```
/*  Group 10 - Andrew Burton, Trevor Bachand, William Peyton, & Kyra Squier
 *  EENG350B - SEED Lab
 *  Mini Project Arduino Code
 *  01 March 2021
 *
 *  Description:
 *  The following program takes in a set point over I2C then uses a positional PI
 *  controller with encoder feedback to set a wheel to the given angle. The
 *  current location of the wheel is also output over I2C back the host device.
 */


#include <Wire.h>                   // I2C Library
#include <Encoder.h>                // Encoder Library

#define SLAVE_ADDRESS 0x04          // Arduino I2C Address

#define COUNTS_PER_REVOLUTION 3200  // Number of counts per revolution on the encoder on the motor

#define M_ENABLE  4                 // Motor enable pin -- must be set high to operate motors
#define M1_DIR    7                 // Motor 1 direction pin -- set true for cw rotation (i think) and false for ccw
#define M1_PWM    9                 // Motor 1 pwm pin -- set value between 0 and 255 (0 to not move, 255 is max speed)
#define TARE     11                 // Tare Button pin -- configure input_pullup

bool M1_Dir_Val = true;             // Value to store dir of motor -- set true for cw rotation and false for ccw
int  M1_PWM_Val = 0;                // Value to store speed of motor -- set value between 0 and 255 (0 to not move, 255 is max speed)

long myPosition  = 0;               // Store current position in cts

byte data_in = 0;                   // Variiable to store data sent from Pi to arduino
byte data_out[3] = {1, 2, 3};       // Variable to store data sent from arduino to Pi

float theta;                        // Radians
float thetaDesired;                 // Radians
float Va;                           // Volts
int T = 5;                          // Sample time in ms
float totalError;                   // Variable to store accumulated error
float prevTime = 0;                 // Variable to store last time control algorithm was applied (ms)
float Kp = 27;                      // Proportional gain
float Ki = 6;                       // Integral gain
int umax = 12;                      // Max ouput of controller (V)

int startTime = 0;                  // variable store program start time (ms)

Encoder myEnc(2, 5);                // Declare encoder object, with pin A = 2 (IOC) and pin B = 5

// Runs once at first boot up
void setup() {
  Wire.begin(SLAVE_ADDRESS);        // Initialize I2C as slave

  Wire.onReceive(receiveData);      // Set I2C interrupts
  Wire.onRequest(sendData);         // Set I2C interrupts

  pinMode(M_ENABLE, OUTPUT);        // Define enable pin as output
  pinMode(M1_DIR, OUTPUT);          // Define direction pin as output
  pinMode(M1_PWM, OUTPUT);          // Define pwm pin as output
  pinMode(TARE, INPUT_PULLUP);      // Define tare pin as input -- configure internal pullup
  digitalWrite(M_ENABLE, HIGH);     // IMPORTANT!! -- set enable pin high

  digitalWrite(M1_DIR, M1_Dir_Val); // Set initial motor direction to cw
  analogWrite(M1_PWM, M1_PWM_Val);  // Set initial motor speed to 0
}

// Runs repeatedly as long as power is applied to the arduino
void loop() {
  if(!digitalRead(TARE)) {          // If the tare button is pressed,
    zeroPosition();                 // Run the taring subroutine
  }
  updateSetPos(data_in);            // Update the set position to the most recent state value sent over I2C from the Pi
  updateCurrentPos();               // Read the encoder and update the position variables
  PI_Controller();                  // Apply the control algorithm
  setMotors();                      // Set the motor outputs to values determined by control algorithm
  convertData(data_out, theta);     // Update the array of bytes storing the digits of the current position to send to the Pi
}

// Interrupt routine called when Pi sends data to the arduino
void receiveData(int byteCount) {
  while (Wire.available())              // While the Pi sends bytes, read them in
    data_in = Wire.read();
  if (data_in == 0 || data_in == 255) { // If either a 0 or 255 is detected, try reading again
    data_in = Wire.read();
  }
}

// Interrupt routine called when arduino sends data to Pi
void sendData() {
  Wire.write(data_out, 3); // Write the array of bytes describing the first three digits of the position
}

// Update the array of bytes storing the digits of the current position to send to the Pi
void convertData(byte *pData, double currPos) {
  pData[0] = currPos;                        // Store the ones digit of currPos
  pData[1] = pData[0] * 10 - pData[0] * 10;   // Store the tenths digit of currPos
  pData[2] = pData[1] * 10 - pData[1] * 10;   // Store the hundreths digit of currPos
}

// Update the set position to the most recent state value sent over I2C from the Pi
void updateSetPos(int nextPos) {
  switch(nextPos) {
    case 0:
      thetaDesired = 0;
      break;
    case 1:
      thetaDesired = 1.57;
      break;
    case 2:
      thetaDesired = 3.14;
      break;
    case 3:
      thetaDesired = 4.71;
      break;
    case 4:
      thetaDesired = 0;
      break;
  }
}
```

```
// Read the encoder and update the position variables
void updateCurrentPos() {
  myPosition = myEnc.read();                            // Store the position in counts
  theta = 2 * PI * myPosition / COUNTS_PER_REVOLUTION;  // Convert counts to radians
}

// Apply the control algorithm
void PI_Controller() {
  float currentTime = millis();           // Returns current time value in ms
  int deltaTime = currentTime - prevTime; // Amount of time since previous algorithm application in ms

  if (deltaTime >= T) {                   // If more than 5 ms have passed since last control loop, then:
    float error = thetaDesired - theta;   // Calculate the current error
    prevTime = currentTime;               // Reset the clock

    // Cap the error to +/- 1 rad to prevent excessive values
    if ( error > 1) {
      error = 1;
    } else if (error < -1) {
      error = -1;
    }

    // If the total error is less than 0.5 rad or the error and total error have opposite signs
    if (abs(totalError) < 0.5 || (error < 0 && totalError > 0) || (error > 0 && totalError < 0)) {
      totalError += (error * deltaTime * 0.001); // Increment the integral of the error
    }

    Va = Kp * error + Ki * totalError;    //PI controller algorithm

    // Cap the output of the controls to +/- umax
    if (Va > umax) {
      Va = umax;
    } else if (Va < -umax) {
      Va = -umax;
    }
  }
}

// Set the motor outputs to values determined by control algorithm
void setMotors() {
  // Determine motor direction
  if (Va > 0) {
    M1_Dir_Val = true;
  } else {
    M1_Dir_Val = false;
  }

  // Determine motor speed and scale for pwm signal output
  M1_PWM_Val = map(abs(Va), 0, 12, 0, 255);

  // Set the motor controller inputs accordingly
  digitalWrite(M1_DIR, M1_Dir_Val);
  analogWrite(M1_PWM, M1_PWM_Val);
}

// Tare the encoder position
void zeroPosition() {
  digitalWrite(M_ENABLE, LOW);    // Disable the motor
  while(!digitalRead(TARE));      // Wait until tare button is released
  myEnc.write(0);                 // Reset the encoder position
  myPosition = 0;                 // Reset position variables
  theta = 0;                      // Reset position variables
  digitalWrite(M_ENABLE, HIGH);   // Re-enable the motors
  delay(250);                     // Wait a little bit to prevent issues
}
```

Controller Block Diagram: