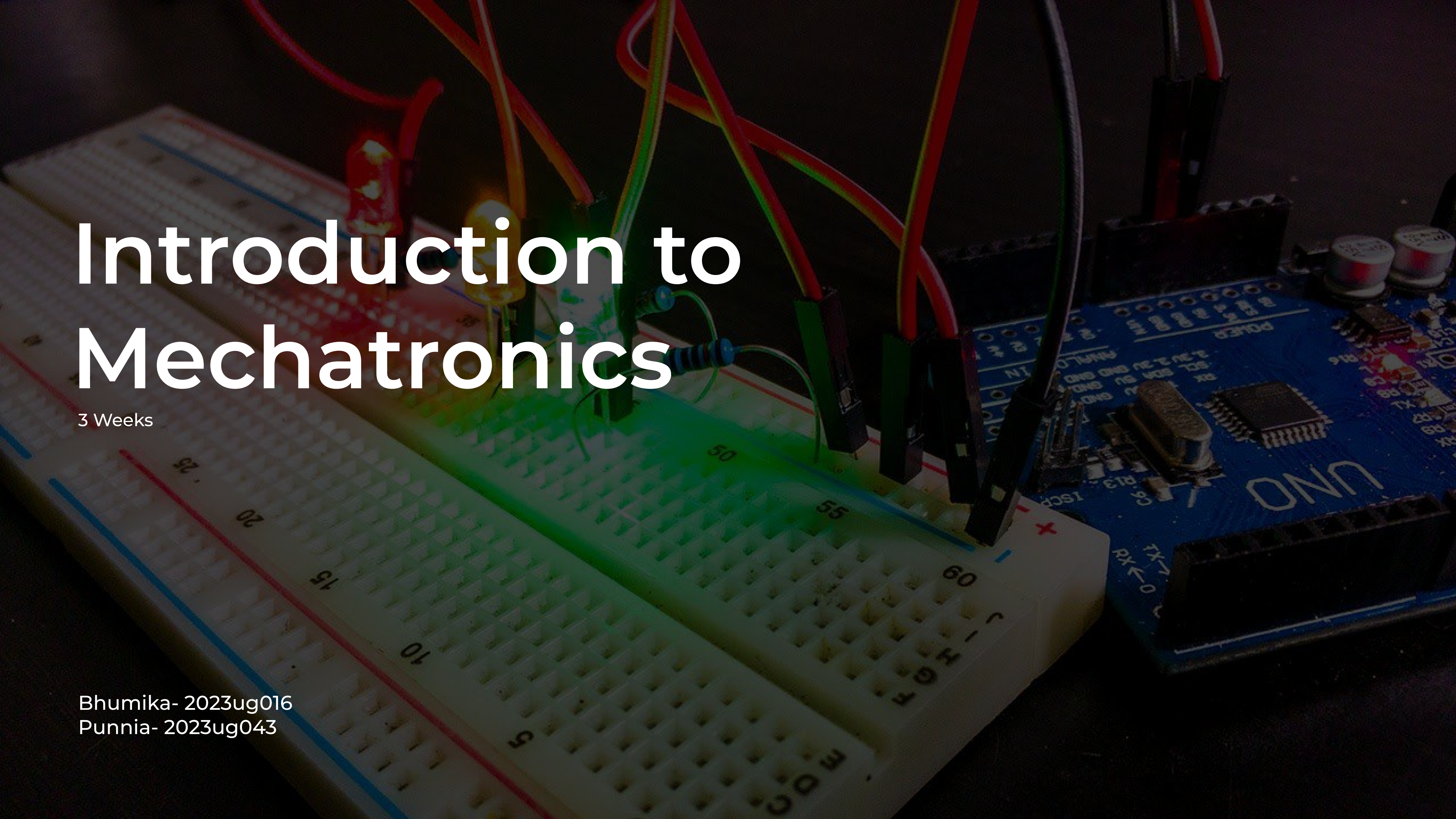


Introduction to Mechatronics

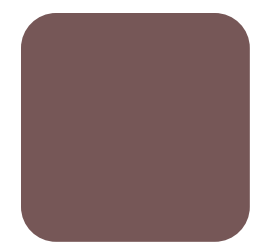


3 Weeks

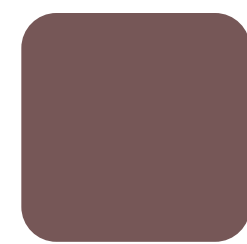
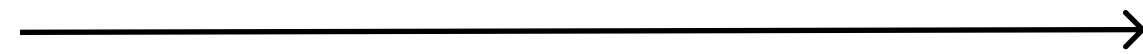
Bhumika- 2023ug016
Punna- 2023ug043

What is mechatronics

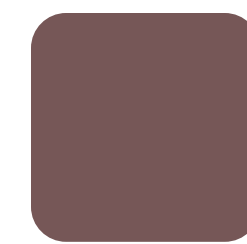
Mechatronics integrates **mechanics, electronics, and programming** to build intelligent systems.



Sensors:
Gather information
from the environment



**Controller (like a
microcontroller):**
Processes that data using
programmed instructions



Actuators:
Perform the required actions,
such as moving, turning, or
switching something on.

Components used

Sensors



ultrasonic sensor



HDT sensor

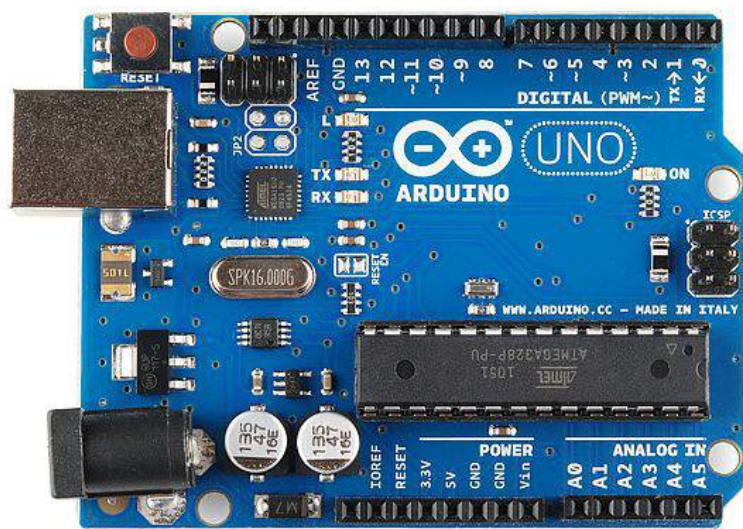


IR sensor



PIR sensor

Controller



Arduino UNO

Actuators



DC Motor



Micro servo motor



LED

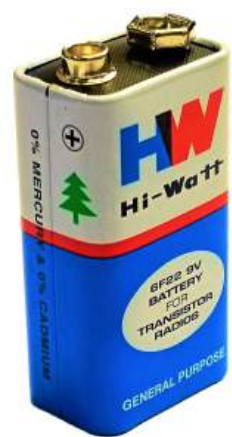


LCD monitor



3 battery pack

Power

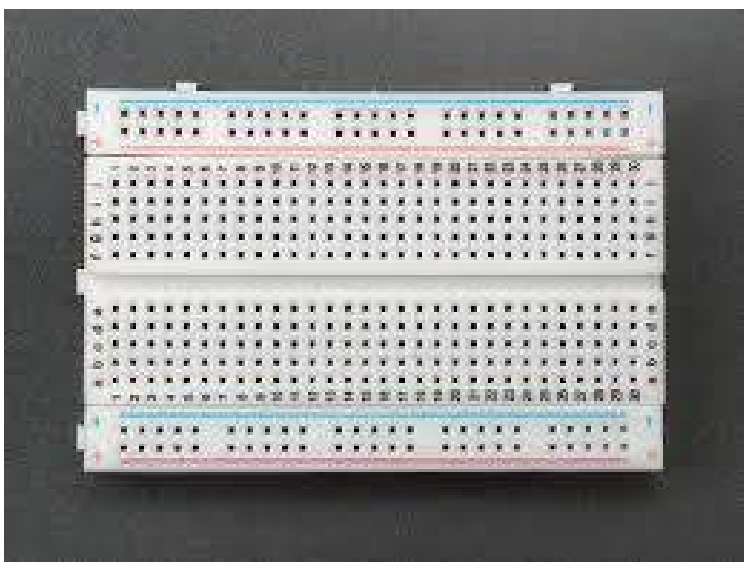


Batteries

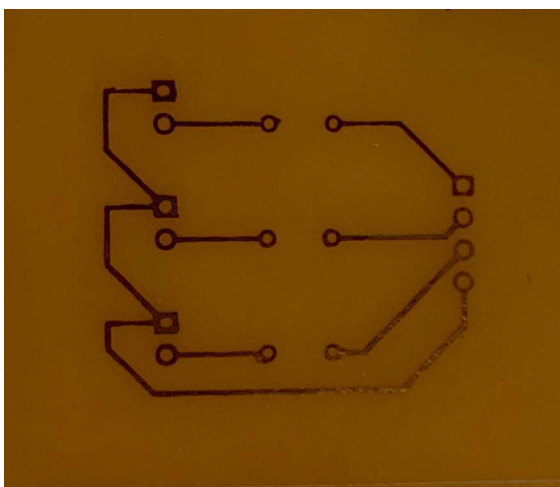


Adapter

Structure



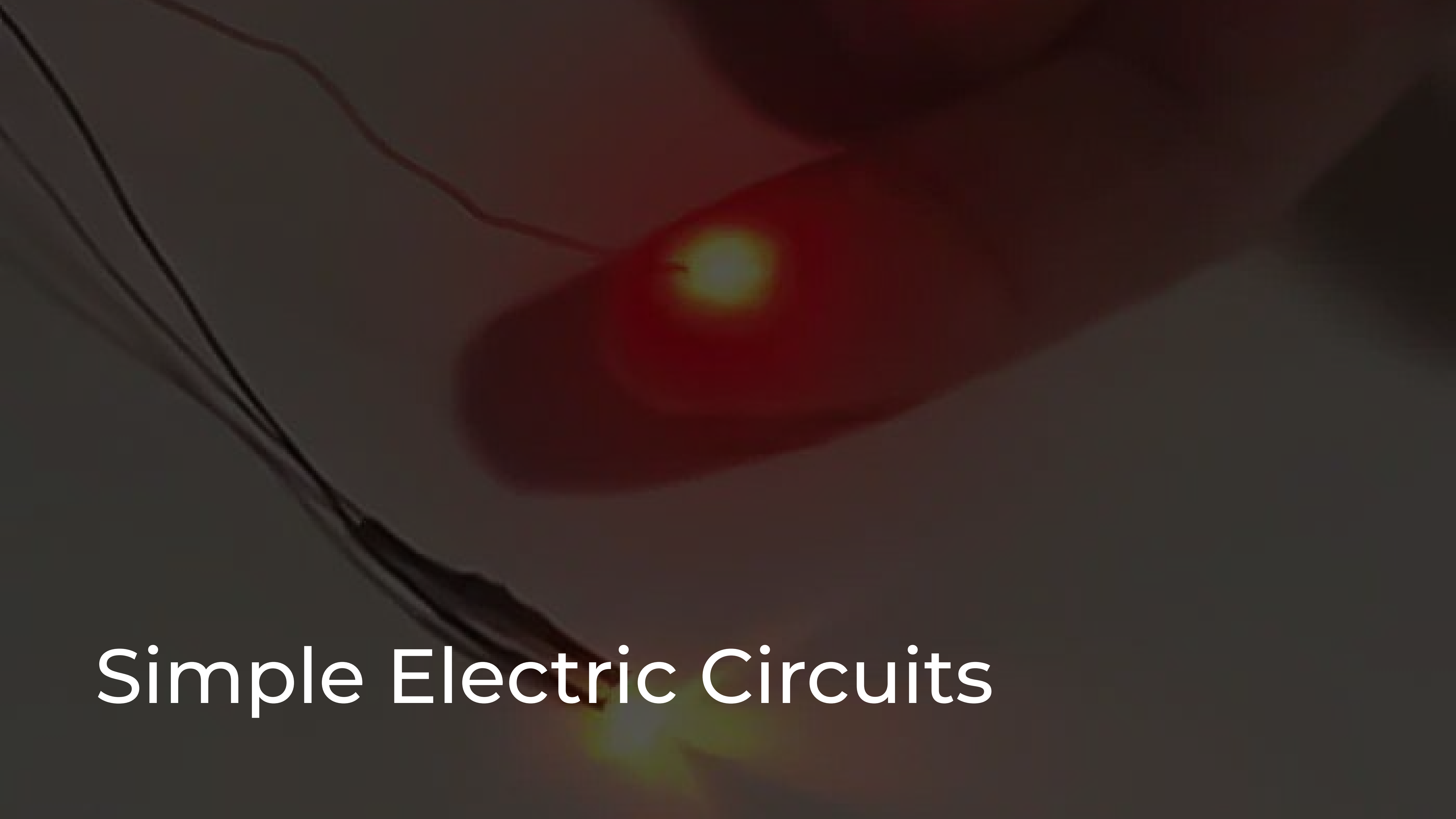
bread board



Copper cladding for circuits

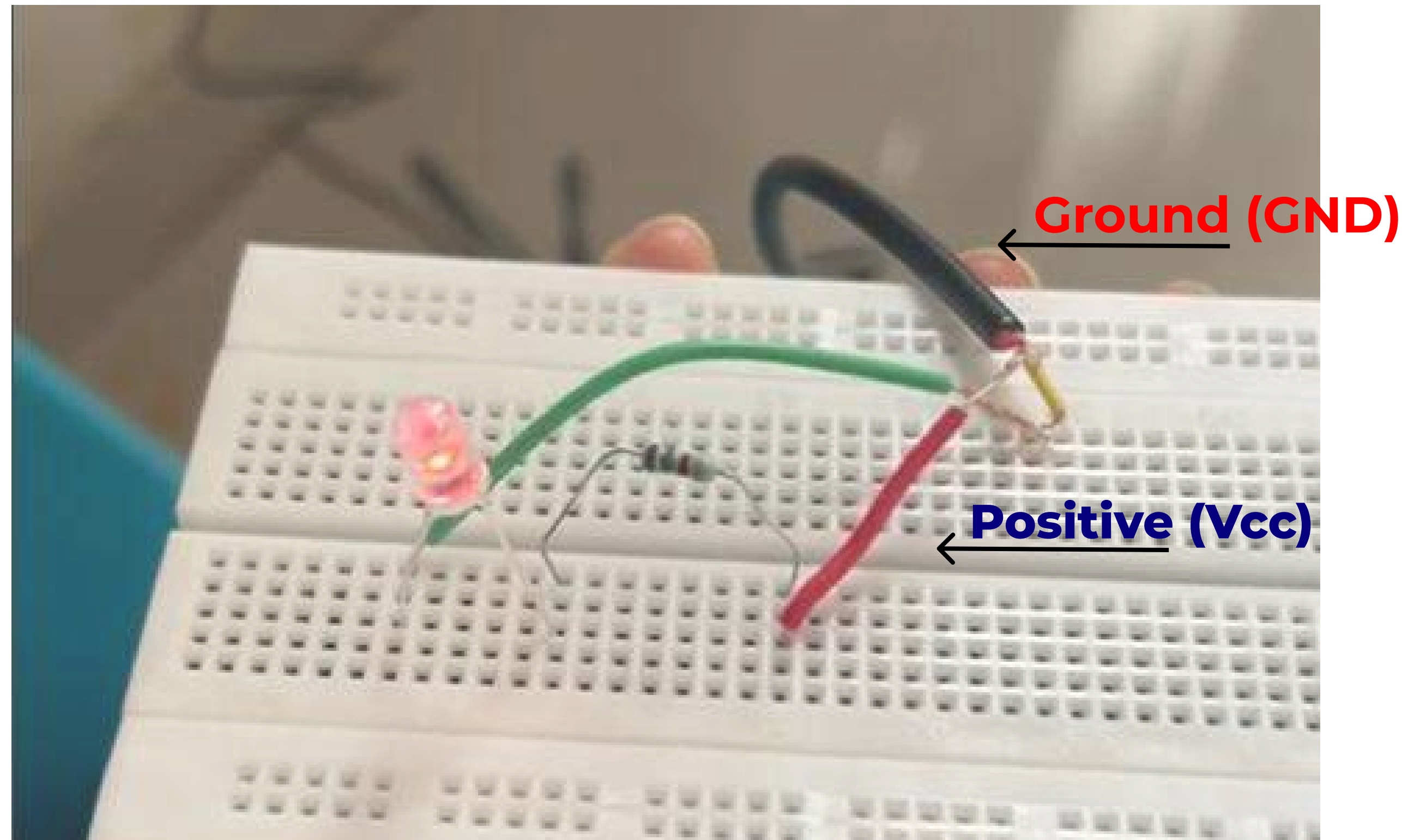
Index

1. simple electric circuits
2. arduino uno circuits
3. vero board
4. esp32 devkit v1
5. mit app inventor
6. thingspeak webserver
7. pcb circuit
8. final project

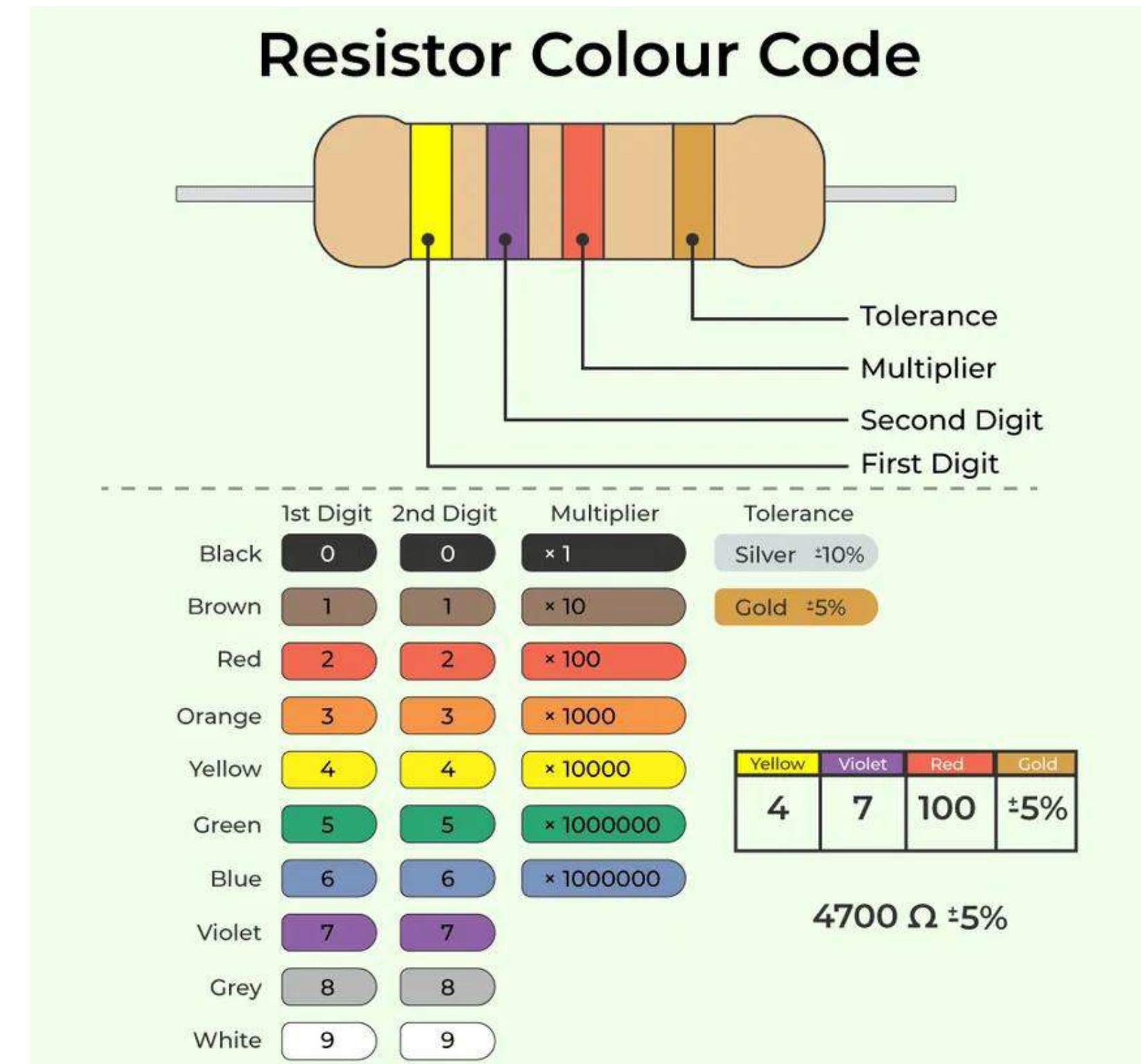


Simple Electric Circuits

LED with 10k Ω Resistor



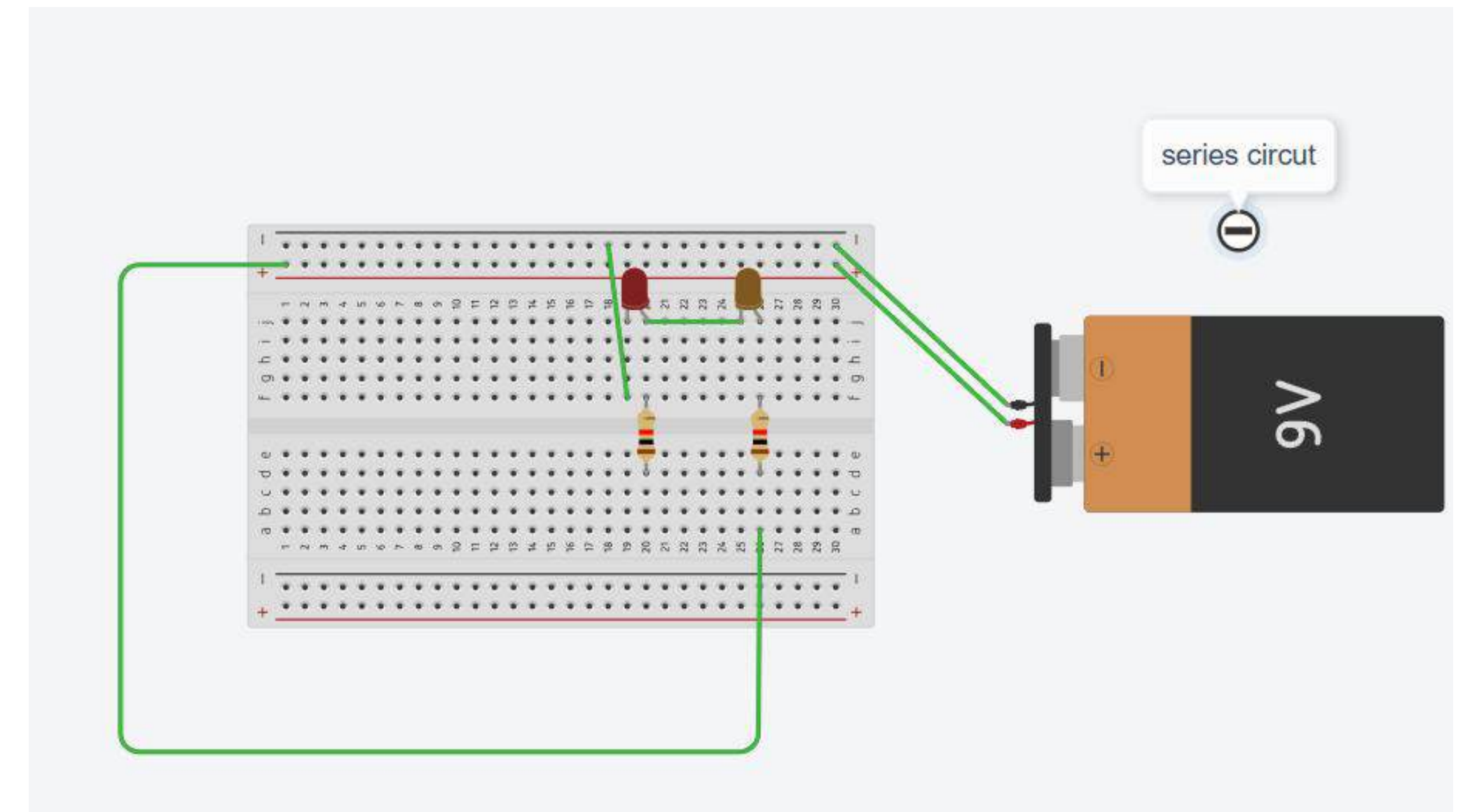
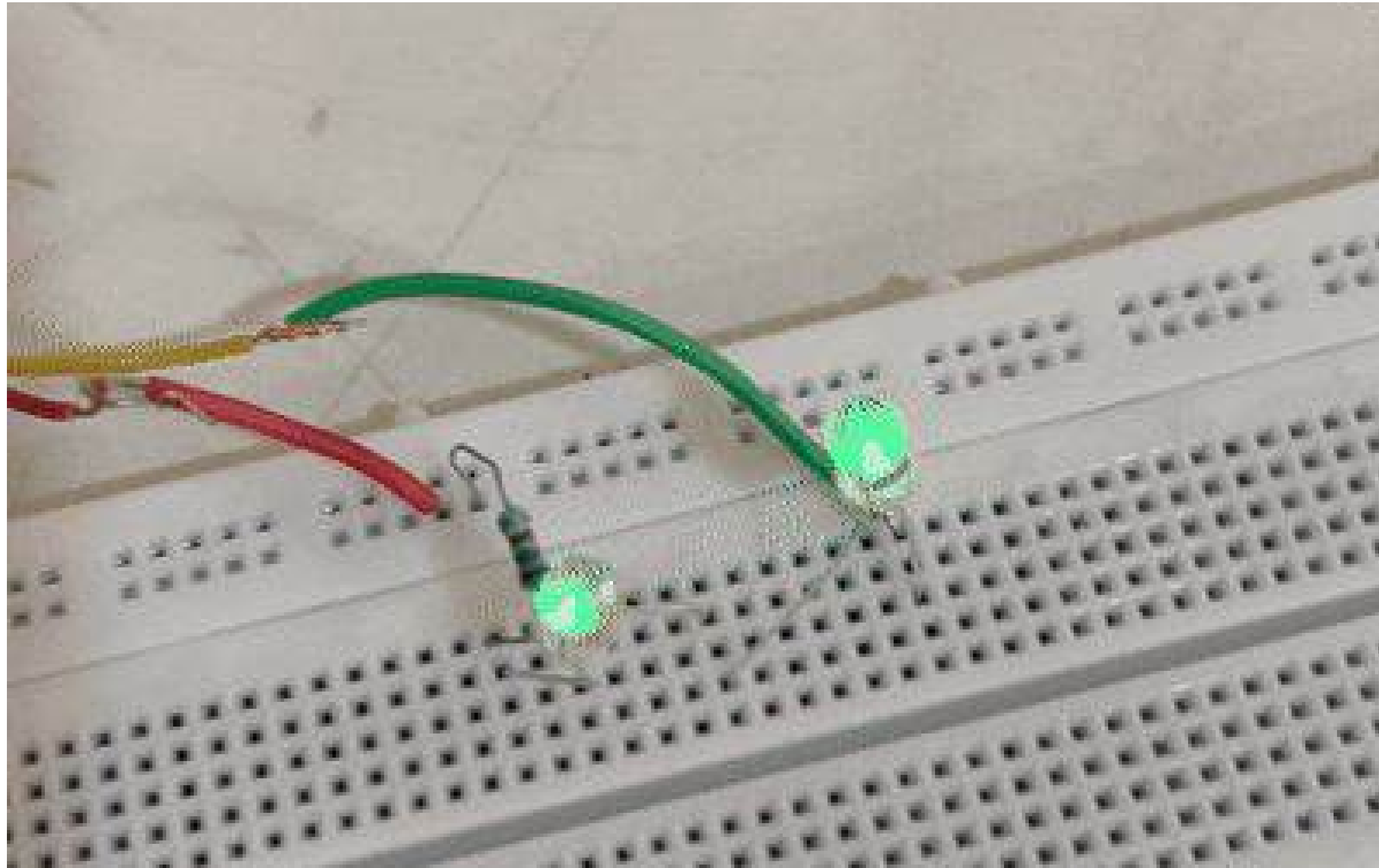
- Red LED connected in series with a resistor.
- Red wire → positive (Vcc), black wire → ground (GND).
- Resistor limits current to protect the LED.
- LED lights up when the circuit is complete.



Our Learnings:

- Built a basic LED circuit using a resistor to limit current and protect the LED.
- Read resistor values using the color code system.
- Understood the resistor's role in controlling current.
- Used a multimeter to measure voltage and current.

LED series connection

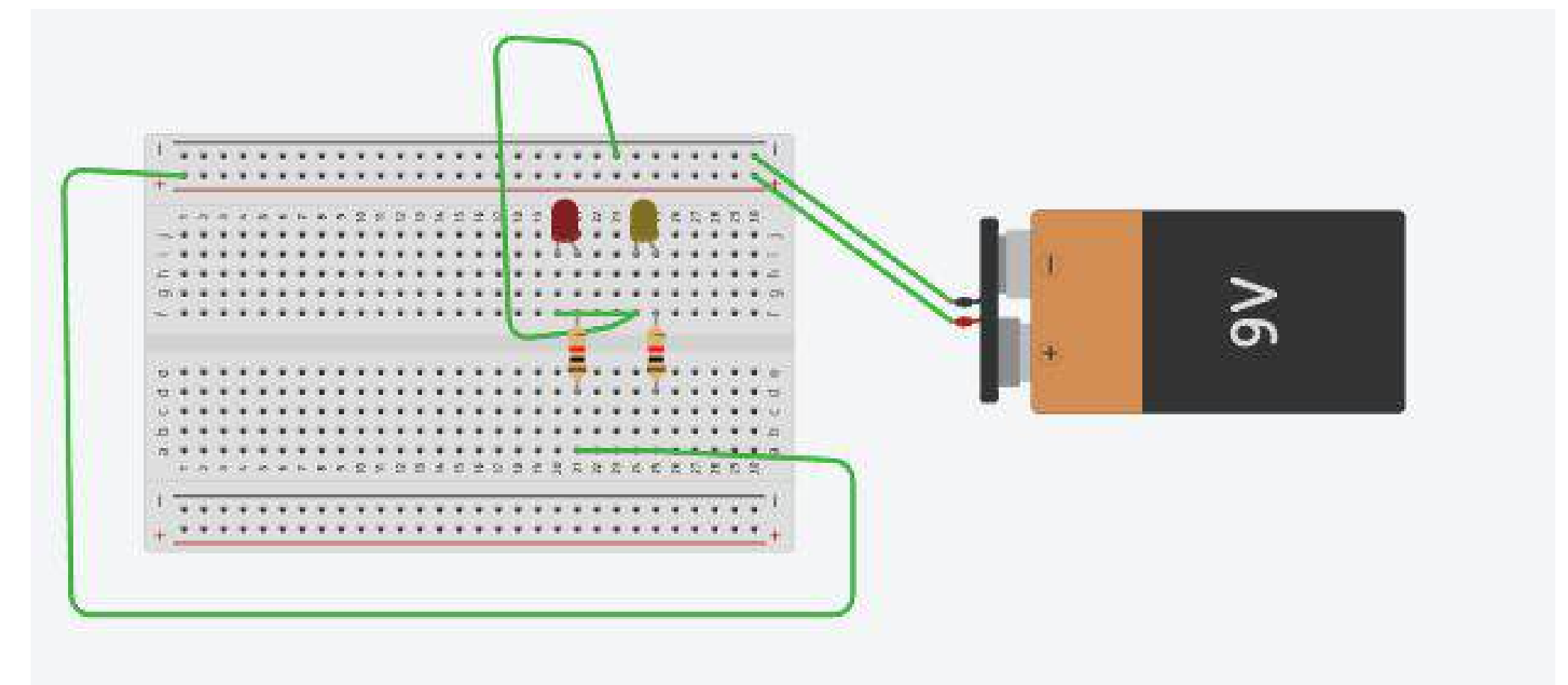
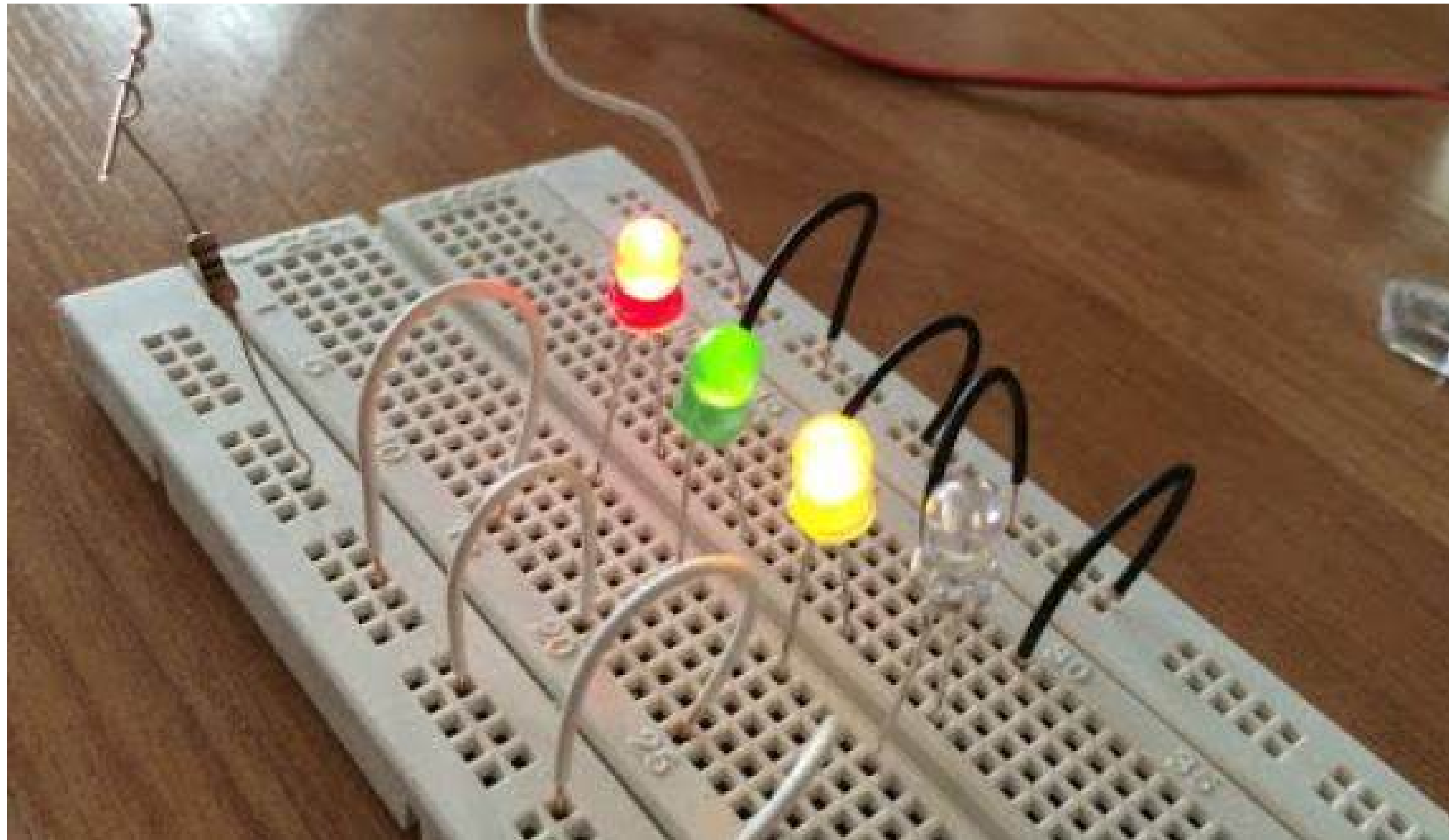


Simulation on Tinkercad

Learnings

- How to connect LEDs in a series circuit so the same current flows through all LEDs.
- Understood that in series, the total voltage is the sum of the voltage drops across each LED.
- Observed that if one LED is removed or fails, the entire circuit turns off.
- Practiced calculating total resistance in a series configuration.
- Noted that brightness may vary if the supply voltage is not sufficient for all LEDs combined.

LED parallel connection

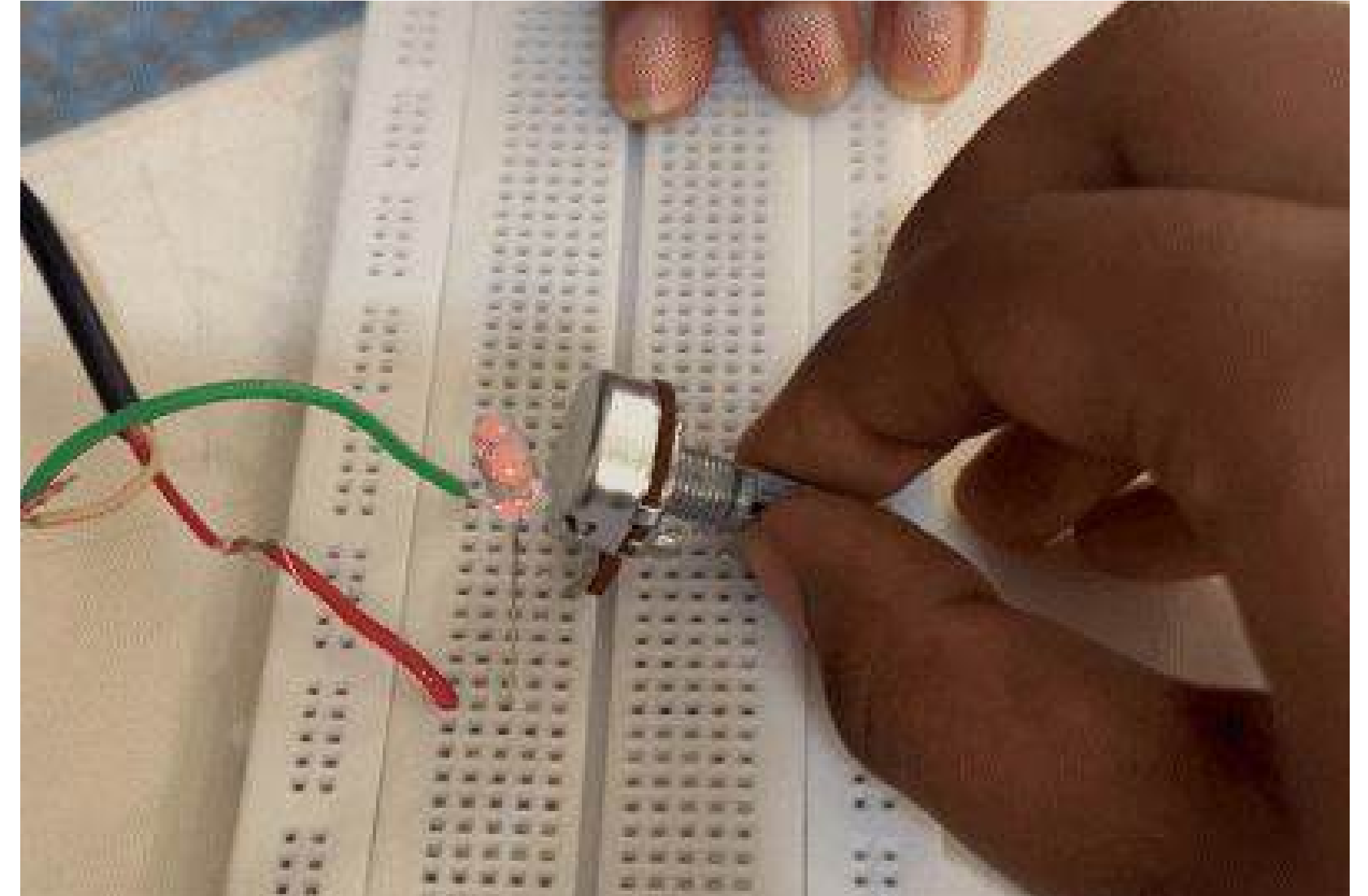
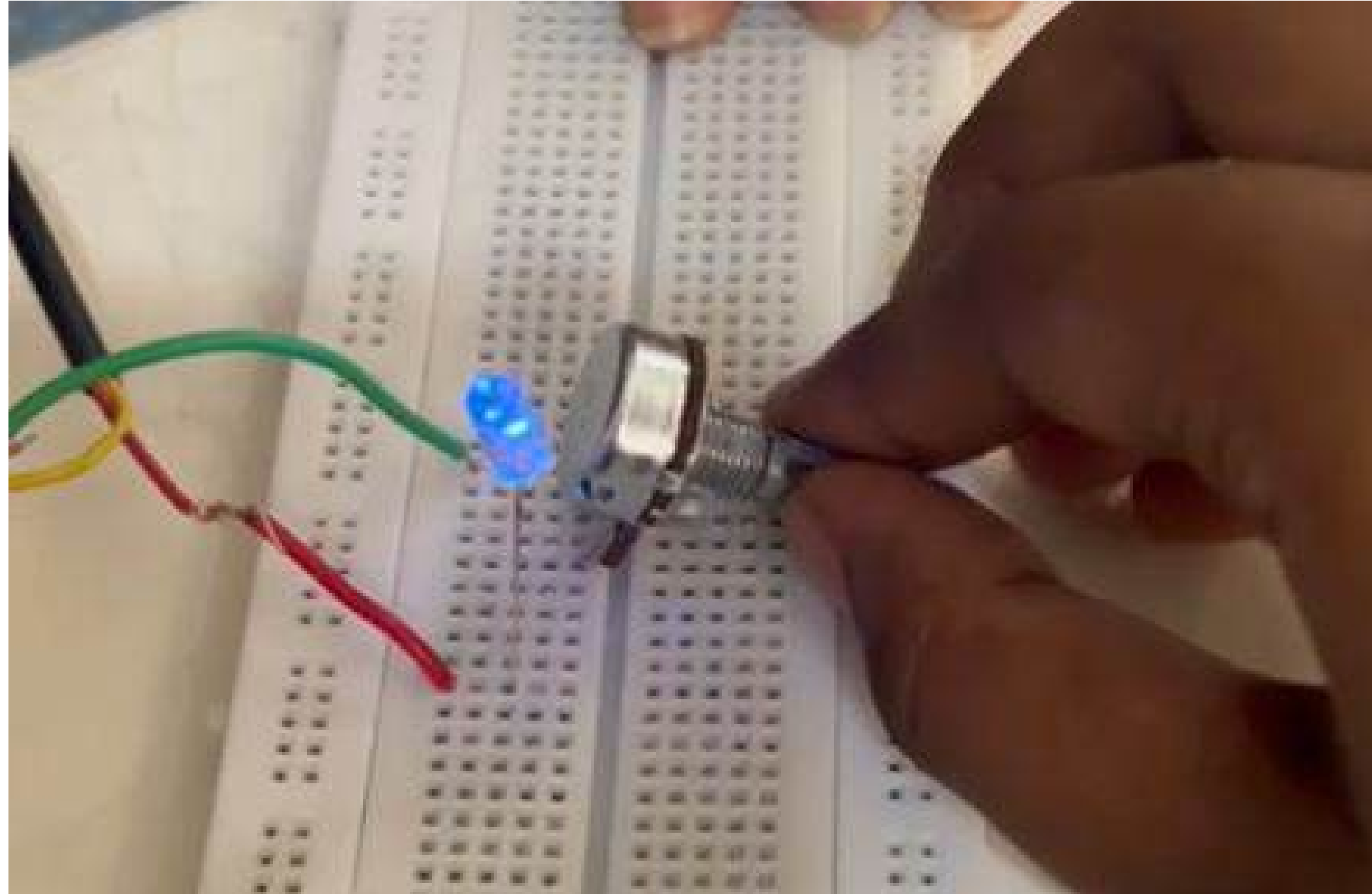


Simulation on Tinkercad

Learnings

- Wired LEDs in parallel with equal voltage.
- In a parallel connection, current divides among branches while voltage remains constant.
- Observed that if one LED fails, the others continue to glow.

LED with potentiometer



This setup demonstrates analog control of an LED's brightness using resistance.

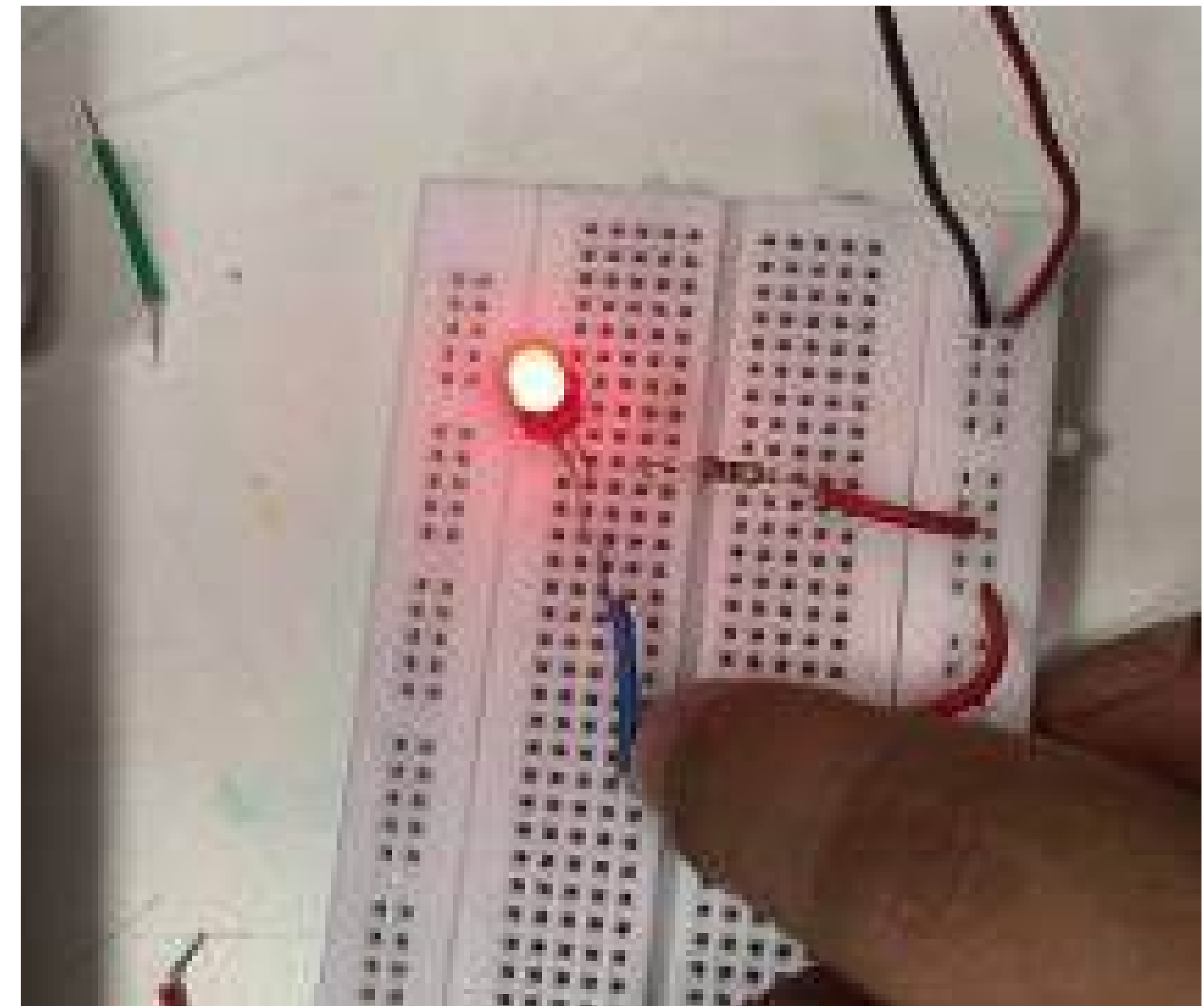
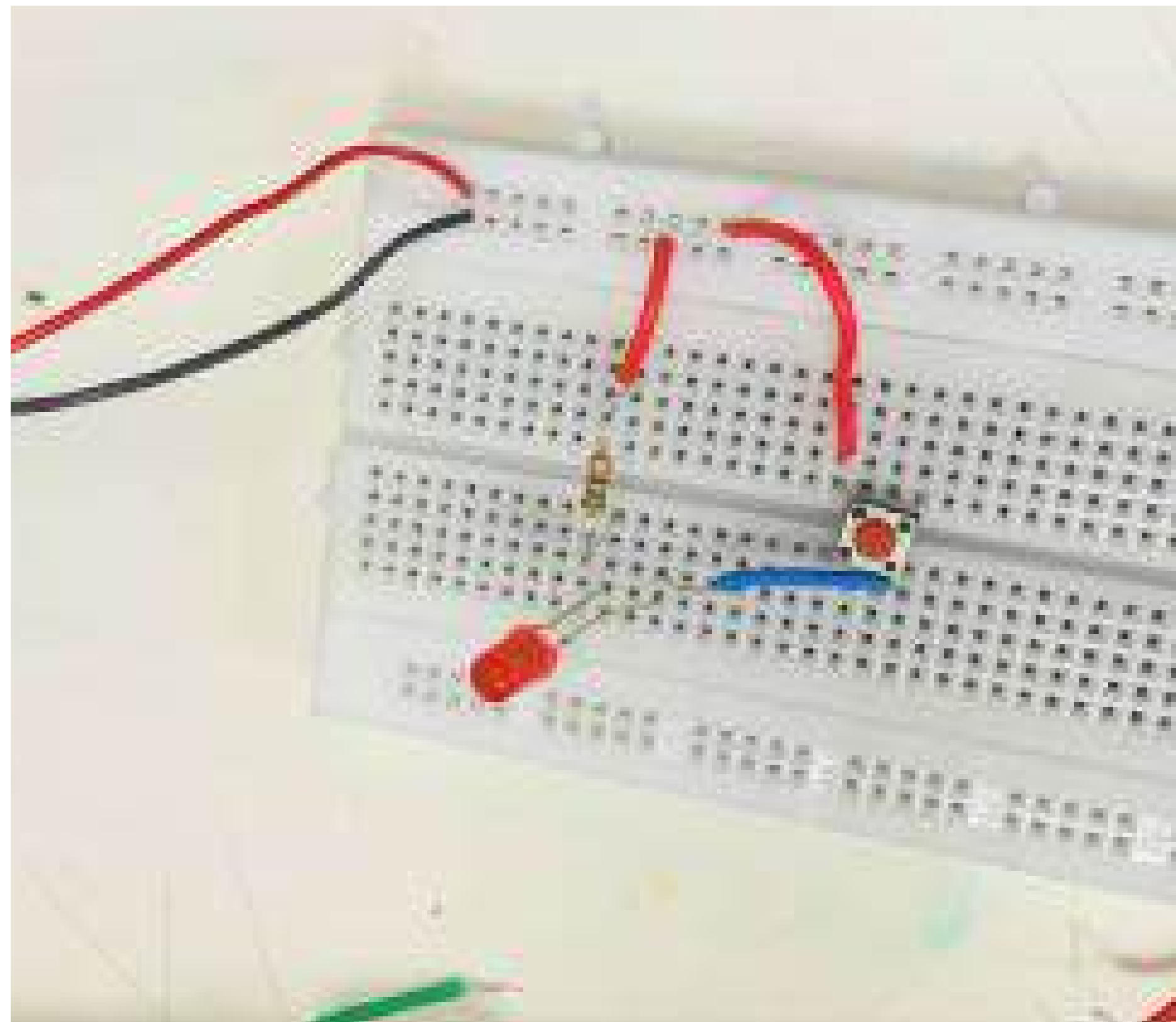
This shows a breadboard circuit where a potentiometer controls an LED's brightness.

Turning the potentiometer changes resistance, adjusting current flow — **in the video, the potentiometer controls the brightness of the led**

Learnings

- Learned how a potentiometer can be used as a variable resistor to control LED brightness.
- Understood the concept of voltage division and how it affects current through the LED.
- Practiced wiring a potentiometer in series with an LED to adjust resistance smoothly

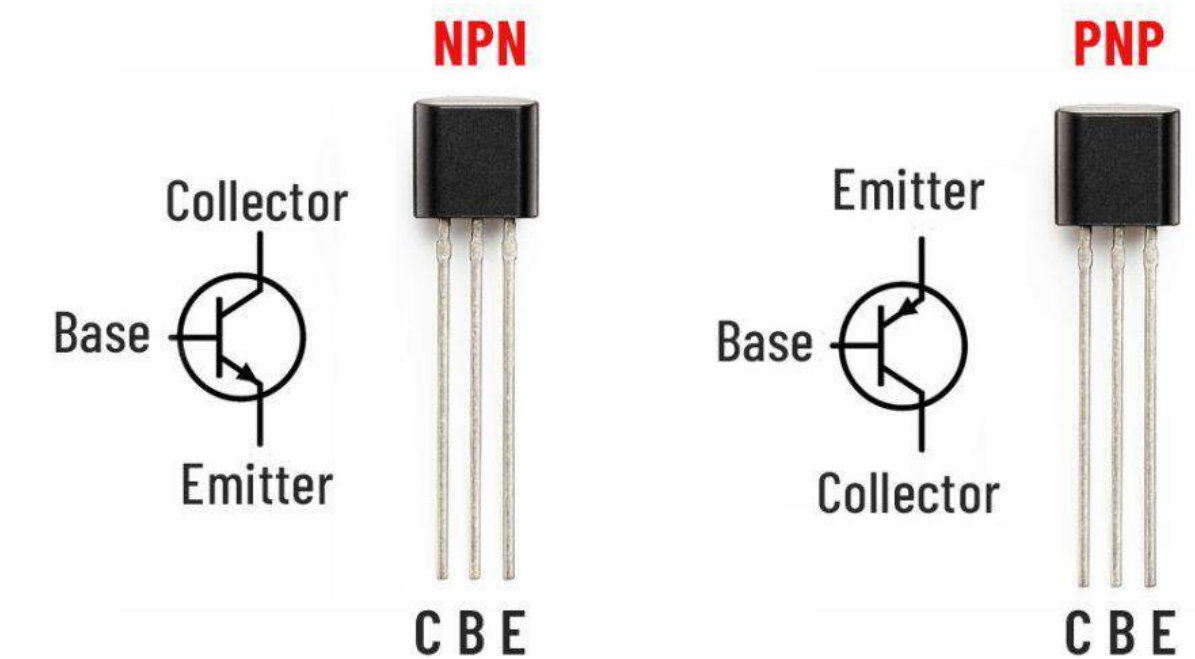
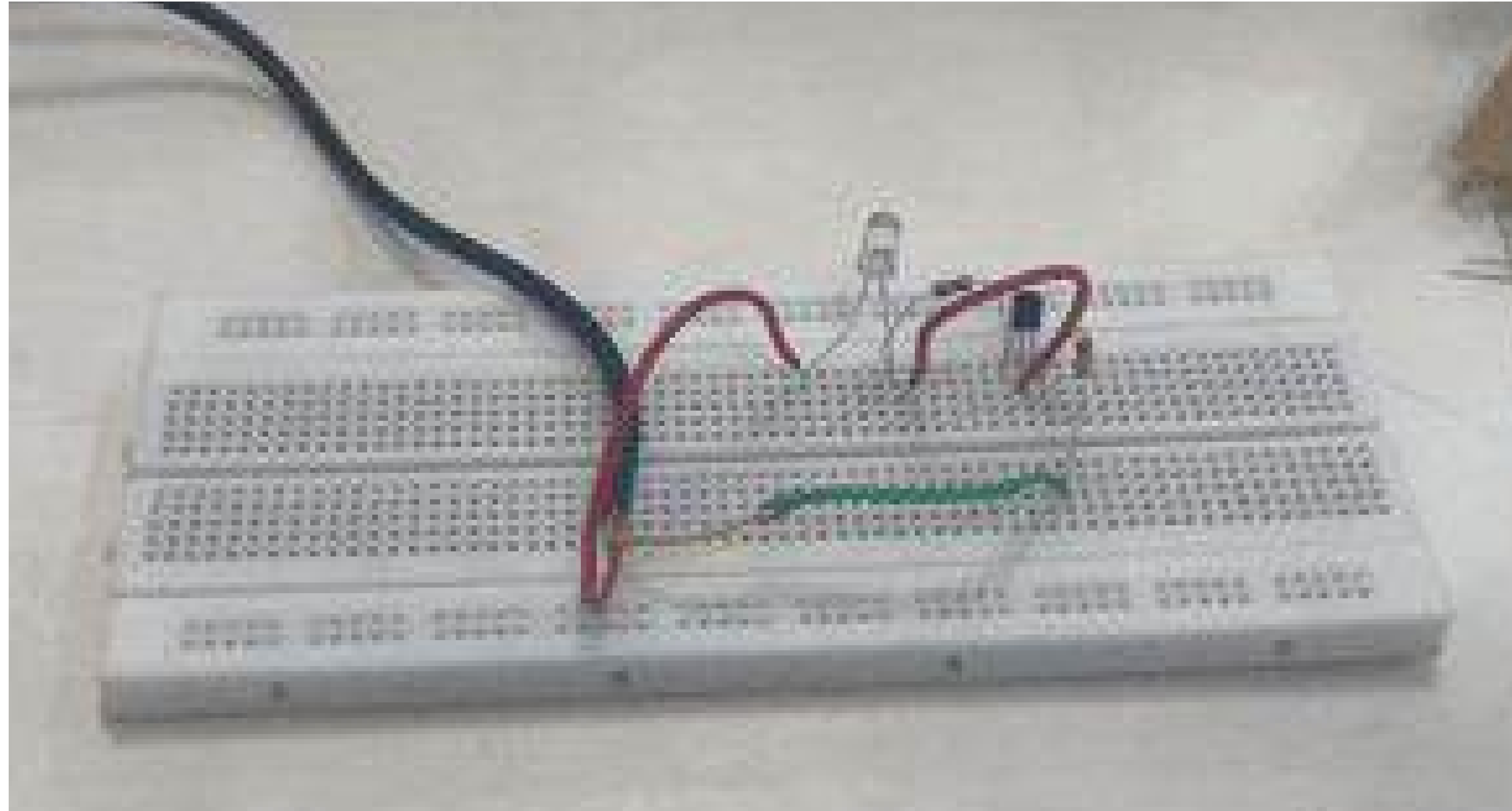
LED with push button



Learnings

- How a push button works as a simple manual switch to control an LED.
- Understood the concept of a digital input — LED turns ON when the button is pressed and OFF when released (or vice versa).
- Observed the need for a pull-down or pull-up resistor to prevent false triggering of the LED.
- Practiced wiring the button correctly so that it makes and breaks the circuit effectively.
- Gained experience in combining mechanical components (push button) with electronic components (LED + resistor).

LED WITH TRANSISTOR (NPN)



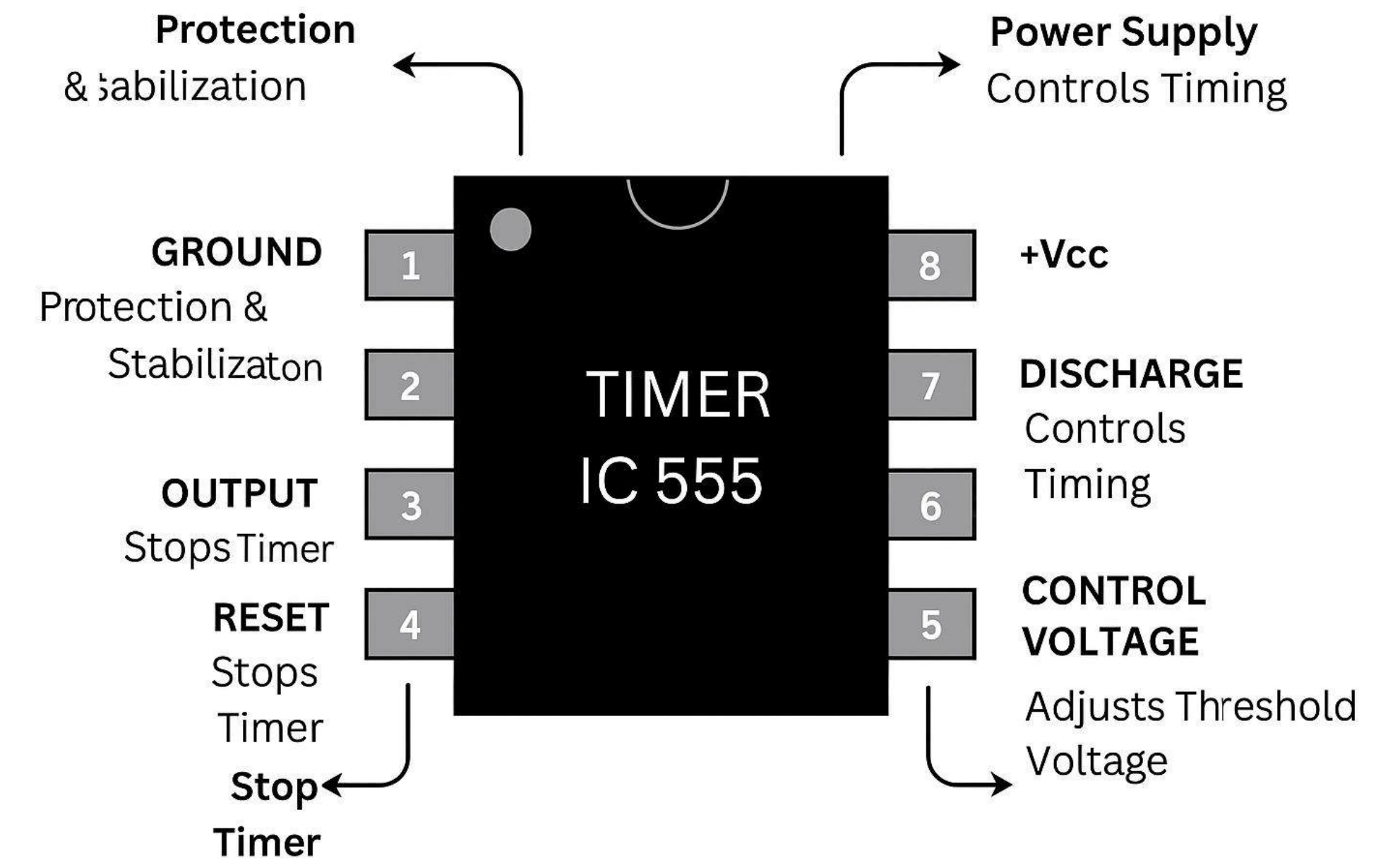
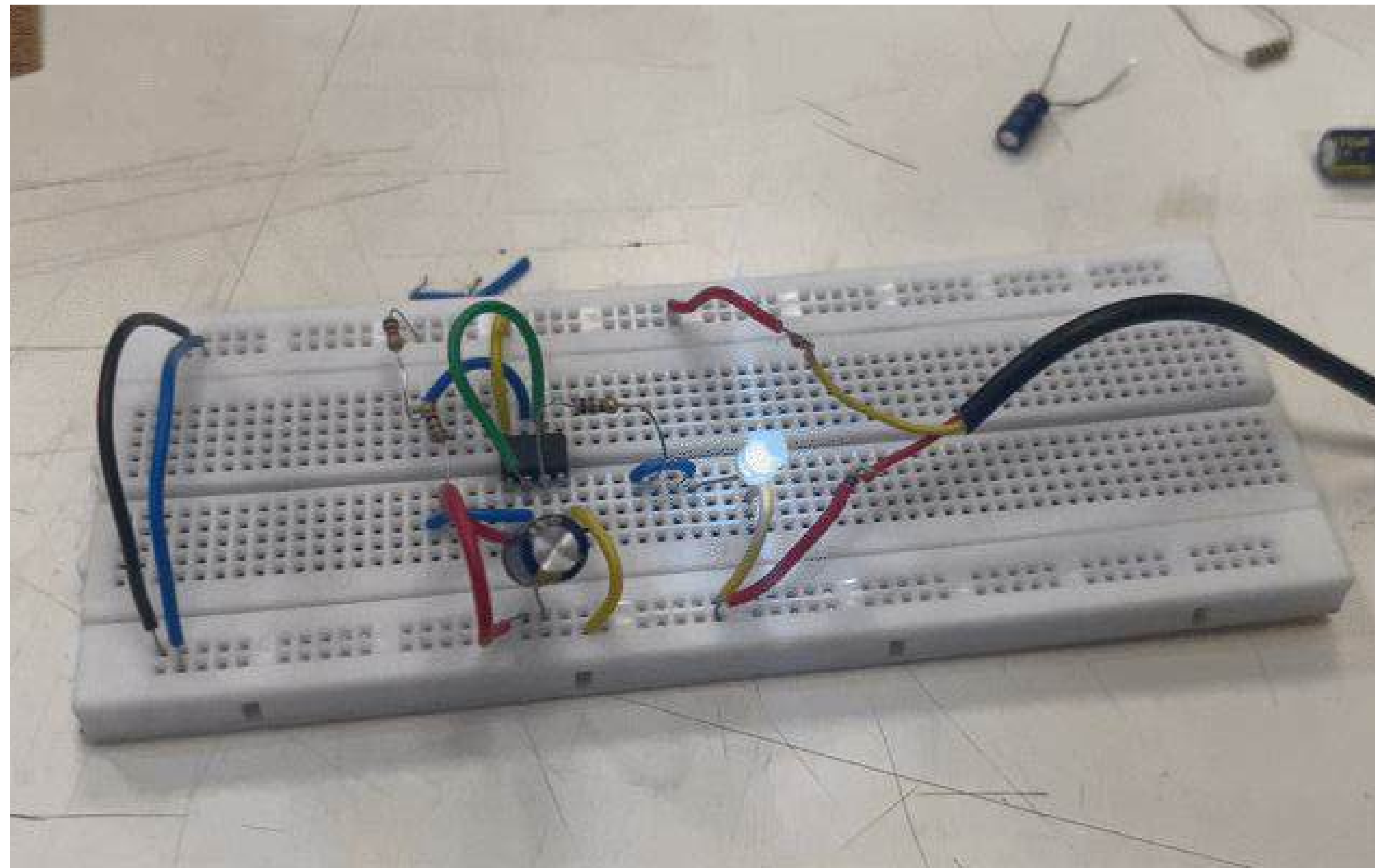
NPN transistor –
Current flows from collector to emitter when a small current is applied to the base.

PNP transistor –
Current flows from emitter to collector when the base is at a lower potential than the emitter.

Learnings

- A transistor is used as a switch or amplifier.
- The LED is connected so that it lights when the transistor is activated.
- Observed that a small base current controls a larger current through the LED.

LED WITH 555ic



Breadboard circuit with IC, resistors, capacitor, and LED

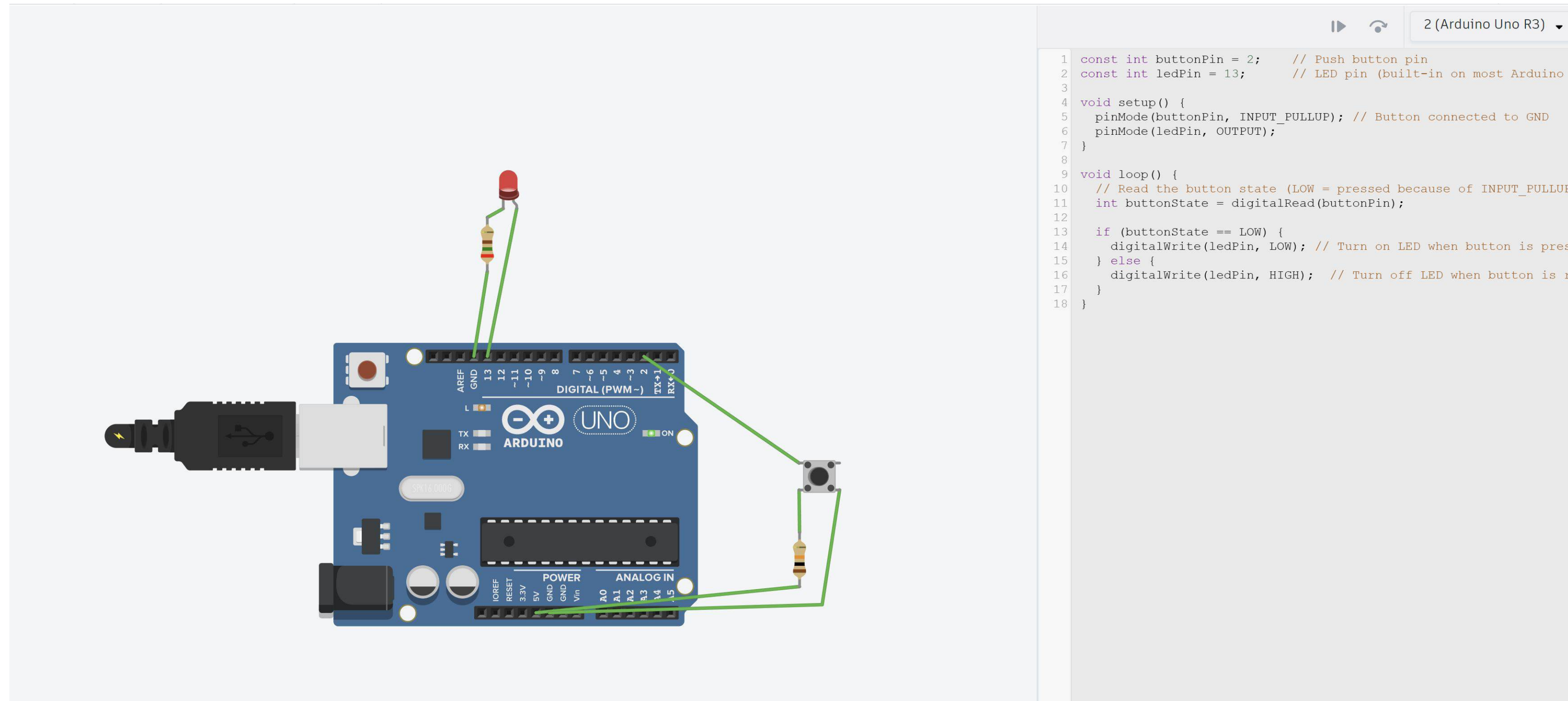
Learnings

- Proper orientation of IC (notch/pin 1 alignment) is important.
- Capacitors can be used for timing or filtering.
- Power supply polarity must be correct to avoid damaging ICs.

Arduino Uno Circuits

Tinkercad for simulations
Breadboard and Arduino Uno for prototyping

Controlling an LED with a push button.



```
1 const int buttonPin = 2;    // Push button pin
2 const int ledPin = 13;     // LED pin (built-in on most Arduino boards)
3
4 void setup() {
5   pinMode(buttonPin, INPUT_PULLUP); // Button connected to GND
6   pinMode(ledPin, OUTPUT);
7 }
8
9 void loop() {
10  // Read the button state (LOW = pressed because of INPUT_PULLUP)
11  int buttonState = digitalRead(buttonPin);
12
13  if (buttonState == LOW) {
14    digitalWrite(ledPin, LOW); // Turn on LED when button is pressed
15  } else {
16    digitalWrite(ledPin, HIGH); // Turn off LED when button is released
17  }
18 }
```

Components used:

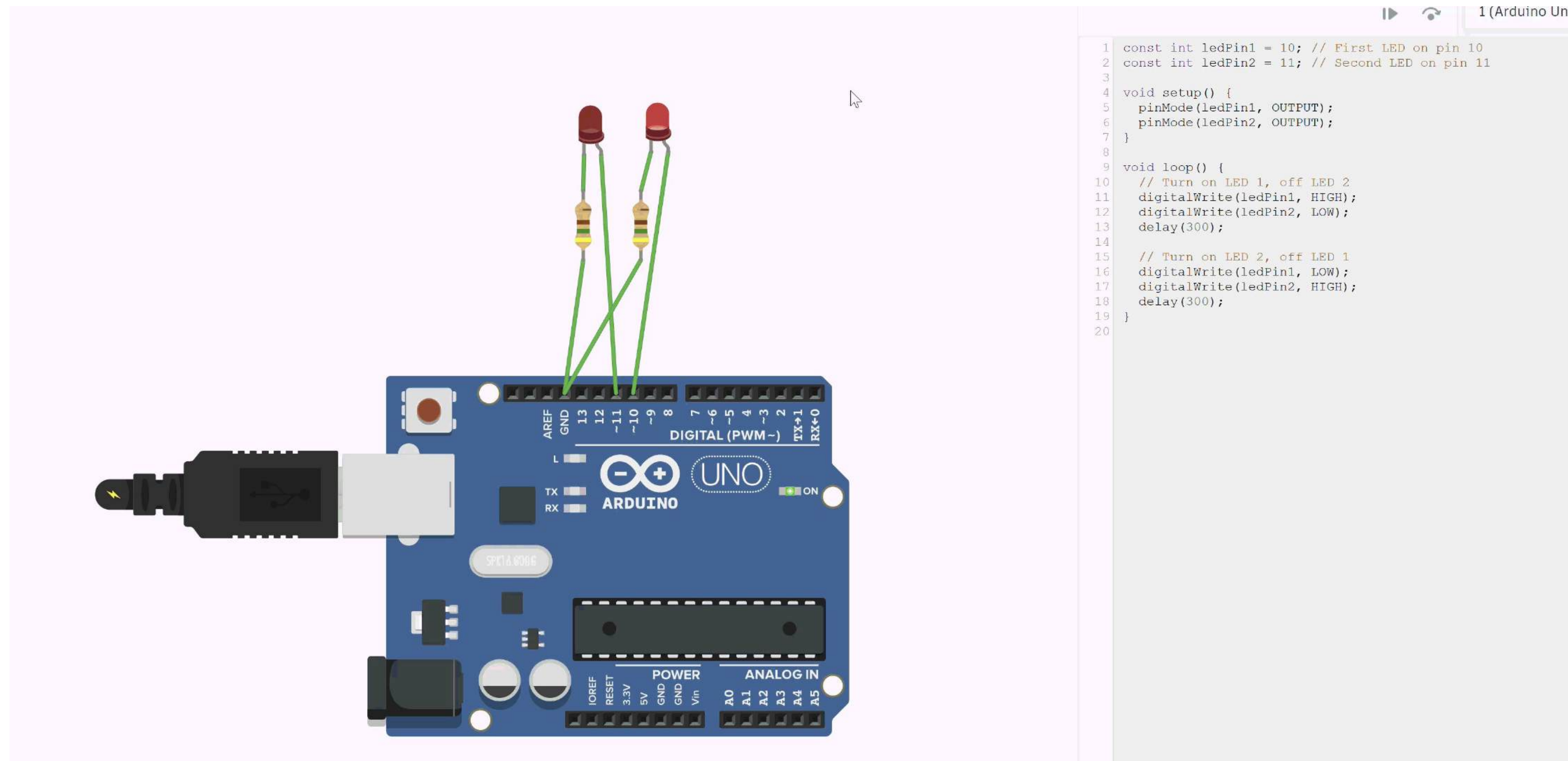
Arduino Uno
Push button with resistor (pull-down)
LED with current-limiting resistor
Jumper wires

Working: When the button is pressed, the Arduino reads the input and turns the LED ON. When released, the LED turns OFF.

Learnings:

- How to use digital input (button) and digital output (LED) with Arduino.
- Importance of resistors (to avoid damage and false signals).
- Basics of control logic (if-else with button state).

Controlling two LEDs alternately (on tinkercad)



Components used:

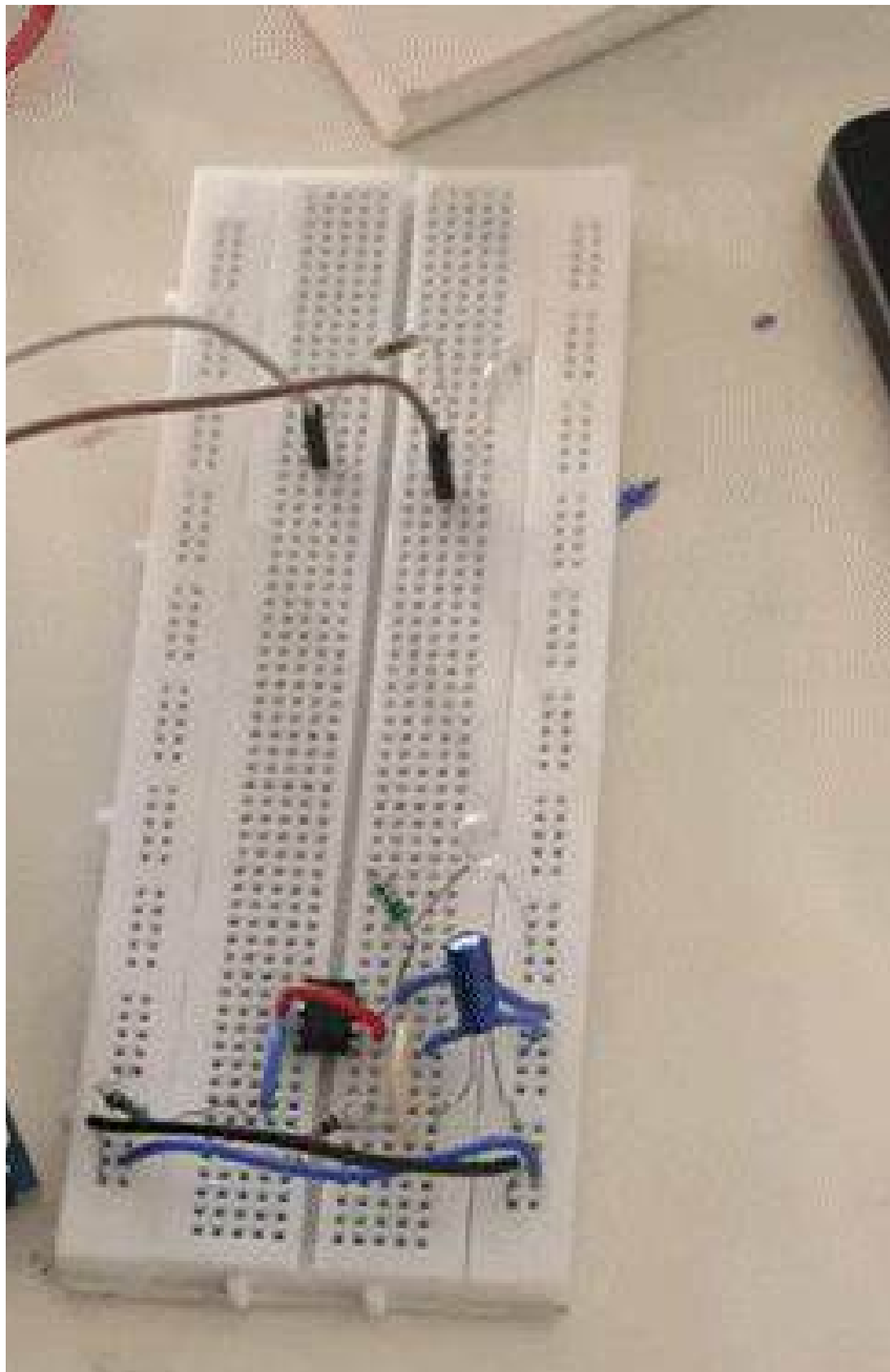
Arduino Uno
2 LEDs with current-limiting resistors
Jumper wires

Working: One LED turns ON while the other turns OFF, then they swap states repeatedly with a short delay.

Learnings:

- Using multiple digital outputs on Arduino.
- Controlling LEDs with timing (delay).
- Basics of creating blinking/alternating light patterns.

Controlling two LEDs alternately (on bread board)

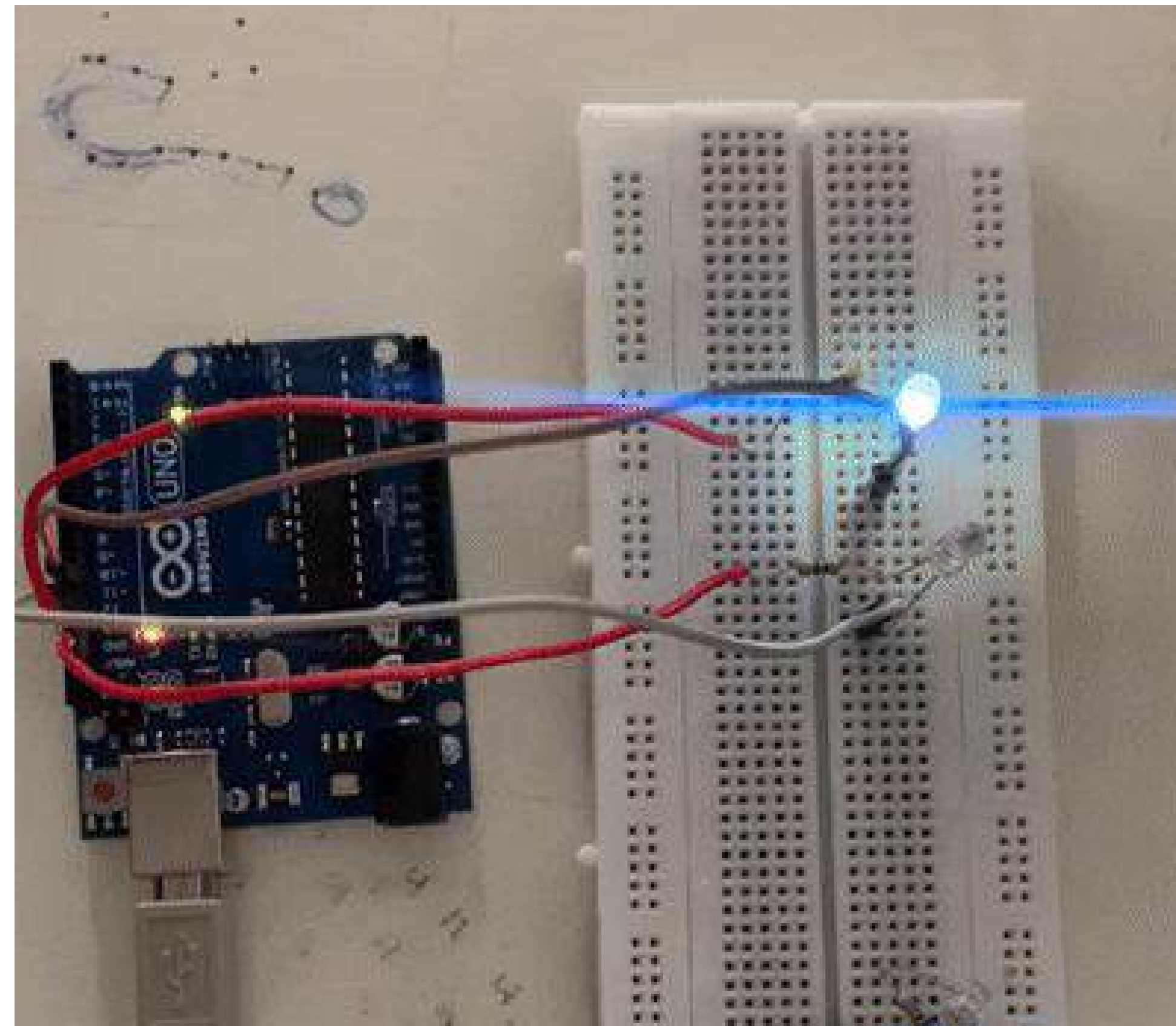


Wiring:

Anode (long leg) → Arduino digital pin via resistor

Cathode (short leg) → Arduino GND

Resistors protect LEDs, breadboard makes wiring neat.



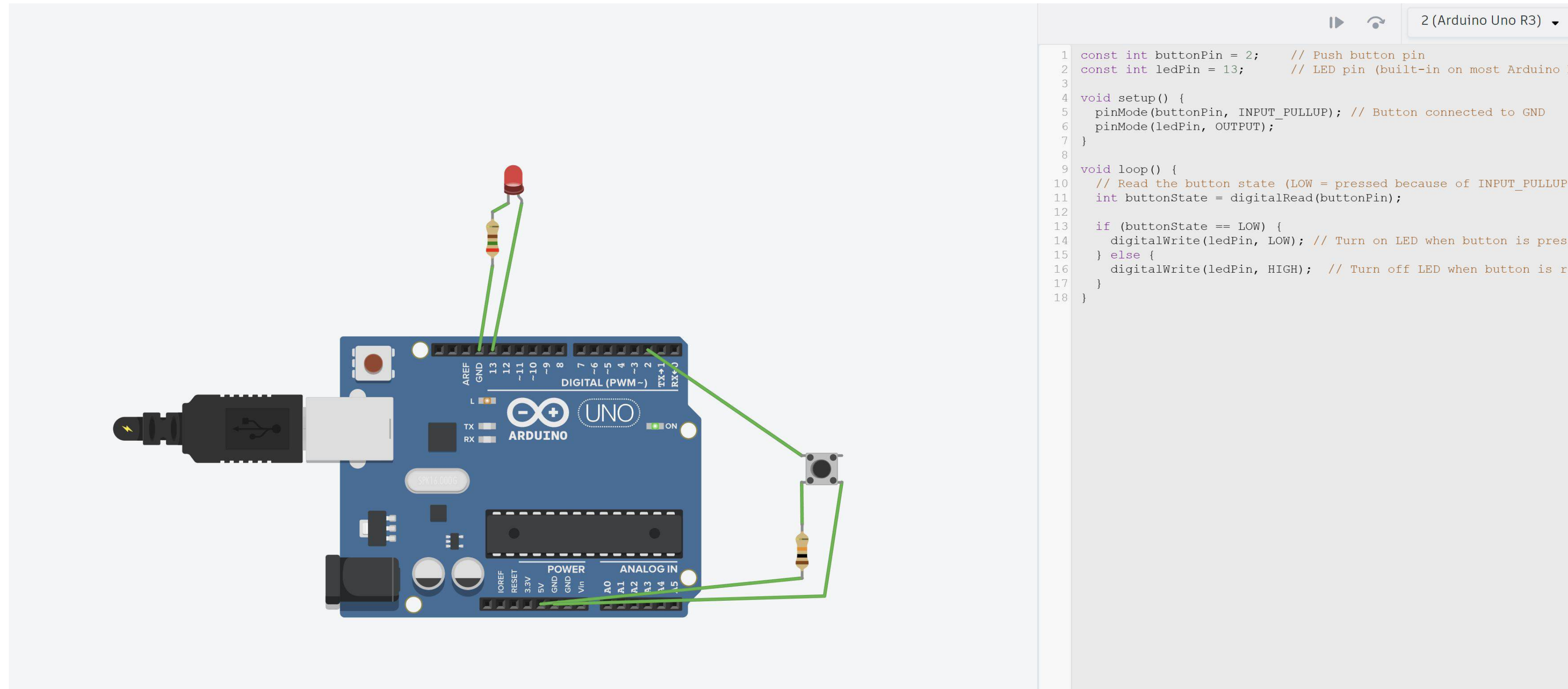
Learnings:

LEDs only work with correct polarity.

Resistors are essential.

Same principle used in real devices (indicators, signals).

Turning off led with push button

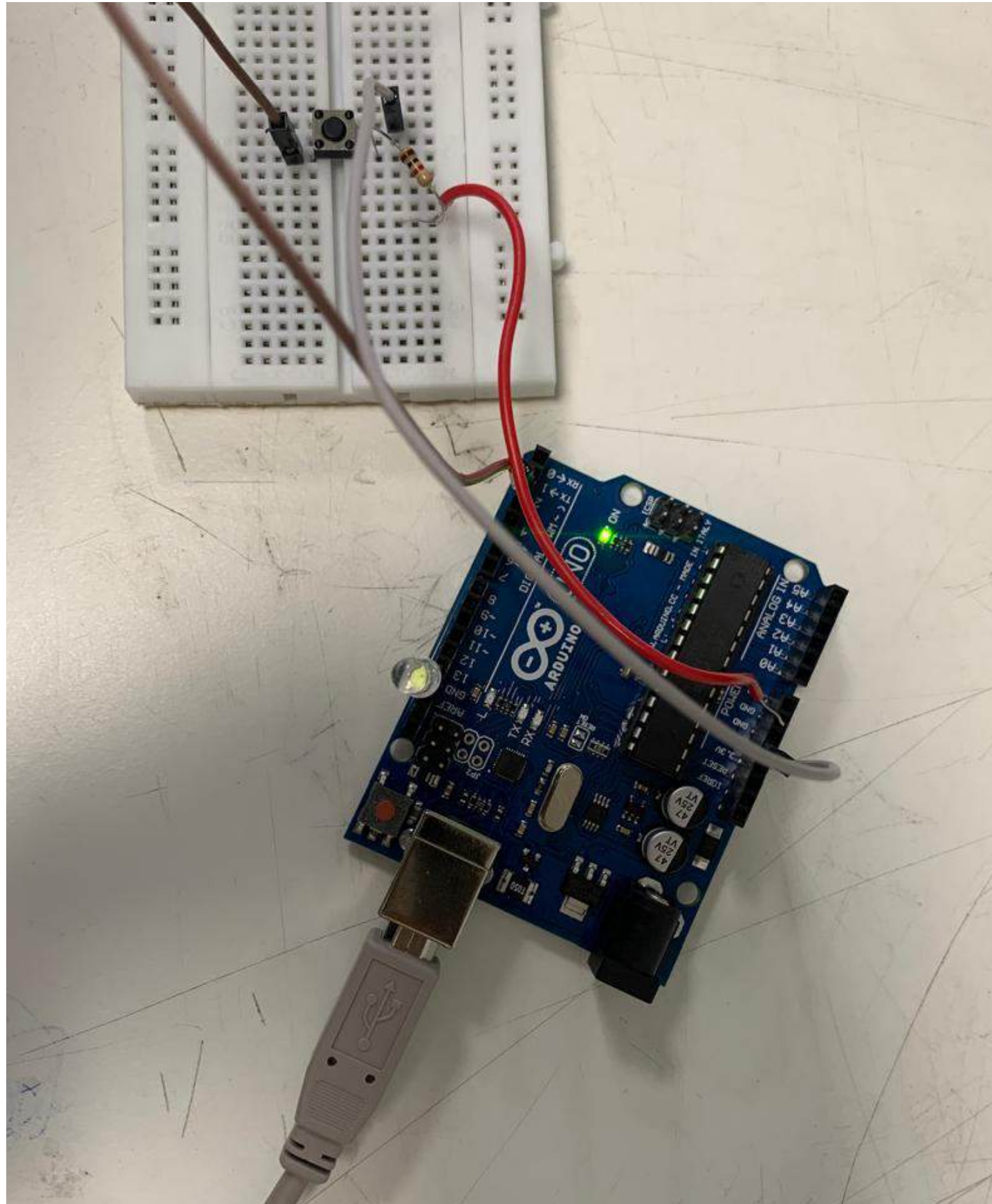


Working:

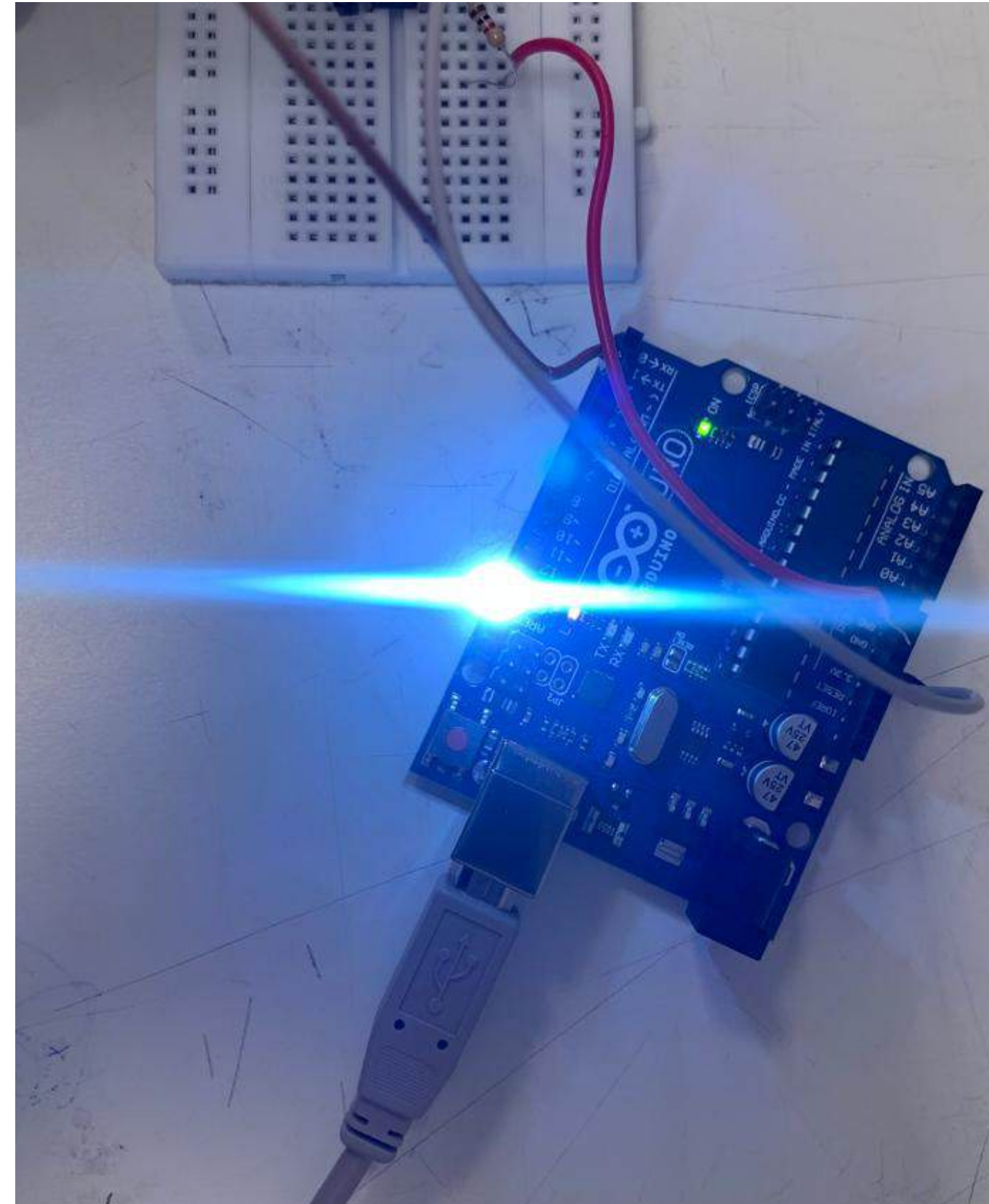
Button not pressed → LED ON.

Button pressed → LED OFF.

Controlling two LEDs alternately (on bread board)

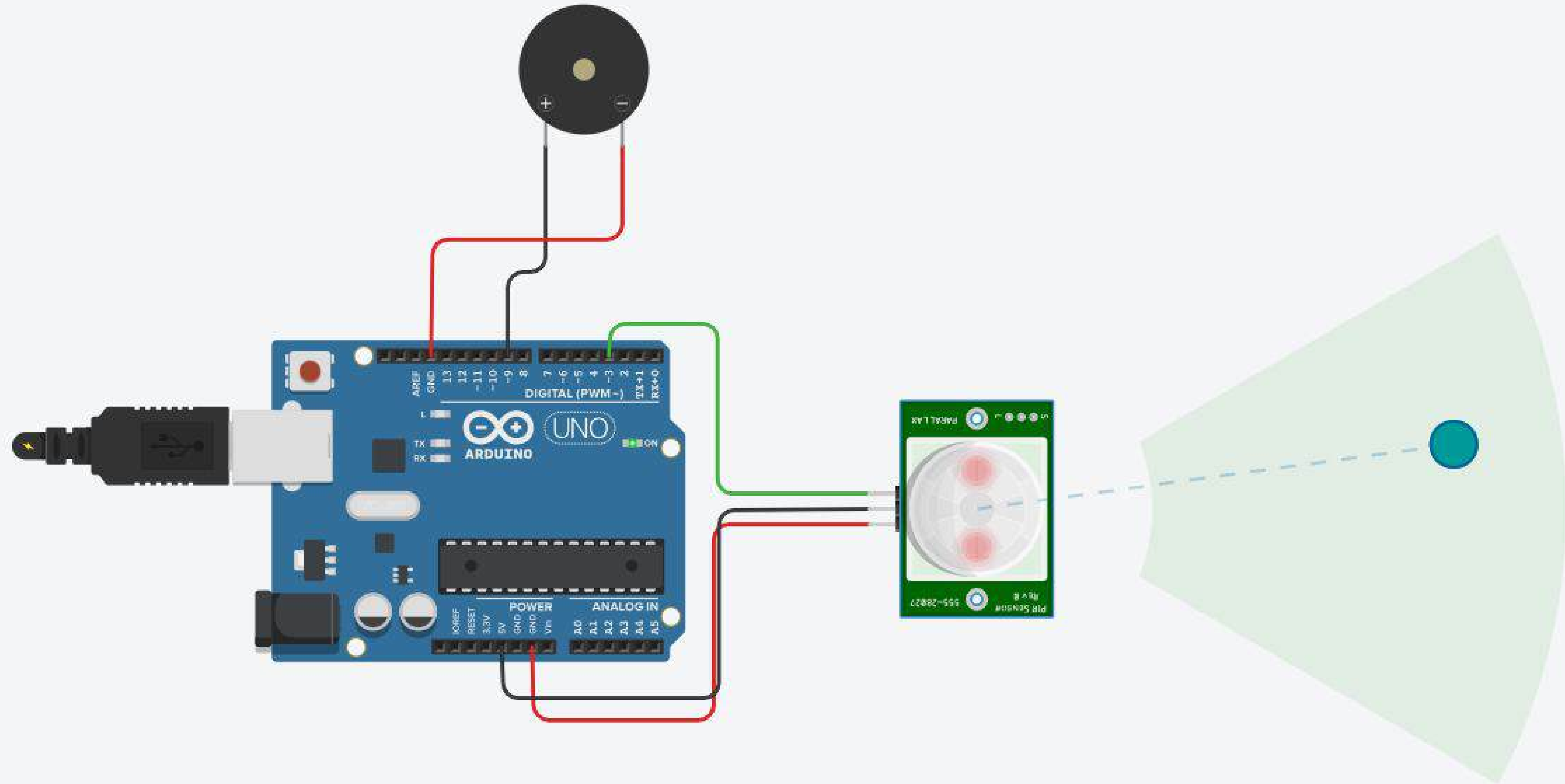


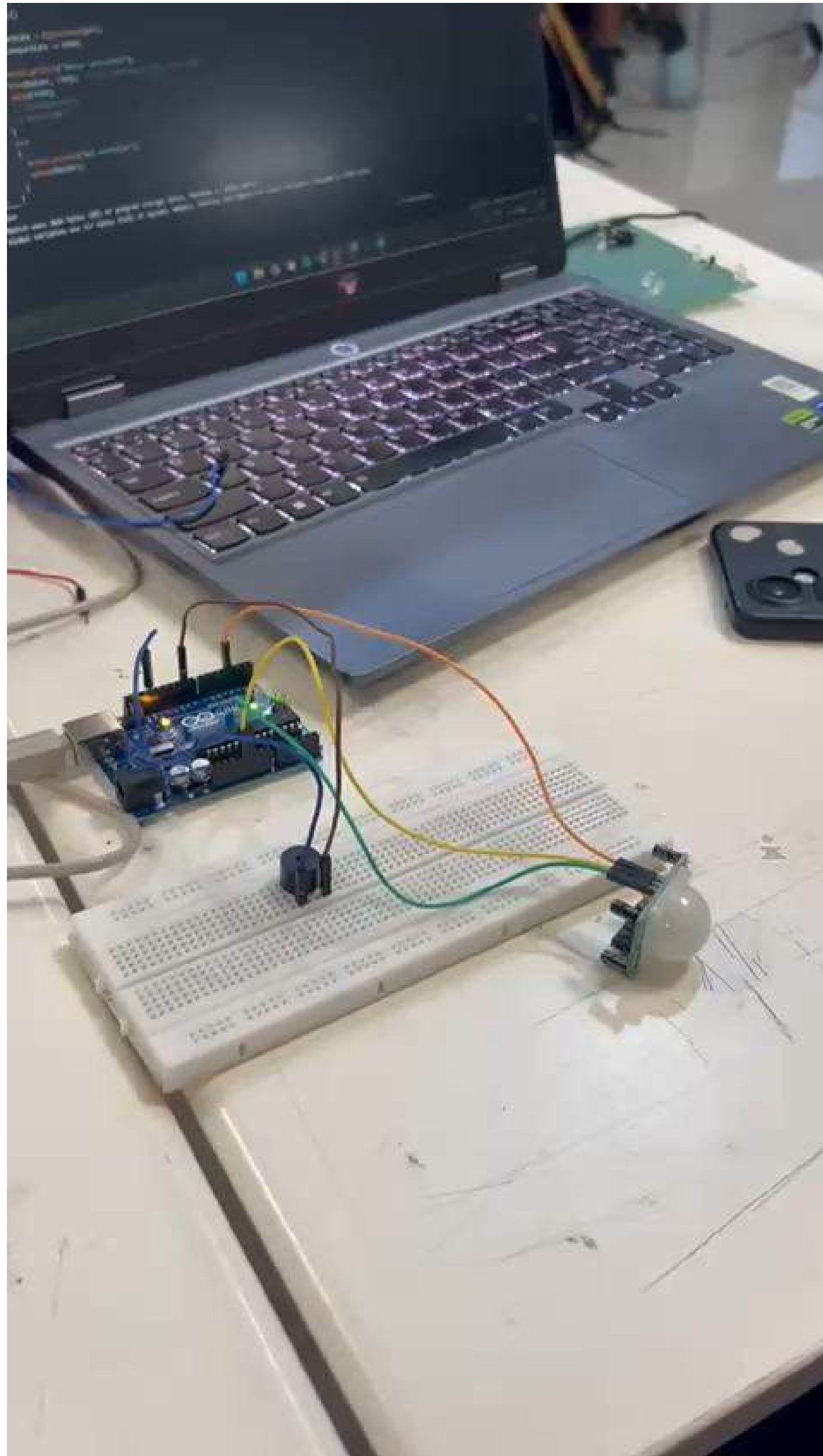
When LED off



When LED on

Controlling PIR sensor and buzzer





Working on bread board

```
sketch_aug1a.ino
1  int buzzer = 9; // buzzer to arduino pin 9
2  int pir = 3;
3  int sensorState = 0;
4
5  void setup()
6  {
7      pinMode(buzzer, OUTPUT); // Set buzzer - pin 9 as an output
8      pinMode(pir, INPUT);
9      Serial.begin(9600);
10 }
11
12 void loop()
13 {
14     sensorState = digitalRead(pir);
15     if (sensorState == HIGH)
16     {
17         Serial.println("Sensor activated!");
18         tone(buzzer, 1000); // Here 1000 means 1KHz sound signal
19         delay(5000);
20         // noTone(buzzer);
21         // delay(1000);
22     }
23     else
24     {
25         Serial.println("Not activated");
26         noTone(buzzer);
27     }
28 }
```

Output Serial Monitor X

Message (Enter to send message to 'Arduino Uno' on 'COM3')

Not activated
Not activated
Not activated
Not activated
Not activated

Code

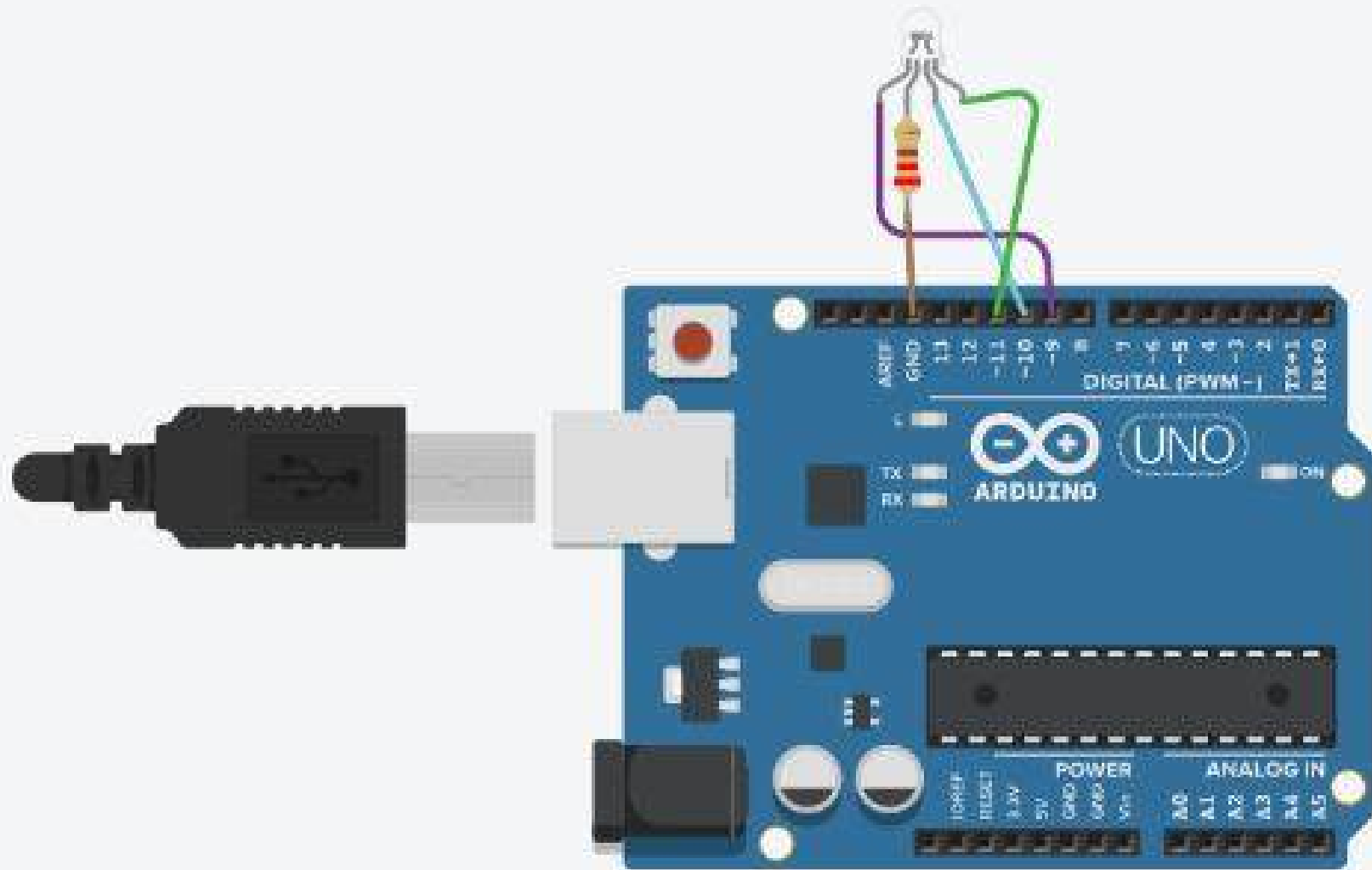
RGB LED (digital code)

TIN KER CAD Frantic Kieran

All changes saved

Code Start Simulation Send To

Text 1 (Arduino Uno R3)



```
1 int red1 = 9;
2 int green1 = 10;
3 int blue1 = 11;
4
5 void setup() {
6   pinMode(red1, OUTPUT);
7   pinMode(green1, OUTPUT);
8   pinMode(blue1, OUTPUT);
9 }
10
11 void RGB(int redValue, int greenValue, int blueValue) {
12   digitalWrite(red1, redValue);
13   digitalWrite(green1, greenValue);
14   digitalWrite(blue1, blueValue);
15 }
16
17 void loop() {
18   RGB(1, 0, 0); // Red
19   delay(1000);
20
21   RGB(0, 1, 0); // Green
22   delay(1000);
23
24   RGB(0, 0, 1); // Blue
25   delay(1000);
26 }
27
```

