CS 5331: Special Problems in CS: Cyber-Physical Systems Spring 2023

Assignment 7: Feedback using PID Controller for CPS

Amir faiyaz

R11772642

(a) An LDR is a type of sensor that alters its resistance according to the amount of light it receives. Its resistance decreases with more light and increases with less light. A red LED can serve as feedback in a PID controller for CPS. The system can use real-time data and feedback from an LDR and a Pot as inputs, and the red LED as output to regulate CPS behavior accurately and rapidly in response to environmental changes.
In a PID controller, Kp is the proportional gain parameter that controls the output's correction in response to the error signal. A larger Kp value results in a more significant correction for a given error. Ki is the integral gain parameter that considers past errors' contribution to the current correction. A larger Ki value results in a more significant correction for a given accumulated error over time. Finally, Kd is the derivative gain parameter that determines the correction amount based on the error signal's rate of change. A larger Kd value results in a more significant correction for a given error rate of change.

(b) **Code**:

```
#include <PID_v1.h>

const int photores = A0; // Photo resistor input
const int pot = A1; // Potentiometer input
const int led = 9; // LED output

double lightLevel; // variable that stores the incoming light level
double Setpoint, Input, Output; // variables for storing values

// Tuning parameters
float Kp = 0.0; // Initial Proportional Gain
float Ki = 10.0; // Initial Integral Gain
float Kd = 0.0; // Initial Differential Gain

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT); // Setup our
PID Loop

// Sampling rate and communication setup
const int sampleRate = 1; // Variable that determines how fast our PID
loop runs
const long serialPing = 500; // Serial pingback interval in milliseconds
unsigned long now = 0; // Placeholder for current timestamp
unsigned long lastMessage = 0; // Keeps track of when our loop last spoke
to serial

void setup() {
  lightLevel = analogRead(photores); // Read in light level
```

```
  Input = map(lightLevel, 0, 1024, 0, 255); // Change read scale to analog
out scale
  Setpoint = map(analogRead(pot), 0, 1024, 0, 255); // Get our setpoint
from our pot

  Serial.begin(9600); // Start a serial session
  myPID.SetMode(AUTOMATIC); // Turn on the PID loop
  myPID.SetSampleTime(sampleRate); // Set the sample rate
  Serial.println("Begin"); // Hello World!
  lastMessage = millis(); // Timestamp
}

void loop() {
  Setpoint = map(analogRead(pot), 0, 1024, 0, 255); // Read our setpoint
  lightLevel = analogRead(photores); // Get the light level
  Input = map(lightLevel, 0, 900, 0, 255); // Map it to the right scale
  myPID.Compute(); // Run the PID loop
  analogWrite(led, Output); // Write out the output from the PID loop to
our LED pin
  now = millis(); // Keep track of time

  if(now - lastMessage > serialPing) { // If it has been long enough give
us some info on serial
    // This should execute less frequently
    Serial.print("Setpoint = ");
    Serial.print(Setpoint);
    Serial.print(" Input = ");
    Serial.print(Input);
    Serial.print(" Output = ");
    Serial.print(Output);
    Serial.print("\n");

    if (Serial.available() > 0) { // If we sent the program a command deal
with it
      Kp = Serial.parseFloat();
      Ki = Serial.parseFloat();
      Kd = Serial.parseFloat();
      myPID.SetTunings(Kp, Ki, Kd); // Set the PID gain constants and
start running
      Serial.print("Kp,Ki,Kd = ");
```

```
      Serial.print(Kp);
      Serial.print(",");
      Serial.print(Ki);
      Serial.print(",");
      Serial.println(Kd); // Let us know what we just received
      while (Serial.available() > 0) Serial.read(); // Clear out any
residual junk
    }

    lastMessage = now; // Update the time stamp
  }
}
```

(c)

1. When a piece of paper is inserted between the LED and LDR, the resistance of the LDR increases, causing the voltage across it to also increase. The feedback loop then detects this increase and sends a signal to the LED to increase its voltage in order to maintain a constant level of brightness. As a result, the LED appears brighter.

2. When Ki is set to 0, the controller only takes into account the current error and does not consider past errors when making adjustments. This may cause the controller to respond slowly to sudden changes in the system.
However, when Ki is set to a non-zero value, the controller integrates the errors over time and adjusts the control action accordingly. This allows the controller to respond more quickly to sudden changes in the system and can be more effective in reducing steady-state errors.
When Ki is set to 0.5, the controller is moderately responsive to changes in the system. Increasing Ki to 1 makes the controller more responsive, while Ki values of 5 or 10 make the controller highly responsive. However, setting Ki too high can lead to overshoot or instability in the system, so it is important to carefully select an appropriate value for Ki based on the system's characteristics.

