CS 5331-003: Special Problems in Computer Science: Embedded Systems

Instructor: Dr. Morshed, Associate Professor

Assignment-7: Feedback Using PID Controller for CPS

Anvesh Muppeda

R11840667

amuppeda@ttu.edu

Submission Date: 04-04-2024

Task (100%):

a) **Describe the implemented system of CPS and explain each term of PID.**

Implemented CPS:

- A closed-loop cyber-physical system (CPS) feedback controller that integrates the cyber and physical domains.
- The PID (Proportional Integral Differential) algorithm will be used as a control algorithm.
- For input and output, we'll utilize a light sensor and an LED.
  These will provide as an interface between the cyber and physical realms. For example, in a CPS, a cyber component (a PID controller) uses input to manage a physical process (the intensity of light).

PID:

- A basic controller system is PID.
- "Proportional, Integral, Derivative" is what it stands for.
- It explains the controller's functioning.
- Proportional (P) term: This term generates an output that is proportionate to the error signal in response to the current error. Depending on how far the current value deviates from the intended setpoint, it provides a control output.
- Integral (I) term: This term reacts to both the length and the magnitude of the error, adding up the errors over time. When the proportional term is insufficient to raise the process variable to the setpoint, it removes the residual error that remains.
- Derivative (D) term: This term uses the error's present rate of change to forecast how it will behave going forward. It assists in reducing the responsiveness of the system, averting oscillations and overshoot.
- The intended value that the system should reach is known as the setpoint (SP).
- Process Variable (PV): The light level in this instance, which is the present value of the physical process under control.
- Output (u(t)): The PID controller's calculated control action, which is used to move the process variable closer to the setpoint.
- "Tuning" refers to computing or modifying the constants, Kp, Ki, and Kd.

b) **Provide the complete software code with sufficient comments.**
Code:

```
#include <PID_v1.h>

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

// Tuning parameters for PID controller
float Kp=0; //Initial Proportional Gain
float Ki=10; //Initial Integral Gain
float Kd=0; //Initial Differential Gain

// Variables for storing sensor readings and control values
```

```
double lightLevel; // Holds incoming light level
double Setpoint, Input, Output; // Variables for storing values

// Create the PID object
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT); // Set up PID Loop: Input
is PV, Output is u(t), Setpoint is SP

// Sampling rate for PID loop
const int sampleRate = 1; // Variable that determines how fast our PID loop runs

// Communication setup
const long serialPing = 500; //This determines how often we ping our loop
                             // Serial pingback interval in milliseconds
unsigned long now = 0; //This variable is used to keep track of time
                       // placehodler for current timestamp
unsigned long lastMessage = 0; //This keeps track of when our loop last
                               //Timestamp of the last serial message

void setup()
{
    // Initialize sensor readings and setpoints
    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

    //Start a serial session
    Serial.begin(9600);

    // Enable PID loop
    myPID.SetMode(AUTOMATIC); //Turn on the PID loop

    // Set PID loop sample rate
    myPID.SetSampleTime(sampleRate); //Sets the sample rate

    // Print initialization message
    Serial.println("Begin"); // Hello World!

    // Record initialization timestamp
    lastMessage = millis(); // timestamp
}

void loop()
{
    // Update setpoint and sensor readings
    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
    //Run the PID loop
    myPID.Compute();

    // Adjust LED brightness based on PID output
    analogWrite(led, Output); //Write out the output from the PID loop
        //to our LED pin

    // Update current time
    now = millis(); //Keep track of time

    // Send data over serial at specified interval
    if(now - lastMessage > serialPing)
    {   //If it has been long enough give us
        //some info on serial
        // this should execute less frequently
        // send a message back to the mother ship
        // Send sensor readings and PID output over serial
        Serial.print("Setpoint = ");
        Serial.print(Setpoint);
        Serial.print(" Input = ");
        Serial.print(Input);
        Serial.print(" Output = ");
        Serial.print(Output);
        Serial.print("\n");

        // Receive PID tuning parameters over serial and update if new values are
received
        if (Serial.available() > 0)
        { //If we sent the program a command deal
            //with it
            for (int x = 0; x < 4; x++)
            {
                switch (x)
                {
                    case 0:
                        Kp = Serial.parseFloat();   // Read Proportional Gain
                        break;
                    case 1:
                        Ki = Serial.parseFloat();   // Read Integral Gain
                        break;
                    case 2:
                        Kd = Serial.parseFloat();   // Read Differential Gain
                        break;
                    case 3:
                        for (int y = Serial.available(); y == 0; y--)
                        {
                            Serial.read();          //Clear out any residual junk
                        }
                        break;
                }
            }

            // Print received PID tuning parameters
            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

            // Update PID controller with new tuning parameters
            myPID.SetTunings(Kp, Ki, Kd); //Set the PID gain constants
            //and start running
        }

        //update the time stamp.
        lastMessage = now;
    }
}
```

c) Answer to the following questions:

A. Explain your observation in Step 12. Why LED turns brighter when a paper is inserted between the LED and the LDR?

- In Step 12, the LED becomes brighter when a piece of paper is placed between it and the LDR.
- Based on how the PID controller operates and how the LDR behaves, this observation can be explained.
- LDR behavior: As the intensity of light falling on it increases, the LDR's resistance reduces. A portion of the light that would otherwise reach the LDR is blocked when paper is placed between the LED and the LDR, increasing the resistance of the LDR.
- How the PID Controller Works: Based on the discrepancy between the intended light level (Setpoint) and the actual light level (Input), the PID controller modifies the output to the LED. To reduce this disparity, the PID controller calculates the control signal (Output) continually.
- Impact on LED Brightness: The actual light level that the sensor detects drops when the paper stops some light from reaching the LDR. The PID controller increases the control signal given to the LED to make up for the decreased light intensity as a result of this drop in light level.
- In other words, when paper is placed between the LED and the LDR, lowering the light intensity reaching the LDR, the PID controller reacts by raising the LED's brightness to maintain the desired light level set by the potentiometer. As a result, the LED becomes brighter as the controller increases the output to maintain the desired light level (Setpoint).

B. In step 13, do the following and note the observations:
Keeping Kp and Kd to 0 and 0, respectively, change ki to 0, 0.5, 1, 5, 10
Keeping Kp and Ki to 0 and 10, respectively, change kd to 0, 0.5, 1, 5, 10
Keeping Kd and Ki to 0 and 10, respectively, change kp to 0, 0.5, 1, 5, 10

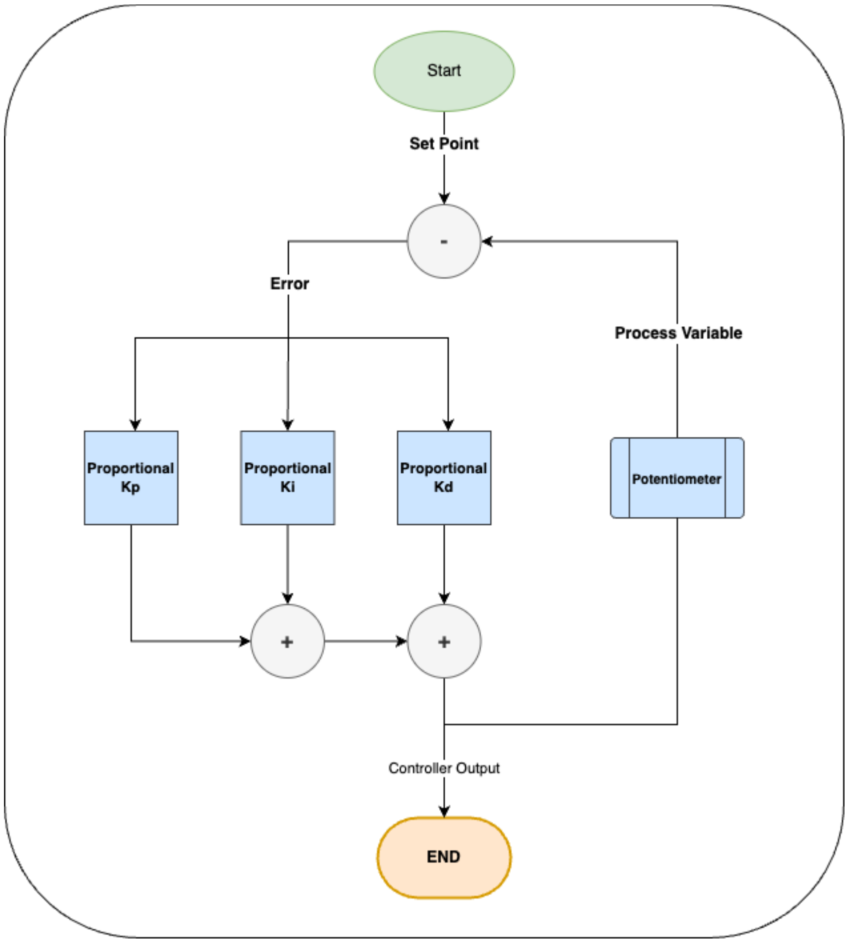
i. Keeping Kp and Kd to 0 and 0, respectively, change ki to 0, 0.5, 1, 5, 10

When Kp and Kd are set to be 0 and 10
Ki = 0 : LED doesn't turn ON at any point
Ki = 0.5 : LED turns ON slowly (from less brightness to the high brightness)
Ki = 1 : LED turns ON slowly (from less brightness to the high brightness)
Ki = 5 : LED turns ON slowly (from less brightness to the high brightness)
Ki = 10 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON

ii. Keeping Kp and Ki to 0 and 10, respectively, change kd to 0, 0.5, 1, 5, 10
When the Kp and Ki are set to be 0 and 10
Kd = 0 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON.
Kd = 0.5 : LED does not turn OFF at any point and the output value will have only two values (0.00 and 255.00)
Kd = 1 : LED does not turn OFF at any point and the output value will have only two values (0.00 and 255.00)
Kd = 5 : LED does not turn OFF at any point and the output value will have only two values (0.00 and 255.00)
Kd = 10 : LED does not turn OFF at any point and the output value will have only two values (0.00 and 255.00)

iii. Keeping Kd and Ki to 0 and 10, respectively, change kp to 0, 0.5, 1, 5, 10
When Kd and Ki are set to be 0 and 10
Kp = 0 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON
Kp = 0.5 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON
Kp = 1 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON
Kp = 5 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON
Kp = 10 : Normal functionality i.e., when pot is at low point (0.00) LED doesn't turn ON and while moving to high point (255.00) then LED turns ON

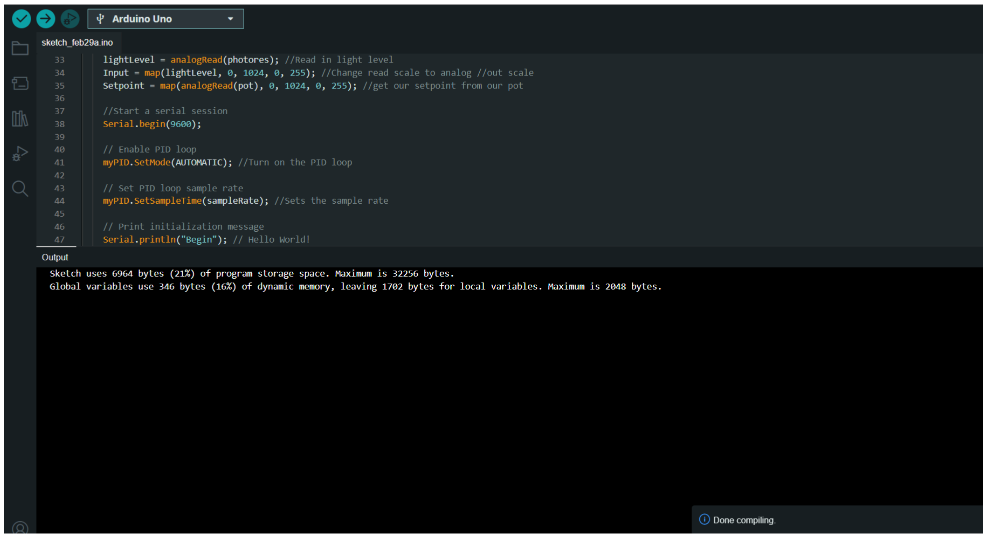
C. How does the rate PID loop reacts to environmental changes of light and changes to the set point? Show this using a flow chart.



D. **How DIRECT (direction) effect the operation? What is the value for this? Invert the DIRECT value and observe what happens. Explain your observation.**
   - A PID controller's "DIRECT" parameter establishes the direction of the control action.
   - It indicates whether a rise in the control output should cause the process variable to rise or fall.
   - Normally, the "DIRECT" parameter's value is either 1 or -1.
   - Direction is represented by 1 for DIRECT and -1 for REVERSE.
   - When the pot is adjusted in the opposite direction from the direction in which the DIRECT value is inverted, or REVERSE, the LED turns ON at the output value of 0.00 and OFF at the output value of 255.00.

d) **Provide snapshots confirming successful download of the code.**

Compiling Snapshot:

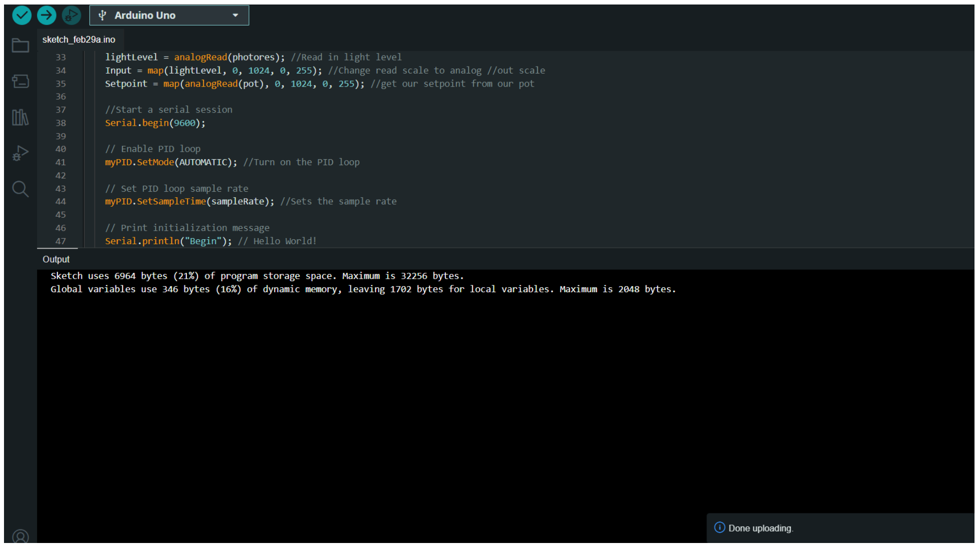


```
lightLevel = analogRead(photores); //Read in light level
Input = map(lightLevel, 0, 1024, 0, 255); //Change read scale to analog //out scale
Setpoint = map(analogRead(pot), 0, 1024, 0, 255); //get our setpoint from our pot

//Start a serial session
Serial.begin(9600);

// Enable PID loop
myPID.SetMode(AUTOMATIC); //Turn on the PID loop

// Set PID loop sample rate
myPID.SetSampleTime(sampleRate); //Sets the sample rate

// Print initialization message
Serial.println("Begin"); // Hello World!
```

```
Sketch uses 6964 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.
```

ⓘ Done compiling.

Upload Snapshot:



```
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

//Start a serial session
Serial.begin(9600);

// Enable PID loop
myPID.SetMode(AUTOMATIC); //Turn on the PID loop

// Set PID loop sample rate
myPID.SetSampleTime(sampleRate); //Sets the sample rate

// Print initialization message
Serial.println("Begin"); // Hello World!
```

```
Sketch uses 6964 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.
```

ⓘ Done uploading.

e) Provide pictures with labels showing your setup and proper functionality.

Setup:

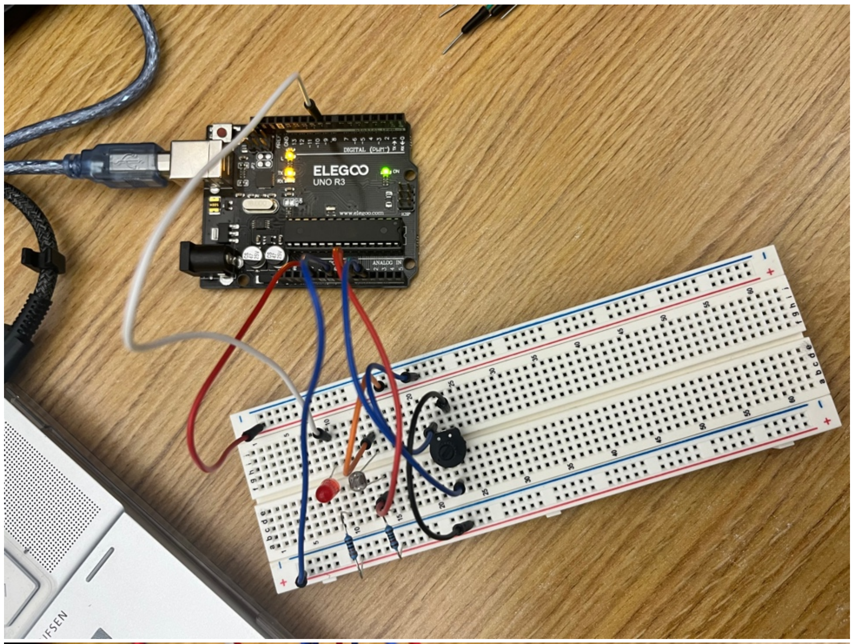
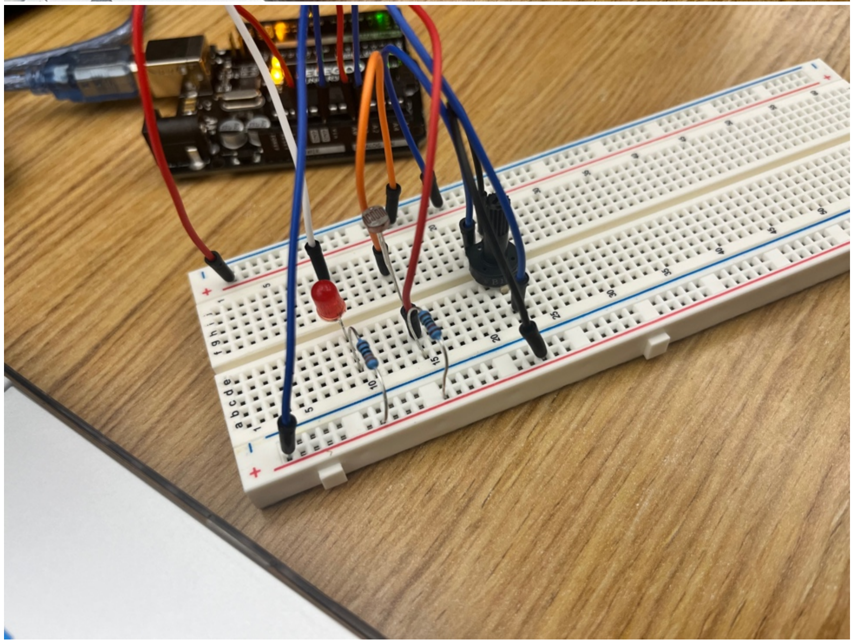
Components: An LDR (light dependent resistor), a POT (potentiometer or variable resistor), a red LED, a 220 (or 330) Ω resistor, a 10 kΩ resistor, a breadboard, and some jumper wires.

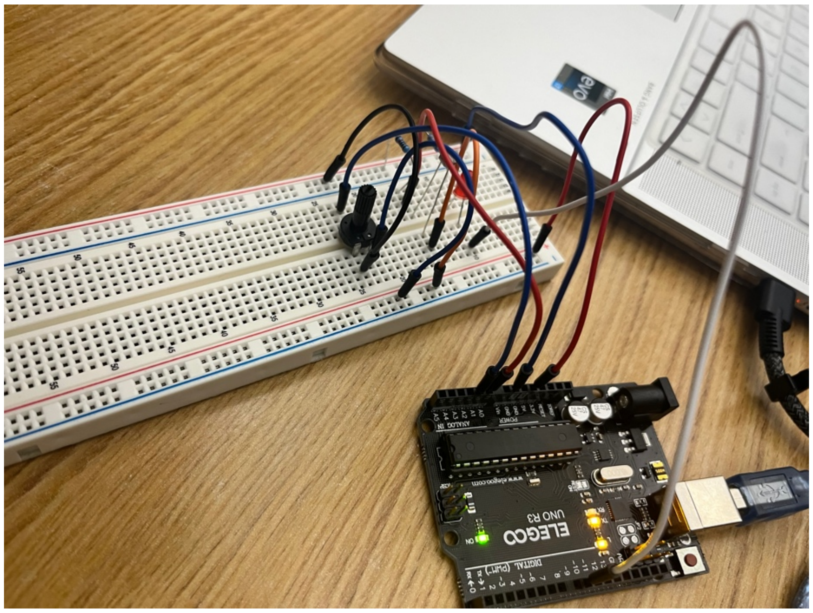
Connection:

- Photoresistor in series with 10 kΩ resistor, then connect the open end of the photoresistor to 3.3 V and the open end of the resistor to 0 V (GND).
- Then connect the common point to A0 input.
- Connectonesideof the POTto3.3Vandtheother side to 0 V (GND).
- Then connect the middle point to A1 input.
- ConnectanLEDinserieswitharesistor (220 Ω or 330Ω)such that negative terminal of the LED connect to the resistor.

- Connect the open positive terminal of the LED to digital pin 9.
- Then connect the open side of the resistor to 0V (GND).

Snapshots:
Setup:


ELEGOO
UNO R3


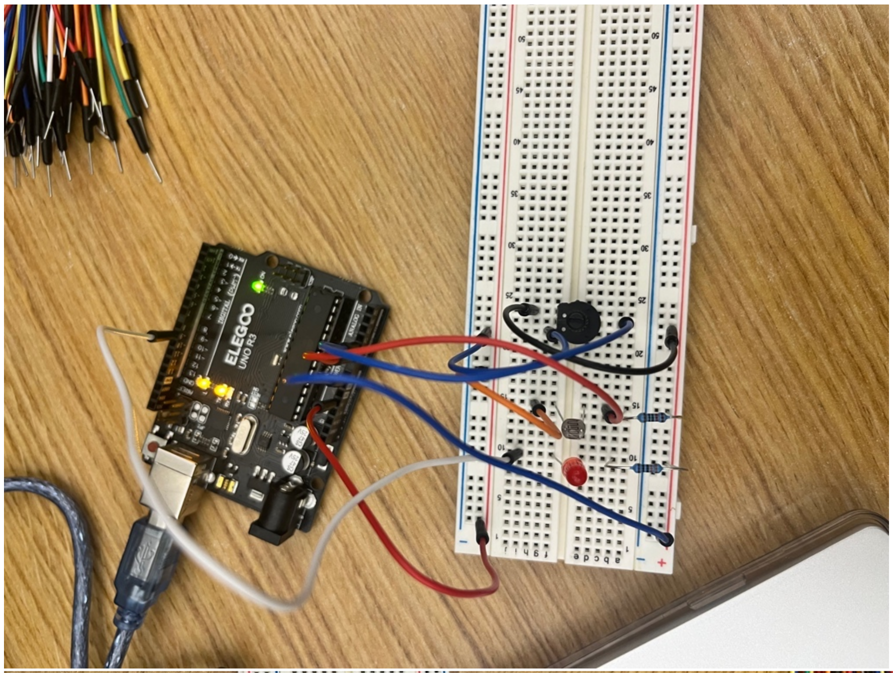


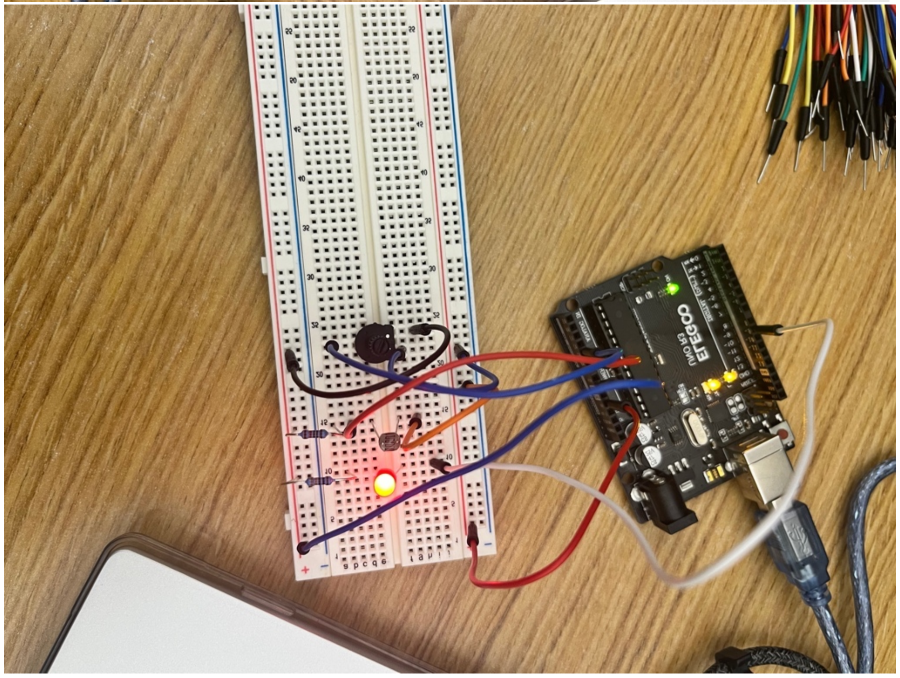
Output:

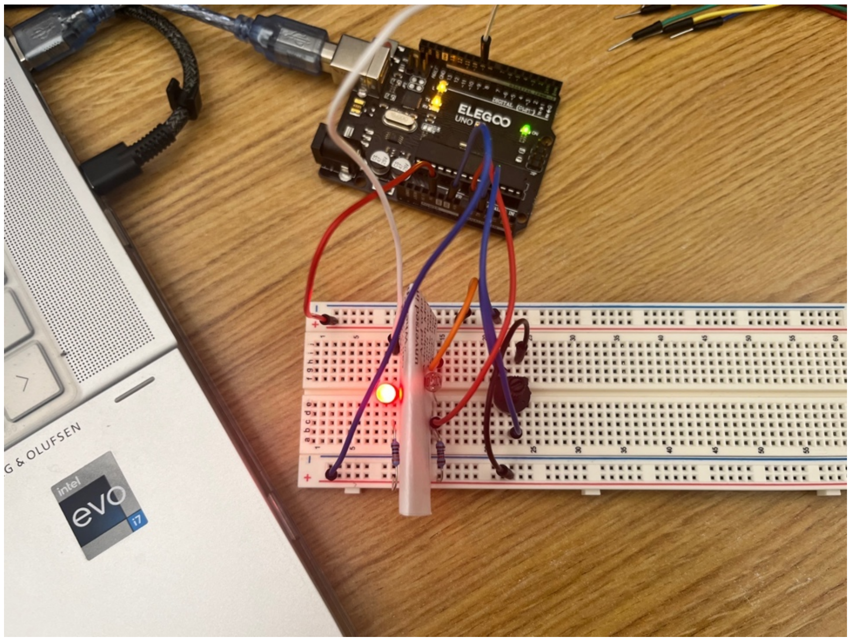
Screenshot1: When pot is at LOW, then LED doesn’t turn ON.

Screenshot2: When pot is at HIGH, then LED turns ON to its maximum.

Screenshot3: When paper is placed between LED and LDR, then LED turns ON more brightly.


ELEGOO
UNO R3


ELEGOO
UNO R3


ELEGOO
UNO R3
intel
evo
i7