Serial Monitor

RGB LED (analogue code)

```
Arduino — RGB LED Color Cycle

/* Define the pins for each color */
const int redPin = 9;
const int greenPin = 10;
const int bluePin = 11;

/* Delay time between color changes (ms) */
const int delayTime = 1000; // milliseconds

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  // Turn on red
  setColor(255, 0, 0);
  delay(delayTime);

  // Turn on green
  setColor(0, 255, 0);
  delay(delayTime);

  // Turn on blue
  setColor(0, 0, 255);
  delay(delayTime);

  // Turn on yellow (red + green)
  setColor(255, 255, 0);
  delay(delayTime);

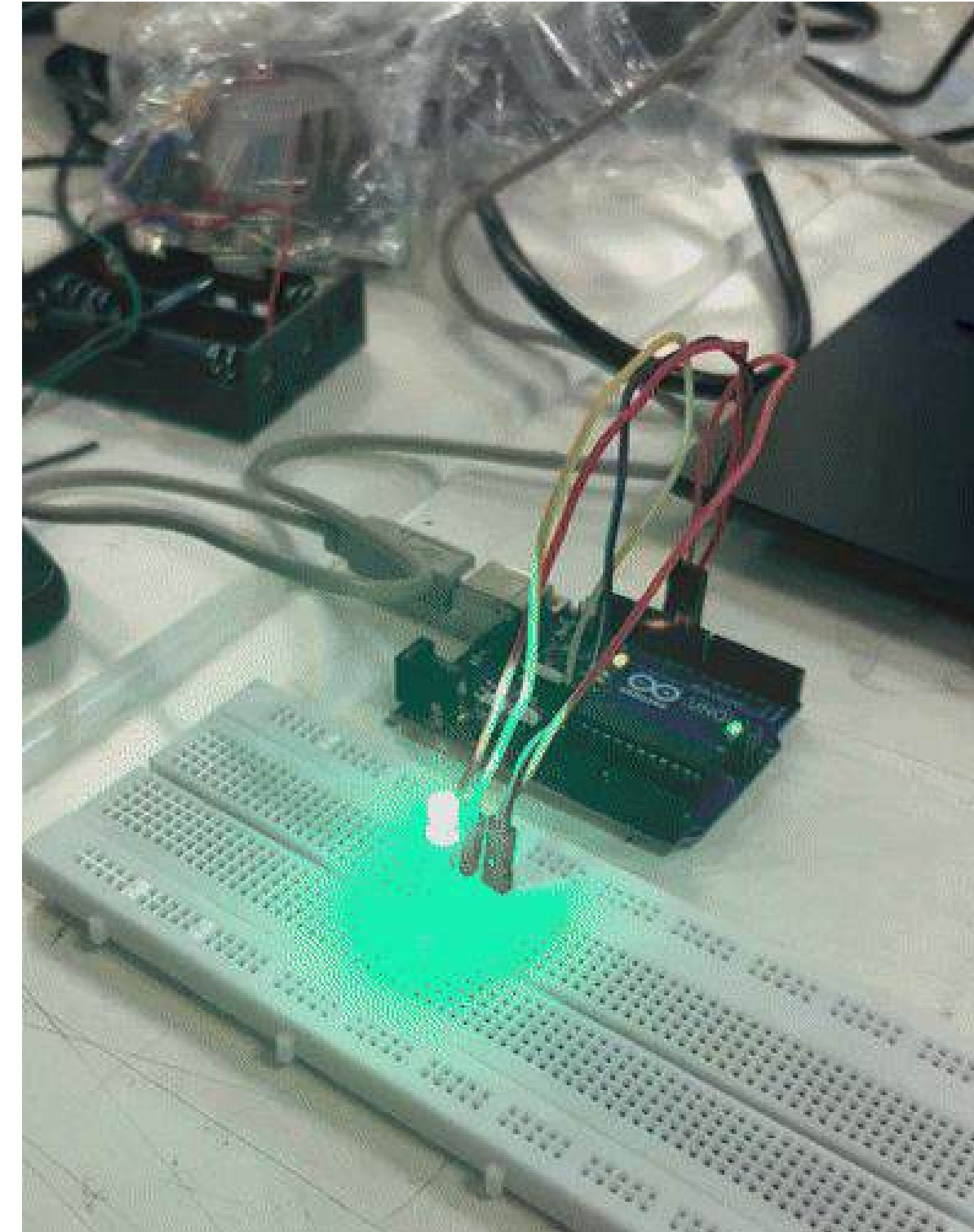
  // Turn on cyan (green + blue)
  setColor(0, 255, 255);
  delay(delayTime);

  // Turn on magenta (red + blue)
  setColor(255, 0, 255);
  delay(delayTime);

  // Turn on white (red + green + blue)
  setColor(255, 255, 255);
  delay(delayTime);

  // Turn off the LED
  setColor(0, 0, 0);
  delay(delayTime);
}

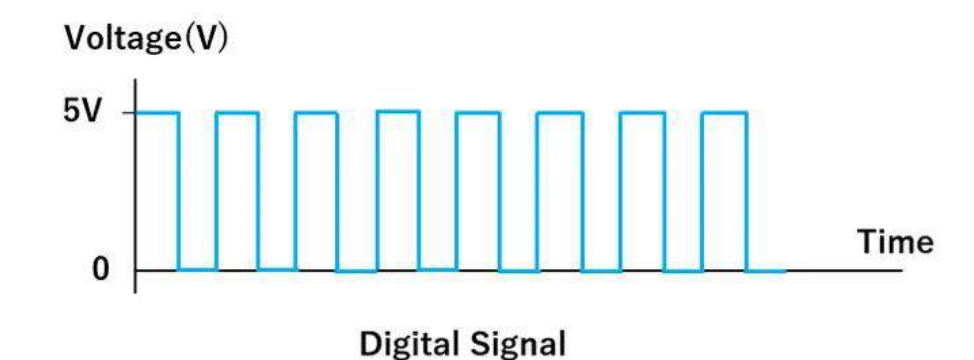
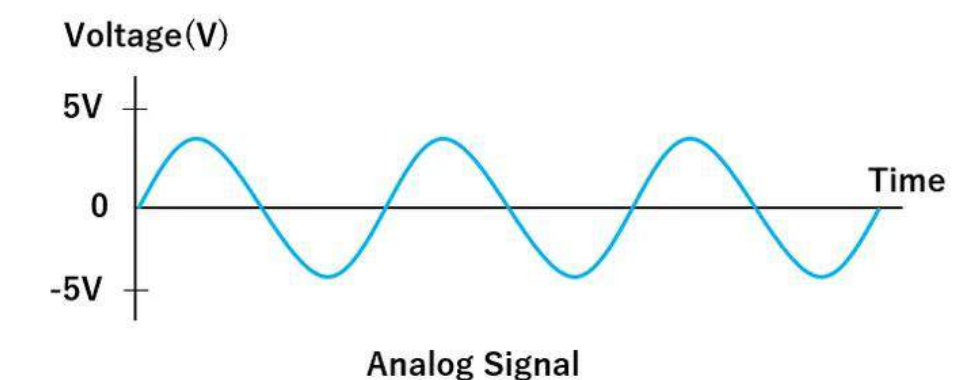
// Function to set the color of the RGB LED
void setColor(int red, int green, int blue) {
  // For common anode, HIGH means off and LOW means on
  analogWrite(redPin, 255 - red);
  analogWrite(greenPin, 255 - green);
  analogWrite(bluePin, 255 - blue);
}
```



- Analogue control (PWM) uses `analogWrite(0–255)` to adjust brightness smoothly, allowing millions of possible color combinations.
- Digital control uses `digitalWrite(HIGH/LOW)`, giving only ON or OFF states for each pin, which limits the number of colors.
- Analogue allows smooth fades and transitions, while digital offers only fixed, abrupt changes.

Our Learnings

- The RGB 4-pin LED has one common pin (either common cathode or common anode) and three separate pins for Red, Green, and Blue.
- By varying the brightness of each color, we can mix them to produce different colors.
- Light from the three internal LEDs combines using additive color mixing.
- Example: Red + Green = Yellow, Green + Blue = Cyan, Red + Blue = Magenta, and all three together = White.



Flex Sensor

TIN
KER
CAD

Powerful Jofo

All changes saved

Simulator time: 00:00:08

Code Stop Simulation Send To

Flex Sensor

Name 1

1 (Arduino Uno R3)

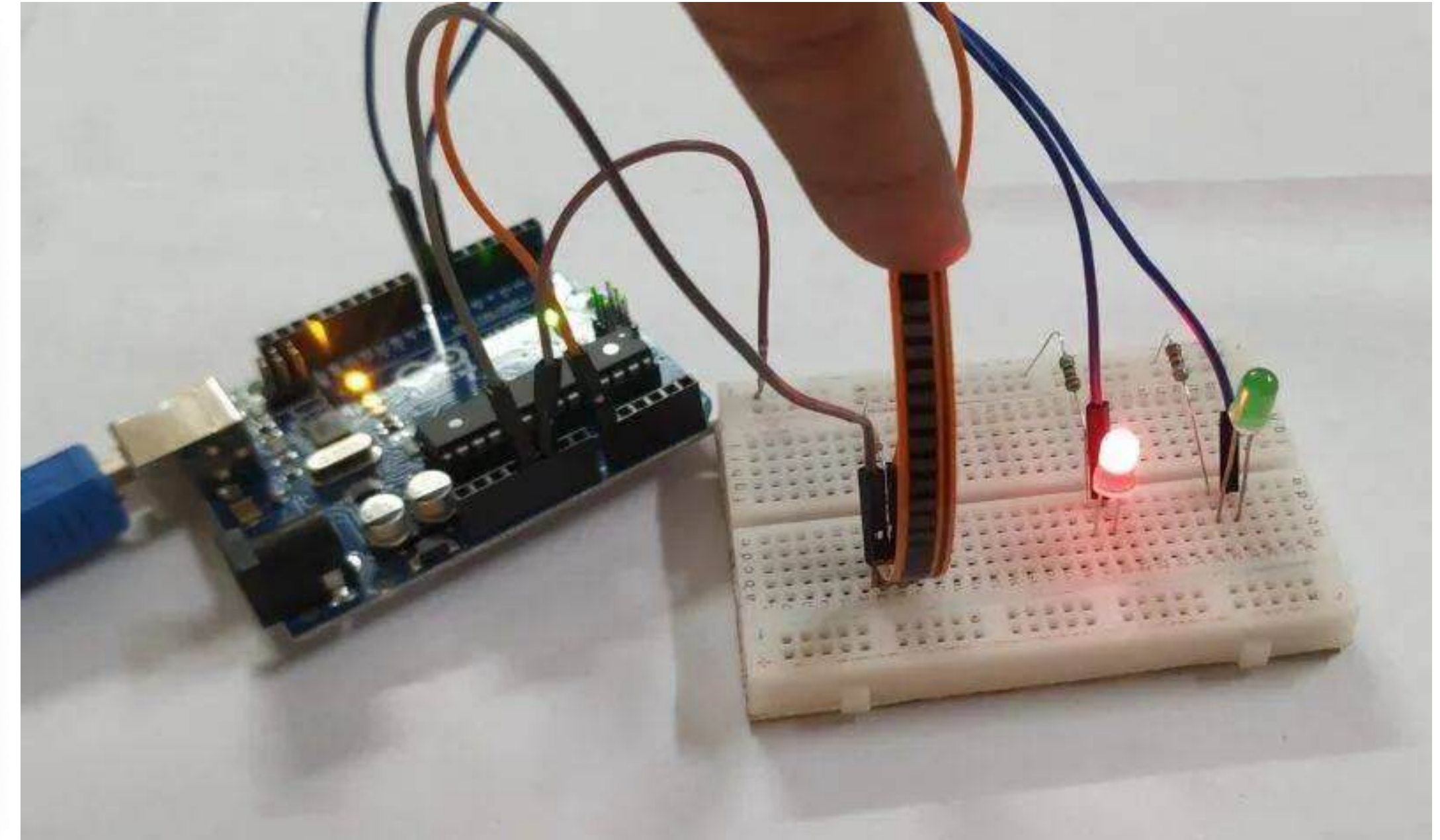
```
1 int flexPin = A2; // Flex sensor connected to analog pin A2
2 int value;
3
4 void setup() {
5   Serial.begin(9600); // Start serial communication
6 }
7
8 void loop() {
9   value = analogRead(flexPin); // Read the analog value from the
10  Serial.println(value);       // Print the value to the Serial M
11 }
```

Serial Monitor

Arduino — Flex Sensor Reading

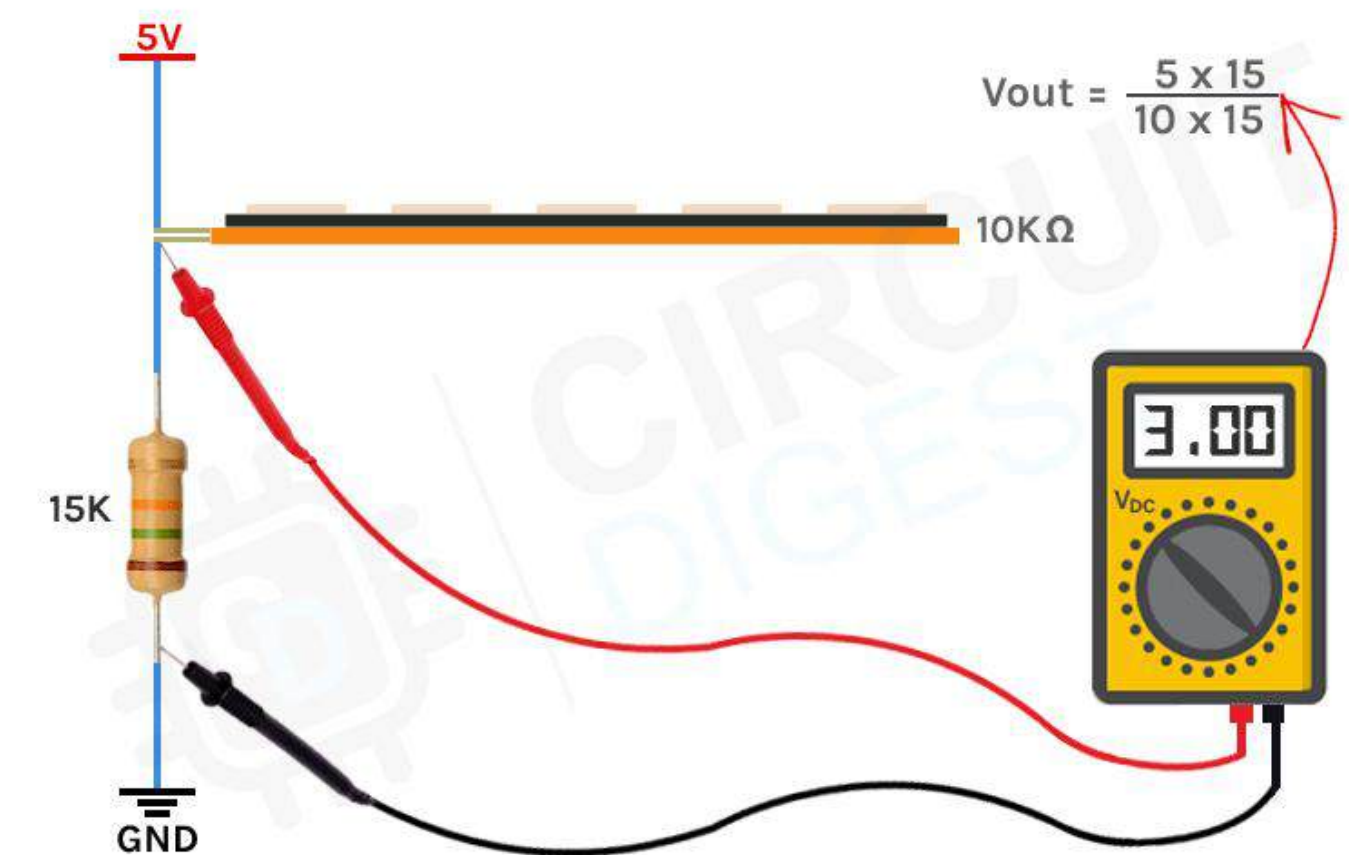
```
int flex= A0;
int val;
int print;
void setup()
{
  Serial.begin(9600);
  pinMode(flex,INPUT);
}

void loop()
{
  val = analogRead(flex);
  print=map(val,0,1023,0,180);
  Serial.println(print);
  delay(100);
}
```



Our Learnings

- A flex sensor changes its resistance when bent — more bend = higher resistance.
- It works as a variable resistor in a voltage divider circuit, producing different voltage outputs based on bend angle.
- We learned to read these voltage changes using an Arduino's analog input pins (analogRead).
- By mapping the sensor values, we can convert bends into meaningful outputs (e.g., controlling a servo, triggering events).
- The sensor is direction-sensitive, meaning it detects bending in one primary direction more accurately.



Ultrasonic sensor

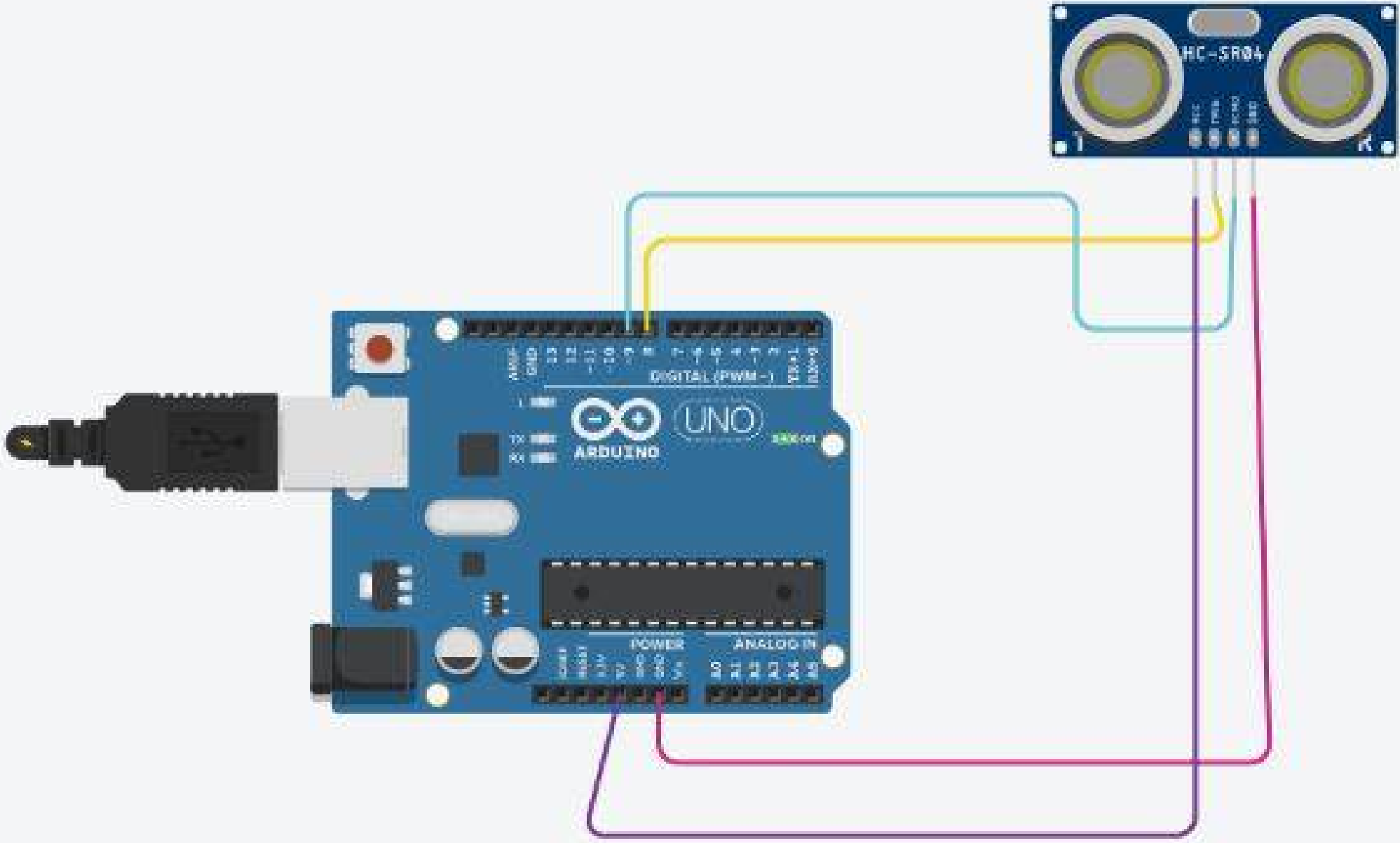
TIN K E R CAD Incredible Snaget

All changes saved

Simulator time: 00:00:18.678

Code Stop Simulation Send To

1 (Arduino Uno R3)



```
1  int trigPin = 8;
2  int echoPin = 9;
3  long duration;
4  int distance;
5  void setup()
6  {
7    pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
8    pinMode(echoPin, INPUT); // Sets the echoPin as an Input
9    Serial.begin(9600); // Starts the serial communication
10 }
11 void loop()
12 {
13     digitalWrite(trigPin, LOW);
14     delay(10);
15     digitalWrite(trigPin, HIGH);
16     delay(10);
17     digitalWrite(trigPin, LOW);
18
19     duration = pulseIn(echoPin, HIGH);
20     // Calculating the distance
21     distance= duration*0.034/2;
22     // Prints the distance on the Serial Monitor
23     Serial.print("Distance: ");
24     Serial.println(distance);
```