Keeping Kp and Ki to 0 and 10, Kd 0 = Setpoint = 254.00 Input = 253.00 Output = 217.48
Keeping Kp and Ki to 0 and 10, Kd 0.5 = Setpoint = 254.00 Input = 255.00 Output = 220.84
Keeping Kp and Ki to 0 and 10, Kd 1 = Setpoint = 254.00 Input = 238.00 Output = 255.00
Keeping Kp and Ki to 0 and 10, Kd 5 = Setpoint = 254.00 Input = 245.00 Output = 0.00
Keeping Kp and Ki to 0 and 10, Kd 10 = Setpoint = 254.00 Input = 245.00 Output = 0.00

Kp = 0, Ki = 10, Kd = 0: The controller will only use the accumulated error to adjust the output, which may result in slow response and overshoot.

Kp = 0, Ki = 10, Kd = 0.5: The controller will be more responsive to changes in the error, and can help reduce overshoot and improve settling time.
Kp = 0, Ki = 10, Kd = 1: The controller will be even more responsive to changes in the error, and can further reduce overshoot and settling time.
Kp = 0, Ki = 10, Kd = 5: The controller will be very responsive to changes in the error, but may start to oscillate if set too high.
Kp = 0, Ki = 10, Kd = 10: The controller may become unstable and oscillate.

Keeping Kd and Ki to 0 and 10, Kp 0 = Setpoint = 254.00 Input = 253.00 Output = 217.48
Keeping Kd and Ki to 0 and 10, Kp 0.5 = Setpoint = 254.00 Input = 251.00 Output = 222.50
Keeping Kd and Ki to 0 and 10, Kp 1 = Setpoint = 254.00 Input = 251.00 Output = 223.56
Keeping Kd and Ki to 0 and 10, Kp 5 = Setpoint = 254.00 Input = 251.00 Output = 236.75
Keeping Kd and Ki to 0 and 10, Kp 10 = Setpoint = 254.00 Input = 251.00 Output = 245.61

When Kp is increased, the system becomes more responsive to the current error, which can help to achieve the setpoint faster but can also lead to overshoot and oscillation.

3. When the lighting conditions in the environment change, the light intensity detected by the sensor will also change, leading to a shift in the error signal. The rate PID loop measures the rate of change in the error signal and adjusts the control output accordingly to maintain the set point. For instance, if the environment becomes brighter, the error signal will increase, and the rate PID loop will amplify the control output to reduce the error and bring the actual light intensity back to the desired set point.
4. The DIRECT parameter plays a crucial role in determining whether the output of the PID controller is a positive or negative value for a given error signal. In this case, the value of DIRECT is set to 0. However, if we invert the value of DIRECT to 1, it may cause the LED to turn off. For example, if the input value is higher than the set point value, it may result in a negative error and a negative output, causing the LED to become dim or turn off.

(d) **Snippet**:

Assignment 7 | Arduino IDE 2.0.3

File Edit Sketch Tools Help

Arduino Uno

Assignment 7.ino

```
8   float Ki=10; //Initial Integral Gain
9   float Kd=0; //Initial Differential Gain
10  double Setpoint, Input, Output; //These are just variables for storingvalues
11
12
13  PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
14  // This sets up our PDID Loop
15  //Input is our PV
16  //Output is our u(t)
17  //Setpoint is our SP
18  const int sampleRate = 1; // Variable that determines how fast our PID loop runs
19  // Communication setup
20  const long serialPing = 500; //This determines how often we ping our loop
21  // Serial pingback interval in milliseconds
22  unsigned long now = 0; //This variable is used to keep track of time
23  // placehodler for current timestamp
24  unsigned long lastMessage = 0; //This keeps track of when our loop last
25  //spoke to serial last message timestamp.
26
27
28  void setup(){
29    lightLevel = analogRead(photores); //Read in light level
30    Input = map(lightLevel, 0, 1024, 0, 255); //Change read scale to analog
31    //out scale
```

Output Serial Monitor

```
Sketch uses 6968 bytes (21%) of program storage space. Maximum is 32256 bytes.
Global variables use 346 bytes (16%) of dynamic memory, leaving 1702 bytes for local variables. Maximum is 2048 bytes.
```

(e)