Microcontrollers and Embedded Systems

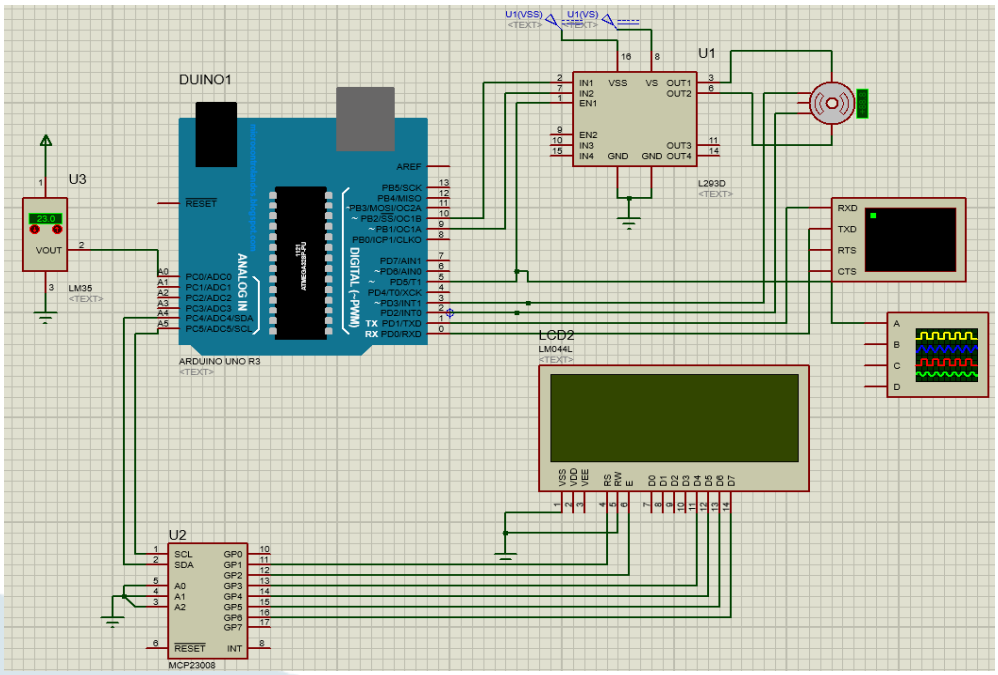
Lecture No. 7-8

Dr. Eng. Essa Alghannam

Exercise 1:

المطلوب: تصميم نظام إلكتروني متكامل (دارة برمجية وعتادية) يؤدي المهام التالية:

1. قراءة البيانات: قراءة الإشارة التشابهية (Analog) الصادرة عن حساس درجة الحرارة LM35.
2. العرض: إظهار قيمة درجة الحرارة المقاسة على شاشة LCD (16x2) باستخدام بروتوكول التخاطب I2C لتوفير عدد المداخل والمخارج.
3. التحكم الذكي:
   - إذا كانت درجة الحرارة 30 درجة مئوية أو أقل: يبقى محرك التيار المستمر (DC Motor) في حالة توقف.
   - إذا تجاوزت درجة الحرارة 30 درجة مئوية: يبدأ المحرك بالدوران تلقائياً.
   - التحكم في السرعة: يجب أن تزداد سرعة دوران المحرك تدريجياً وبشكل طردي مع ارتفاع درجة الحرارة (باستخدام تقنية PWM).
5. المراقبة: طباعة قيمة سرعة المحرك الحالية (باستخدام encoder) على شاشة الـ LCD بجانب درجة الحرارة.




```
#include <Wire.h>
#include <LiquidTWI2.h>
#define LCD_I2CADDR 0x20     // Connect via i2c, default address 0x20     // Set A0, A1, A2 to ground of 1st MCP23008
LiquidTWI2 lcd(LCD_I2CADDR);
const int TEMP=0; //Temp Sensor is on pin A0
double valueTEMP= 0;

// Motor Driver Pins
const int motorPin1 = 10;    // Motor driver input 1 (for direction)
const int motorPin2 = 9;    // Motor driver input 2 (for direction)
const int motorPWMPin = 5;   // Motor driver PWM enable pin (for speed)
```

```
const int encoderPinA = 2;  // Encoder output A (e.g., D2 on Arduino Uno for INT0)
const int encoderPinB = 3;  // Encoder output B (e.g., D3 on Arduino Uno for INT1) - Optional for direction
    int motorSpeed = 0;

volatile long encoderPulses = 0;   // volatile for variables modified in ISRs

void readEncoderA() {
 if (digitalRead(encoderPinB) == LOW) {
   encoderPulses++; // Or decrement depending on wiring and desired direction
 } else {
   encoderPulses--;
 }
}

void setup(){
 Serial.begin(9600);
 pinMode(motorPin1, OUTPUT);
 pinMode(motorPin2, OUTPUT);
 pinMode(motorPWMPin, OUTPUT);
 pinMode(13, OUTPUT);

 pinMode(encoderPinA,INPUT);
 pinMode(encoderPinB,INPUT);
 lcd.setMCPType(LTI_TYPE_MCP23008); // must be called before begin()
 lcd.begin(20,2);            // using 20 x 4 LCD display
 Wire.begin();
 lcd.home ();            // go home
```

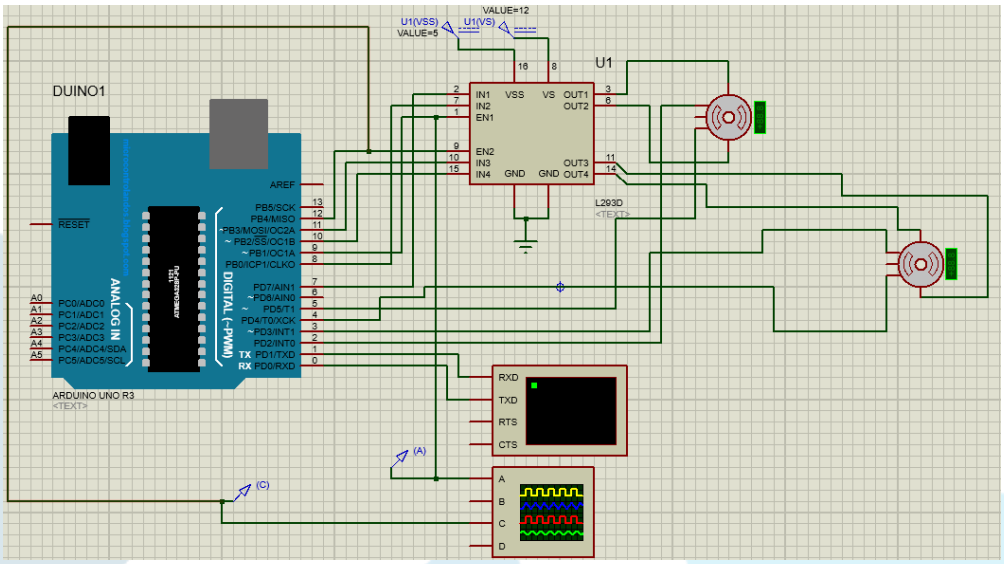
```
 lcd.setCursor(0,0);
 lcd.print(F("Using LiquidTWI2"));
 delay(2000);              // 2 seconds warm-up time
 lcd.clear();      // clears the LCD screen and positions the cursor in the upper-left corner.
 attachInterrupt(digitalPinToInterrupt(encoderPinA), readEncoderA, RISING);
 digitalWrite(motorPin1, HIGH);
 digitalWrite(motorPin2, LOW);
 analogWrite(motorPWMPin, 0);
}

void loop(){
    lcd.setCursor(0, 0);      //to adjust one whitespace for operator
    lcd.print(F("Using LiquidTWI2"));
    valueTEMP= 0.488*analogRead(TEMP);
    Serial.print("valueTEMP:");
    Serial.println(valueTEMP);
    delay(200);
    if (valueTEMP > 30) {
       motorSpeed = map(valueTEMP, 30, 50, 0, 255);
       motorSpeed = constrain(motorSpeed, 0, 255);
       analogWrite(motorPWMPin, motorSpeed);
       }
    else {
  analogWrite(motorPWMPin, 0);
 }
 Serial.print("motorSpeed:");
Serial.println(motorSpeed);
 delay(20); }
```

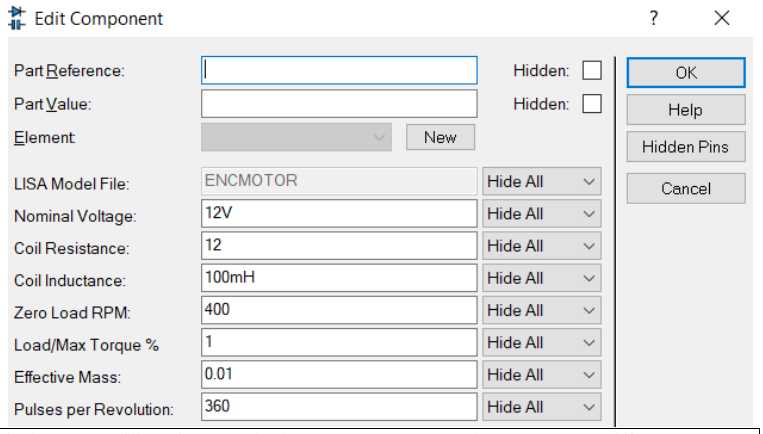
Arduino Code for Controlling the Speed of 2 DC Motors Using PID Control and Encoders with Interrupts

Exercise 2:

The code is structured to control two DC motors with speed feedback from encoders. The PID parameters are needed for each motor to achieve the desired performance and control characteristics.

Circuit:






```
//Arduino Code for Controlling the Speed of Two DC Motors Using PID Control//
                //-----and Encoders with Interrupts-----//

//PIN DEFINITIONS
//---------------------------------------------------------------//

// Motora Driver Pins
const int motoraPin1 = 7;     // Motora driver input 1 (for direction)
const int motoraPin2 = 8;     // Motora driver input 2 (for direction)
const int motoraPWMPin = 9;   // Motora driver PWM enable pin (for speed)

// Motorb Driver Pins
const int motorbPin1 = 10;     // Motorb driver input 1 (for direction)
const int motorbPin2 = 11;     // Motorb driver input 2 (for direction)
const int motorbPWMPin = 12;   // Motorb driver PWM enable pin (for speed)

// Encoder Pins (must be interrupt capable pins)
const int encoderaPinA = 2;   // Encoder output A (e.g., D2 on Arduino Uno for INT0)
const int encoderaPinB = 5;   // Encoder output B (e.g., D5 on Arduino Uno)  for direction
const int encoderbPinA = 3;   // Encoder output A (e.g., D3 on Arduino Uno for INT1)
const int encoderbPinB = 4;   // Encoder output B (e.g., D4 on Arduino Uno)  for direction

//---------------------------------------------------------------//
// PID PARAMETERS
//---------------------------------------------------------------//
// --- These MUST be tuned for your specific motor and setup! ---
double Kp = 3; // Proportional gain
double Ki = 1; // Integral gain
double Kd = 0.1; // Derivative gain
```

Commented [EA1]: Motor and Encoder Variables:
Two sets of motor pins, PWM pins, and encoder pins (motorPin1A, motorPin1B, motorPin2A, motorPin2B, etc.) are defined for the two motors.
Separate variables for encoder pulses (encoderPulses1, encoderPulses2) and RPM calculations are created to handle each motor individually.

Separate PID parameters **:Commented [EA2]**
(Kp1, Ki1, Kd1 for motor 1 and Kp2, Ki2, Kd2 for motor 2) are declared, allowing independent tuning for each motor.

```
//---------------------------------------------------------------//
//MOTOR & ENCODER SPECS
//---------------------------------------------------------------//
const int pulsesPerRevolution = 360; // Adjust to your encoder's PPR
                // (e.g., if encoder is 20 PPR and gearbox is 18:1, then 20*18=360)

//---------------------------------------------------------------//
// GLOBAL VARIABLES
//---------------------------------------------------------------//
volatile long encoderaPulses = 0;    // Use volatile for variables modified in ISRs
volatile long encoderbPulses = 0;    // Use volatile for variables modified in ISRs
long prevEncoderPulsesa = 0;
long prevEncoderPulsesb = 0;

double targetRPMa = 200.0;           // Desired motor speed in RPM
double targetRPMb = 200.0;
double currentRPMa = 0.0;
double currentRPMb = 0.0;

//double previousFilteredRPMa=0.0;
//double previousFilteredRPMb=0.0;


double pidErrora = 0.0;
double pidErrorb = 0.0;

double prevErrora = 0.0;
double integralErrora = 0.0;
double derivativeErrora = 0.0;
double pidOutputa = 0.0;             // PID controller output (will be mapped to PWM)

double prevErrorb = 0.0;
double integralErrorb = 0.0;
double derivativeErrorb = 0.0;
double pidOutputb = 0.0;      // PID controller output (will be mapped to PWM)

unsigned long lastPIDTime = 0;
unsigned long lastSpeedCalcTime = 0;
const unsigned int pidInterval = 20;     // PID calculation interval (ms)
const unsigned int speedCalcInterval = 50; // Speed calculation interval (ms)

//---------------------------------------------------------------//
//ENCODER INTERRUPT SERVICE ROUTINE
//---------------------------------------------------------------//
void readEncoderaA() {
  // Basic pulse counting. For direction, you'd also read PinB here.
```

```
  //encoderaPulses++;
  // If using PinB for direction:
  if (digitalRead(encoderaPinB) == LOW) {
    encoderaPulses++; // Or decrement depending on wiring and desired direction
  } else {
    encoderaPulses--;
  }
}

void readEncoderbA() {
  // Basic pulse counting. For direction, you'd also read PinB here.
  //encoderPulses++;
  // If using PinB for direction:
  if (digitalRead(encoderbPinB) == LOW) {
     encoderbPulses++; // Or decrement depending on wiring and desired direction
  } else {
    encoderbPulses--;
  }
}

// Optional: If you want to use both encoder channels for more precision or direction
// void readEncoderB() {
//   // For quadrature encoding to get 4x resolution and direction
//   // This requires more complex logic based on states of A and B
// }

//----------------------------------------------------------------//
//SETUP FUNCTION
//----------------------------------------------------------------//
void setup() {
  Serial.begin(9600);
  Serial.println("PID DC Motor Speed Control");

  // Motor Pins
  pinMode(motoraPin1, OUTPUT);
  pinMode(motoraPin2, OUTPUT);
  pinMode(motoraPWMPin, OUTPUT);

  pinMode(encoderaPinA,INPUT);
  pinMode(encoderaPinB,INPUT);

  pinMode(motorbPin1, OUTPUT);
  pinMode(motorbPin2, OUTPUT);
  pinMode(motorbPWMPin, OUTPUT);
  pinMode(encoderbPinA,INPUT);
  pinMode(encoderbPinB,INPUT);
```

Interrupt Service Routines (ISRs): :Commented [EA3]

```
  // Attach Interrupts
  // For Uno: Pin 2 is INT0, Pin 3 is INT1
  attachInterrupt(digitalPinToInterrupt(encoderaPinA), readEncoderaA, RISING);
  attachInterrupt(digitalPinToInterrupt(encoderbPinA), readEncoderbA, RISING);


  // Initialize motor to stopped state
  setMotora(0, true); // Speed 0, forward (direction doesn't matter much at speed 0)
  setMotorb(0, true);
  lastPIDTime = millis();
  lastSpeedCalcTime = millis();

  Serial.println("Enter target RPM via Serial Monitor (e.g., 'a100' or 'b100' for 100 RPM, 'p1.5' for Kp=1.5):");
}
//---------------------------------------------------------------//
//MAIN LOOP
//---------------------------------------------------------------//
void loop() {
  handleSerialInput(); // Check for commands to change setpoint or PID gains

  unsigned long currentTime = millis();

  // Calculate Current Speed periodically
  if (currentTime - lastSpeedCalcTime >= speedCalcInterval) {
    long currentPulsesa = encoderaPulses; // Make a copy for safe calculation
    long currentPulsesb = encoderbPulses;
    double deltaTimeSeconds = (currentTime - lastSpeedCalcTime) / 1000.0;

    long pulsesDeltaa = currentPulsesa - prevEncoderPulsesa;
    prevEncoderPulsesa = currentPulsesa;
    long pulsesDeltab = currentPulsesb - prevEncoderPulsesb;
    prevEncoderPulsesb = currentPulsesb;

    //RPM = (Pulses / PulsesPerRevolution) / (TimeDelta in Minutes)
    //RPM = (PulsesDelta / pulsesPerRevolution) * (60 / deltaTimeSeconds)
    if (deltaTimeSeconds > 0) { // Avoid division by zero
        currentRPMa = (double)(pulsesDeltaa) / pulsesPerRevolution * 60.0 / deltaTimeSeconds;
        currentRPMb = (double)(pulsesDeltab) / pulsesPerRevolution * 60.0 / deltaTimeSeconds;
    } else {
        currentRPMa = 0;
        currentRPMb = 0;
    }

    lastSpeedCalcTime = currentTime;
```

Commented [EA4]: Two ISRs are created for each encoder to count pulses (readEncoder1A, readEncoder1B for motor 1 and readEncoder2A, readEncoder2B for motor 2).

Commented [EA5]: The loop contains logic to calculate the RPM using encoder counts for both motors independently.

```
  // Optional: Moving average filter for RPM if it's too noisy
    currentRPMa = 0.7 * currentRPMa + 0.3 * previousFilteredRPMa;
    previousFilteredRPMa = currentRPMa;
    currentRPMb = 0.7 * currentRPMb + 0.3 * previousFilteredRPMb;
    previousFilteredRPMb = currentRPMb;
  }

  // Compute PID periodically
  if (currentTime - lastPIDTime >= pidInterval) {

    pidErrora = targetRPMa - currentRPMa;
    pidErrorb = targetRPMb - currentRPMb;
    // Proportional Term
    double pTerma = Kp * pidErrora;
    double pTermb = Kp * pidErrorb;

    // Integral Term (with anti-windup)
    integralErrora += Ki * pidErrora * (pidInterval / 1000.0); // Scale by time
    integralErrora = constrain(integralErrora, -200, 200); // Anti-windup: Limit integral term
    integralErrorb += Ki * pidErrorb * (pidInterval / 1000.0); // Scale by time
    integralErrorb = constrain(integralErrorb, -200, 200); // Anti-windup: Limit integral term

    // Derivative Term
    derivativeErrora = Kd * (pidErrora - prevErrora) / (pidInterval / 1000.0); // Scale by time
    prevErrora = pidErrora;
    derivativeErrorb = Kd * (pidErrorb - prevErrorb) / (pidInterval / 1000.0); // Scale by time
    prevErrorb = pidErrorb;

    pidOutputa = pTerma + integralErrora + derivativeErrora;
    pidOutputb = pTermb + integralErrorb + derivativeErrorb;

    // Constrain PID output to PWM range (0-255 for Arduino analogWrite)
    // For bi-directional control, PID output might be -255 to 255
    int motorSpeedPWMa = constrain(abs(pidOutputa), 0, 255);
    int motorSpeedPWMb = constrain(abs(pidOutputb), 0, 255);
    // Determine direction based on target RPM (or PID output sign if you map it differently)
    // positive targetRPM is always for forward.
    // For reverse, you might use negative targetRPM.
    bool forwarda = (targetRPMa >= 0); // Or (pidOutput >= 0) if you allow negative PID output
for reverse
    if (abs(targetRPMa) < 0.1) motorSpeedPWMa = 0; // If target is near zero, stop motor
    bool forwardb = (targetRPMb >= 0); // Or (pidOutput >= 0) if you allow negative PID output
for reverse
    if (abs(targetRPMb) < 0.1) motorSpeedPWMb = 0; // If target is near zero, stop motor
```

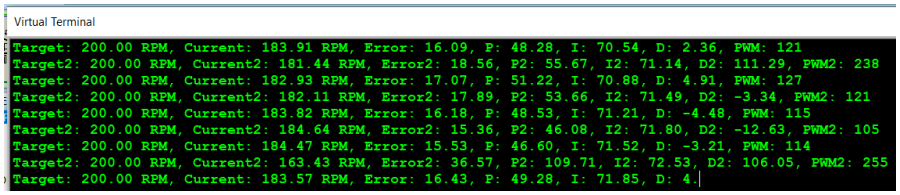
PID Control: Each motor has its own PID control calculations, updating PWM outputs for each motor based on the calculated PID outputs. **:Commented [EA6]**

```
    setMotora(motorSpeedPWMa, forwarda);
    setMotorb(motorSpeedPWMb, forwardb);

    lastPIDTime = currentTime;

    // Serial Output for Debugging/Tuning motor1
    Serial.print("Target: "); Serial.print(targetRPMa);
    Serial.print(" RPM, Current: "); Serial.print(currentRPMa, 2);
    Serial.print(" RPM, Error: "); Serial.print(pidErrora, 2);
    Serial.print(", P: "); Serial.print(pTerma, 2);
    Serial.print(", I: "); Serial.print(integralErrora, 2);
    Serial.print(", D: "); Serial.print(derivativeErrora, 2);
    Serial.print(", PWM: "); Serial.println(motorSpeedPWMa);

    // Serial Output for Debugging/Tuning motor2
    Serial.print("Target2: "); Serial.print(targetRPMb);
    Serial.print(" RPM, Current2: "); Serial.print(currentRPMb, 2);
    Serial.print(" RPM, Error2: "); Serial.print(pidErrorb, 2);
    Serial.print(", P2: "); Serial.print(pTermb, 2);
    Serial.print(", I2: "); Serial.print(integralErrorb, 2);
    Serial.print(", D2: "); Serial.print(derivativeErrorb, 2);
    Serial.print(", PWM2: "); Serial.println(motorSpeedPWMb);
  }
}

//-----------------------------------------------------------------//
//MOTOR CONTROL FUNCTION
//-----------------------------------------------------------------//
void setMotora(int speedPWM, bool forward) {
  if (forward) {
    digitalWrite(motoraPin1, HIGH);
    digitalWrite(motoraPin2, LOW);
  } else {
   digitalWrite(motoraPin1, LOW);
    digitalWrite(motoraPin2, HIGH);
  }
  analogWrite(motoraPWMPin, speedPWM);
}

void setMotorb(int speedPWM, bool forward) {
  if (forward) {
    digitalWrite(motorbPin1, HIGH);
    digitalWrite(motorbPin2, LOW);
  } else {
    digitalWrite(motorbPin1, LOW);
    digitalWrite(motorbPin2, HIGH);
```

Motor Control :Commented [EA7]
Function:The setMotor function is modified to handle two motors by taking an additional parameter to identify the motor being controlled.

```
  }
  analogWrite(motorbPWMPin, speedPWM);
}

//-------------------------------------------------------------------//
//HANDLE SERIAL INPUT FUNCTION
//-------------------------------------------------------------------//
void handleSerialInput() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();

    if (command.startsWith("b")) { // Set target RPM, e.g., b100
      targetRPMb = command.substring(1).toFloat();
      Serial.print("New Target RPM2: "); Serial.println(targetRPMb);
      integralErrorb = 0; // Reset integral error when setpoint changes
      prevErrorb = 0;     // Reset previous error
    }
    else if (command.startsWith("a")) {// Set target RPM, e.g., a100
    targetRPMa = command.substring(1).toFloat();
      Serial.print("New Target RPM1: "); Serial.println(targetRPMa);
      integralErrora = 0; // Reset integral error when setpoint changes
      prevErrora = 0;     // Reset previous error
    }
    else if (command.startsWith("p")) { // Set Kp, e.g., p2.5
      Kp = command.substring(1).toFloat();
      Serial.print("New Kp: "); Serial.println(Kp);
    } else if (command.startsWith("i")) { // Set Ki, e.g., i5.0
      Ki = command.substring(1).toFloat();
      Serial.print("New Ki: "); Serial.println(Ki);
    } else if (command.startsWith("d")) { // Set Kd, e.g., d0.1
      Kd = command.substring(1).toFloat();
      Serial.print("New Kd: "); Serial.println(Kd);
    } else {
      Serial.println("Unknown command. Use 's[RPM]', 'p[Kp]', 'i[Ki]', 'd[Kd]'");
    }
  }
}
```

The example handleSerialInput function is added to handle commands from the Serial Monitor. It sets a target RPM for both motors. You can expand this section to allow individual control of each motor by introducing more commands. **:Commented [EA8]**



PID DC Motor Speed Control
Enter target RPM via Serial Monitor (e.g., 'a100' or 'b100' for 100 RPM, 'p1.5' for Kp=1.5):
Target: 200.00 RPM, Current: 0.00 RPM, Error: 200.00, P: 600.00, I: 8.00, D: 2000.00, PWM: 255
Target2: 200.00 RPM, Current2: 0.00 RPM, Error2: 200.00, P2: 600.00, I2: 8.00, D2: 2000.00, PWM2: 255
Target: 200.00 RPM, Current: 12.40 RPM, Error: 187.60, P: 562.80, I: 15.50, D: -124.02, PWM: 255
Target2: 200.00 RPM, Current2: 12.40 RPM, Error2: 187.60, P2: 562.80, I2: 15.50, D2: -124.02, PWM2: 255
Target: 200.00 RPM, Current: 49.40 RPM, Error: 150.60, P: 451.81, I: 21.53, D: -369.95, PWM: 103
Target2: 200.00 RPM, Current2: 49.40 RPM, Error2: 150.60, P2: 451.81, I2: 21.53, D2: -369.95, PWM2: 103
Target: 200.00 RPM, Current: 78.41 RPM, Error: 121.59, P: 364.78, I: 26.39, D: -290.08, PWM: 101
Target2: 200.00 RPM, Current2: 72.32 RPM, Error2: 127.68, P2: 383.03, I2: 26.64, D2: -229.26, PWM2: 180
Target: 200.00 RPM, Current: 93.19 RPM, Error: 106.81, P: 320.43, I: 30.66, D: -147.85, PWM: 203
Target2: 200.00 RPM, Current2: 88.05 RPM, Error2: 111.95, P2: 335.86, I2: 31.11, D2: -157.25, PWM2: 209
Target: 200.00 RPM, Current: 108.13 RPM, Error: 91.87, P: 275.61, I: 34.34, D: -149.41, PWM: 160
Target2: 200.00 RPM, Current2: 117.65 RPM, Error2: 82.35, P2: 247.06, I2: 34.41, D2: -295.99, PWM2: 14
Target: 200.00 RPM, Current: 126.88 RPM, Error: 73.12, P: 219.35, I: 37.26, D: -187.53, PWM: 69
Target2: 200.00 RPM, Current2: 133.07 RPM, Error2: 66.93, P2: 200.78, I2: 37.08, D2: -154.25, PWM2: 83
Target: 200.00 RPM, Current: 136.31 RPM, Error: 63.69, P: 191.07, I: 39.81, D: -94.27, PWM: 136
Target2: 200.00 RPM, Current2: 125.89 RPM, Error2: 74.11, P2: 222.34, I2: 40.05, D2: 71.85, PWM2: 255
Target: 200.00 RPM, Current: 139.70 RPM, Error: 60.30, P: 180.91, I: 42.22, D: -33.86, PWM: 189
Target2: 200.00 RPM, Current2: 128.75 RPM, Error2: 71.25, P2: 213.74, I2: 42.90, D2: -28.68, PWM2: 227
Target: 200.00 RPM, Current: 150.20 RPM, Error: 49.80, P: 149.39, I: 44.22, D: -105.06, PWM: 88
Target2: 200.00 RPM, Current2: 151.94 RPM, Error2: 48.06, P2: 144.17, I2: 44.82, D2: -231.89, PWM2: 42
Target: 200.00 RPM, Current: 158.38 RPM, Error: 41.62, P: 124.87, I: 45.88, D: -81.76, PWM: 88
Target2: 200.00 RPM, Current2: 162.25 RPM, Error2: 37.75, P2: 113.25, I2: 46.33, D2: -103.06, PWM2: 56
Target: 200.00 RPM, Current: 159.13 RPM, Error: 40.87, P: 122.60, I: 47.51, D: -7.54, PWM: 162
Target2: 200.00 RPM, Current2: 151.88 RPM, Error2: 48.12, P2: 144.36, I2: 48.26, D2: 103.70, PWM2: 255
Target: 200.00 RPM, Current: 161.60 RPM, Error: 38.40, P: 115.19, I: 49.05, D: -24.70, PWM: 139
Target2: 200.00 RPM, Current2: 152.13 RPM, Error2: 47.87, P2: 143.60, I2: 50.17, D2: -2.54, PWM2: 191
Target: 200.00 RPM, Current: 169.09 RPM, Error: 30.91, P: 92.72, I: 50.29, D: -74.91, PWM: 68
Target2: 200.00 RPM, Current2: 172.45 RPM, Error2: 27.55, P2: 82.64, I2: 51.27, D2: -203.17, PWM2: 69
Target: 200.00 RPM, Current: 169.66 RPM, Error: 30.34, P: 91.02, I: 51.50, D: -5.67, PWM: 136
Target2: 200.00 RPM, Current2: 180.30 RPM, Error2: 19.70, P2: 59.11, I2: 52.06, D2: -78.44, PWM2: 32
Target: 200.00 RPM, Current: 168.70 RPM, Error: 31.30, P: 93.89, I: 52.75, D: 9.57, PWM: 156
Target2: 200.00 RPM, Current2: 167.91 RPM, Error2: 32.09, P2: 96.27, I2: 53.34, D2: 123.86, PWM2: 255
Target: 200.00 RPM, Current: 173.54 RPM, Error: 26.46, P: 79.39, I: 53.81, D: -48.35, PWM: 84
Target2: 200.00 RPM, Current2: 165.90 RPM, Error2: 34.10, P2: 102.30, I2: 54.71, D2: 20.08, PWM2: 177
Target: 200.00 RPM, Current: 174.93 RPM, Error: 25.07, P: 75.22, I: 54.81, D: -13.90, PWM: 116
Target2: 200.00 RPM, Current2: 183.35 RPM, Error2: 16.65, P2: 49.96, I2: 55.37, D2: -174.44, PWM2: 69
Target: 200.00 RPM, Current: 174.22 RPM, Error: 25.78, P: 77.35, I: 55.85, D: 7.10, PWM: 140
Target2: 200.00 RPM, Current2: 190.83 RPM, Error2: 9.17, P2: 27.50, I2: 55.74, D2: -74.88, PWM2: 8
Target: 200.00 RPM, Current: 176.48 RPM, Error: 23.52, P: 70.57, I: 56.79, D: -22.60, PWM: 104
Target2: 200.00 RPM, Current2: 177.40 RPM, Error2: 22.60, P2: 67.80, I2: 56.65, D2: 134.34, PWM2: 255

Target: 200.00 RPM, Current: 177.50 RPM, Error: 22.50, P: 67.50, I: 57.69, D: -10.23, PWM: 114
Target2: 200.00 RPM, Current2: 174.96 RPM, Error2: 25.04, P2: 75.12, I2: 57.65, D2: 24.40, PWM2: 157
Target: 200.00 RPM, Current: 177.28 RPM, Error: 22.72, P: 68.16, I: 58.60, D: 2.21, PWM: 128
Target2: 200.00 RPM, Current2: 191.81 RPM, Error2: 8.19, P2: 24.58, I2: 57.97, D2: -168.49, PWM2: 85
Target: 200.00 RPM, Current: 177.86 RPM, Error: 22.14, P: 66.43, I: 59.48, D: -5.78, PWM: 120
Target2: 200.00 RPM, Current2: 197.66 RPM, Error2: 2.34, P2: 7.03, I2: 58.07, D2: -58.49, PWM2: 6
Target: 200.00 RPM, Current: 179.89 RPM, Error: 20.11, P: 60.33, I: 60.29, D: -20.34, PWM: 100
Target2: 200.00 RPM, Current2: 184.09 RPM, Error2: 15.91, P2: 47.73, I2: 58.70, D2: 135.67, PWM2: 242
Target: 200.00 RPM, Current: 179.09 RPM, Error: 20.91, P: 62.74, I: 61.12, D: 8.03, PWM: 131
Target2: 200.00 RPM, Current2: 180.35 RPM, Error2: 19.65, P2: 58.96, I2: 59.49, D2: 37.42, PWM2: 155
Target: 200.00 RPM, Current: 178.40 RPM, Error: 21.60, P: 64.80, I: 61.99, D: 6.88, PWM: 133
Target2: 200.00 RPM, Current2: 196.51 RPM, Error2: 3.49, P2: 10.48, I2: 59.63, D2: -161.59, PWM2: 91
Target: 200.00 RPM, Current: 181.05 RPM, Error: 18.95, P: 56.84, I: 62.74, D: -26.53, PWM: 93
Target2: 200.00 RPM, Current2: 201.93 RPM, Error2: -1.93, P2: -5.78, I2: 59.55, D2: -54.19, PWM2: 0
Target: 200.00 RPM, Current: 180.70 RPM, Error: 19.30, P: 57.89, I: 63.52, D: 3.48, PWM: 124
Target2: 200.00 RPM, Current2: 187.54 RPM, Error2: 12.46, P2: 37.38, I2: 60.05, D2: 143.87, PWM2: 241
Target: 200.00 RPM, Current: 180.60 RPM, Error: 19.40, P: 58.20, I: 64.29, D: 1.04, PWM: 123
Target2: 200.00 RPM, Current2: 183.79 RPM, Error2: 16.21, P2: 48.62, I2: 60.70, D2: 37.44, PWM2: 146
Target: 200.00 RPM, Current: 180.57 RPM, Error: 19.43, P: 58.29, I: 65.07, D: 0.31, PWM: 123
Target2: 200.00 RPM, Current2: 199.26 RPM, Error2: 0.74, P2: 2.23, I2: 60.73, D2: -154.62, PWM2: 91
Target: 200.00 RPM, Current: 181.18 RPM, Error: 18.82, P: 56.45, I: 65.82, D: -6.13, PWM: 116
Target2: 200.00 RPM, Current2: 204.60 RPM, Error2: -4.60, P2: -13.81, I2: 60.55, D2: -53.48, PWM2: 6
Target: 200.00 RPM, Current: 181.78 RPM, Error: 18.22, P: 54.65, I: 66.55, D: -6.00, PWM: 115
Target2: 200.00 RPM, Current2: 188.81 RPM, Error2: 11.19, P2: 33.57, I2: 60.99, D2: 157.96, PWM2: 252
Target: 200.00 RPM, Current: 181.40 RPM, Error: 18.60, P: 55.81, I: 67.29, D: 3.86, PWM: 126
Target2: 200.00 RPM, Current2: 184.64 RPM, Error2: 15.36, P2: 46.09, I2: 61.61, D2: 41.72, PWM2: 149
Target: 200.00 RPM, Current: 183.15 RPM, Error: 16.85, P: 50.54, I: 67.97, D: -17.59, PWM: 100
Target2: 200.00 RPM, Current2: 201.94 RPM, Error2: -1.94, P2: -5.83, I2: 61.53, D2: -173.06, PWM2: 117
Target: 200.00 RPM, Current: 181.71 RPM, Error: 18.29, P: 54.87, I: 68.70, D: 14.45, PWM: 138
Target2: 200.00 RPM, Current2: 205.86 RPM, Error2: -5.86, P2: -17.57, I2: 61.30, D2: -39.13, PWM2: 4
Target: 200.00 RPM, Current: 181.99 RPM, Error: 18.01, P: 54.02, I: 69.42, D: -2.83, PWM: 120
Target2: 200.00 RPM, Current2: 189.81 RPM, Error2: 10.19, P2: 30.58, I2: 61.70, D2: 160.50, PWM2: 252
Target: 200.00 RPM, Current: 183.16 RPM, Error: 16.84, P: 50.53, I: 70.09, D: -11.65, PWM: 108
Target2: 200.00 RPM, Current2: 186.07 RPM, Error2: 13.93, P2: 41.80, I2: 62.26, D2: 37.38, PWM2: 141
Target: 200.00 RPM, Current: 183.51 RPM, Error: 16.49, P: 49.48, I: 70.75, D: -3.50, PWM: 116
Target2: 200.00 RPM, Current2: 201.37 RPM, Error2: -1.