  delay(1000);
  Serial.print("------------ I am Arduino Uno I2C at address 0x");
  Serial.print(ADDR);
  Serial.println(" ------------");
  delay(1000);

  Wire.onRequest(requestData1);
  Wire.onReceive(receiveData1);

  //setup motor pins
  pinMode(4,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  digitalWrite(7,HIGH);
}

void loop() {
  if(millis() >= Tc + sampleTime){
    //Read the set point here
    curEnc = encoder.read();
    e = setpoint-curEnc/3200*6.28;\
```

```
    //Code for PID loop
      I += Ts*e;
      u = Kp*e+Ki*I;

      if(abs(u) > umax){
        u = sgn(u)*umax;
        I = (u-Kp*e)/Ki;
      }
      //Write data to the motors
      if( u < 0){
        analogWrite(9,0);
        digitalWrite(7,LOW);
      }
      if( u >= 0){
        analogWrite(9,0);
        digitalWrite(7,HIGH);
      }
      //analogWrite(9, u*12.34);
      analogWrite(9,abs(u)*255/12.34);
      Ts = millis()-Tc;
      Tc = millis();
    }
}

//Send position data back to the raspberry pi
//This is not needed but inteded to make the code more functional down the line
void requestData1() {
  Serial.println("---> Recieved Requests");
  Serial.println("Sending Back New Position");
  Serial.print("Sending value : ");
  Serial.println(value);
  Wire.write(value);
}

//Handle recieving data from the arduino over i2c
void receiveData1(int numBytes) {
  if(numBytes > 1) {
    int i = 0;
    while (Wire.available()) {
      data[1] = Wire.read();
      i++;
    }

    Serial.println(F("---> Recieved Events"));
    Serial.print(F("Recieved value : "));
```

```
    Serial.println(data[1]);

    value = int(data[1]);
    //Decode an angle from the i2c quadrant data
    switch(value){
      case 1:
        setpoint = 0;
        Serial.println(setpoint);
        break;
      case 2:
        setpoint = 1.57;
        Serial.println(setpoint);
        break;
      case 3:
        setpoint = 3.14;
        Serial.println(setpoint);
        break;
      case 4:
        setpoint = 4.71;
        Serial.println(setpoint);
        break;
    }
  }
}
//Handle negative rotation of the wheel
int sgn(float num){
  int sign = 1;
  if(num >= 0){
    sign = 1;
  } else {
    sign = -1;
  }
  return sign;
}
//Resources used are constant from previous assignments
```

### Arduino Code for Step Response

```
//SEED lab team 3
//Mini Project Arduino Code
//Code to handle a step response
//REcevie and sends data to Matlab over serial
#include "DualMC33926MotorShield.h"
#include "Encoder.h"
```

```
#include "Wire.h"

#define sampleTime 10
#define ADDR 0x8

DualMC33926MotorShield ms;
Encoder encoder(2,5);

//const double Ki = 0.156633991202905;
double Ki = .07, Kp = 15.8; //Not the same from matlab
//const double Kp = 3.18865312201695;
double setpoint = 0, I = 0, e = 0, Tc = 0, Ts = 0, u =0;
double curEnc = 0;
double const umax = 7;

void setup() {
  ms.init();
  Serial.begin(115200);
  pinMode(4,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  digitalWrite(7,HIGH);
  while(millis() <= 500){
    Serial.print(encoder.read());
    Serial.write(13);
    Serial.write(11);
  }
  setpoint = 6.28;
}

void loop() {
  if(millis() >= Tc + sampleTime){
    //Read the set point here
    curEnc = encoder.read();
    e = setpoint-curEnc/3200*6.28;
      I += Ts*e;
      u = Kp*e+Ki*I;

      if(abs(u) > umax){
        u = sgn(u)*umax;
        I = (u-Kp*e)/Ki;
      }
      if( u < 0){
```

```
        analogWrite(9,0);
        digitalWrite(7,LOW);
      }
      if( u >= 0){
        analogWrite(9,0);
        digitalWrite(7,HIGH);
      }
      //analogWrite(9, u*12.34);
      analogWrite(9,abs(u)*255/12.34);
      Ts = millis()-Tc;
      Tc = millis();
    }
    Serial.print(curEnc*6.8/3200);
    Serial.write(13);
    Serial.write(10);
}
//Handle negative rotation of the motor
int sgn(float num){
  int sign = 1;
  if(num >= 0){
    sign = 1;
  } else {
    sign = -1;
  }
  return sign;
}
```

### Arduino Code for Velocity Step Response

```
//SEED lab team 3
//Mini Project Arduino Code
//Velocity step response
#include "DualMC33926MotorShield.h"

#include "DualMC33926MotorShield.h"
#include "Encoder.h"

#define sampleTime 10

DualMC33926MotorShield motor;
Encoder encoder(2,5);

double dTheta = 0, dT = 0, dW = 0, prevTime = 0, currTime = 0, angVel = 0;
double currEnc = 0, preEnc = 0;
```

```
int CR = 13, LF = 10; //Define the Caridge Return and the Line Feed
double Tc = 0, Ts = 0;
void setup() {
  motor.init();
  Serial.begin(115200);

  //Define the following pinouts for the motor to run correctly
  pinMode(4,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);

  //Define the direction of the motor
  digitalWrite(7, HIGH);
  while(millis() <= 250){
    Serial.print(0);
    Serial.write(13);
    Serial.write(10);
  }
  analogWrite(9,144);
  Tc = millis();
}

void loop() {
  if(millis() >= Tc+sampleTime){
    preEnc = currEnc;
    currEnc = encoder.read();
    angVel = (currEnc - preEnc)*(6.28/3200)*1000/(Tc-Ts);
    Ts = Tc;
    Tc = millis();
  }
  Serial.print(angVel);
  Serial.write(CR);
  Serial.write(LF);
}
```

### MATLAB code for step response

```
%% STEP RESPONSE
clear all
close all
clc
%configure the configuration on the serial port
port = "COM3"; %Check on the system under devices
baud = 115200;
```

```
%Create a serial communication object
arduino = serialport(port, baud);

%configure the terminotaor with the Carrige Return and Line Feed values
configureTerminator(arduino, "CR/LF"); %Value of 13,10; respectivly

%clear the chanel of any remaing values
flush(arduino);

arduino.UserData = struct("AngVel", [], "Time", 1);
angVel = 0;
itt = 1;
%configureCallback(arduino,"terminator", @pollDUT);

while itt <= 10000
    angVel(itt) = readline(arduino);
    itt = itt + 1;
end

configureCallback(arduino,"off");

x = linspace(0,10,10000);

plot(x, angVel);
%%

 K = 12.34;
 sigma = .65;
%
open_system("Motor_Model.slx");
out = sim("Motor_Model.slx");
close_system("Motor_Model.slx");
plot(out.simout)
```