Serial Monitor

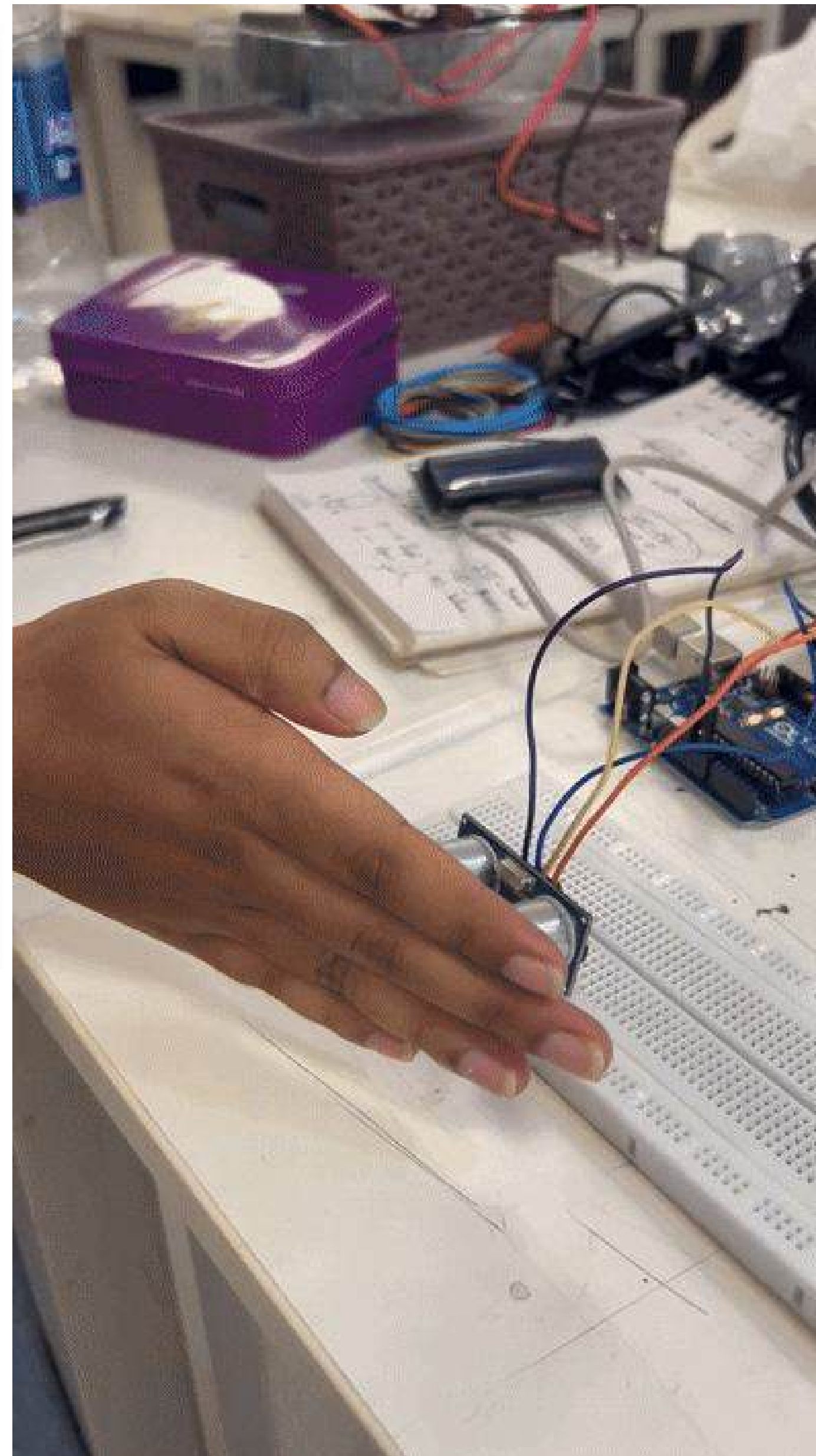
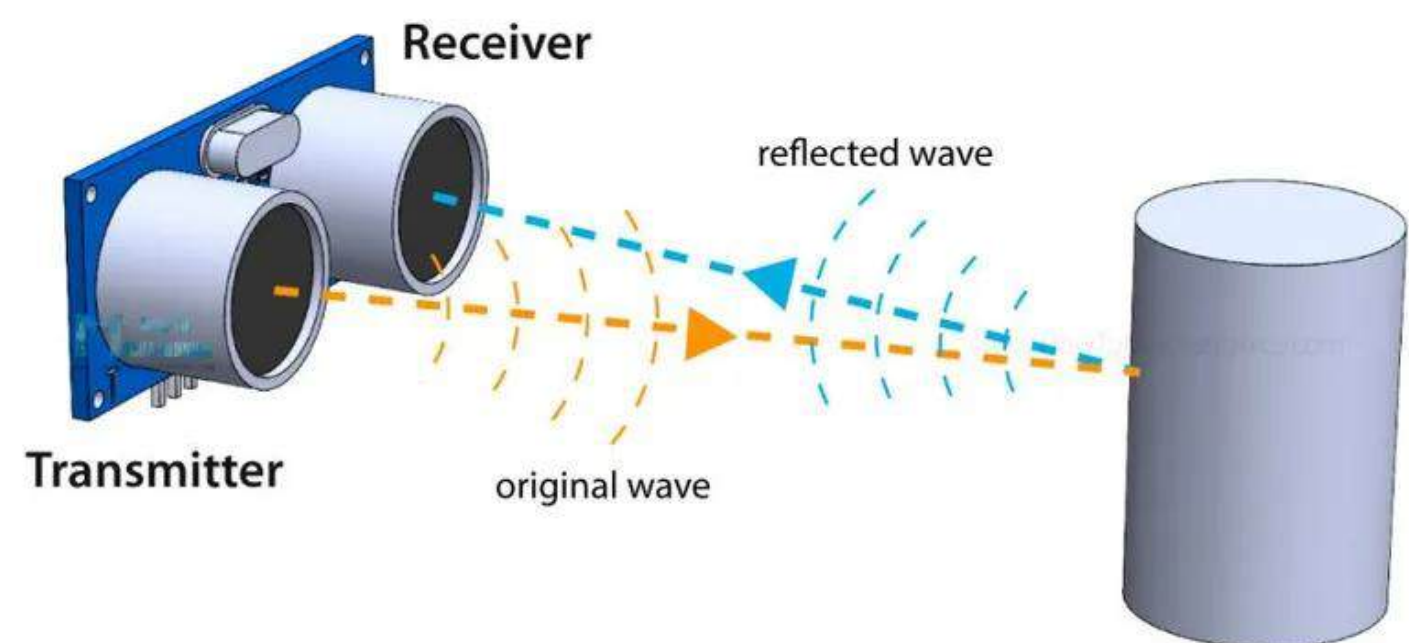
Distance: 332
Distance: 332
Distance: 332
Distance: 332
Distance: 332
Distance: 332

Send Clear

Arduino — Ultrasonic Sensor

```
int trigPin = 8;
int echoPin = 9;
long duration;
int distance;
void setup()
{
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop()
{
  digitalWrite(trigPin, LOW);
  delay(10);
  digitalWrite(trigPin, HIGH);
  delay(10);
  digitalWrite(trigPin, LOW);

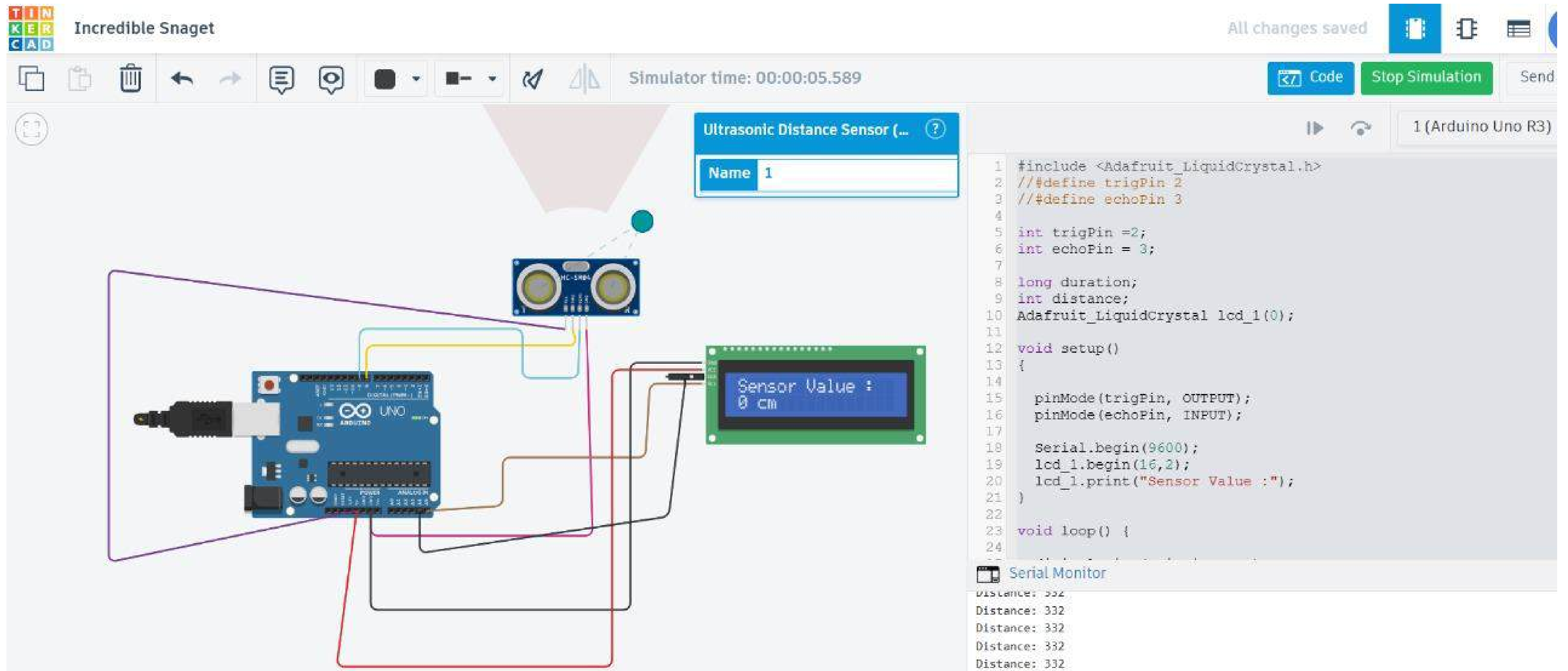
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
```



Our Learnings

- Our Learnings
- A flex sensor changes its resistance when bent — more bend = higher resistance.
- It works as a variable resistor in a voltage divider circuit, producing different voltage outputs based on bend angle.
- We learned to read these voltage changes using an Arduino's analog input pins (analogRead).
- By mapping the sensor values, we can convert bends into meaningful outputs (e.g., controlling a servo, triggering events).
- The sensor is direction-sensitive, meaning it detects bending in one primary direction more accurately.

Ultrasonic sensor with LCD



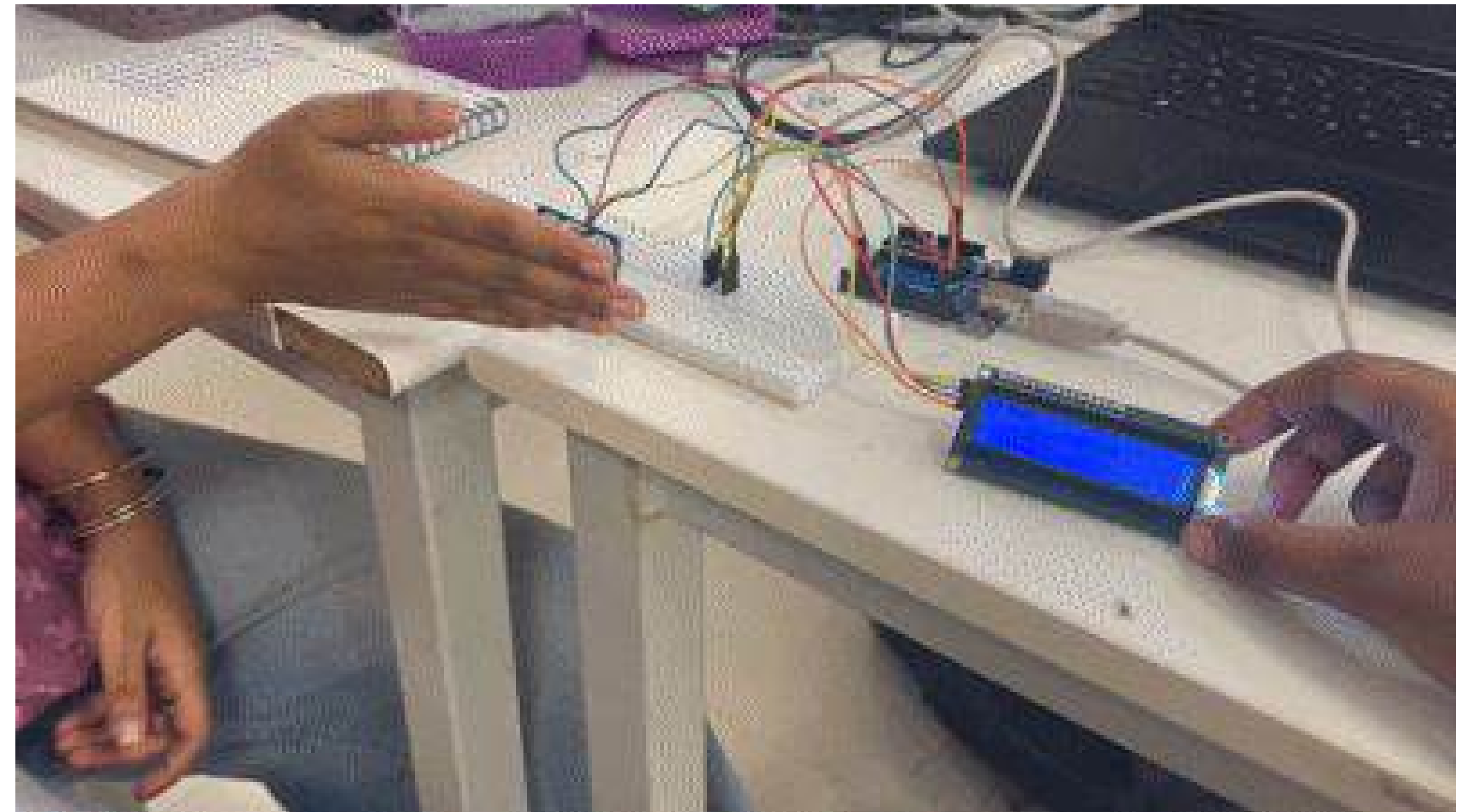

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int trigPin = 2;
int echoPin = 3;
long duration;
int distance;
LiquidCrystal_I2C lcd(0x27,20,4); // set the LCD address to 0x27 for a 16 chars and 2 line display

void setup()
{
  lcd.init(); // initialize the lcd
  lcd.backlight();
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop()
{
  lcd.clear();
  digitalWrite(trigPin, LOW);
  delay(10);
  digitalWrite(trigPin, HIGH);
  delay(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  lcd.setCursor(0,0);
  lcd.print("Distance ");
  lcd.print(distance);
  delay(50);
}

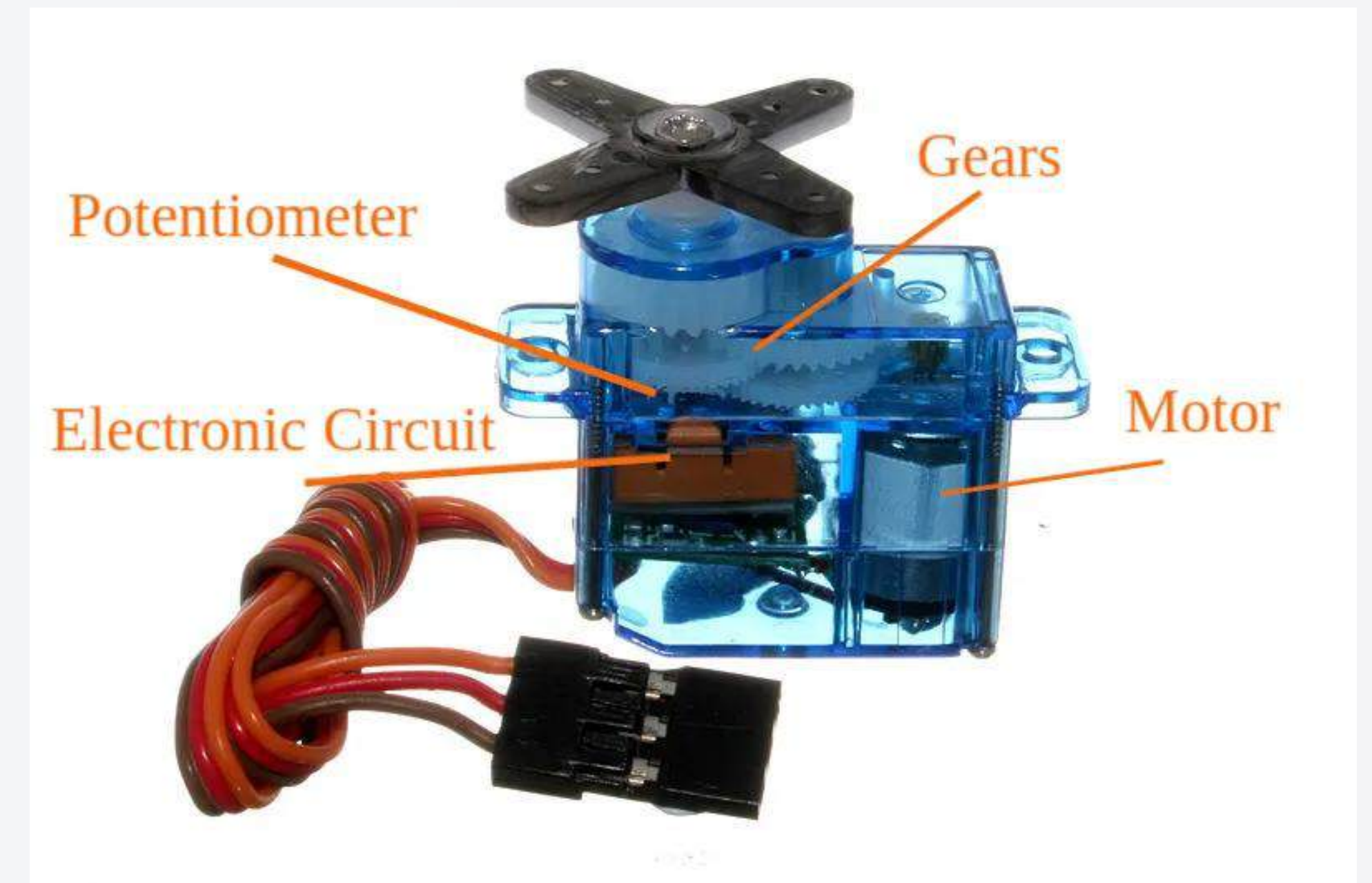
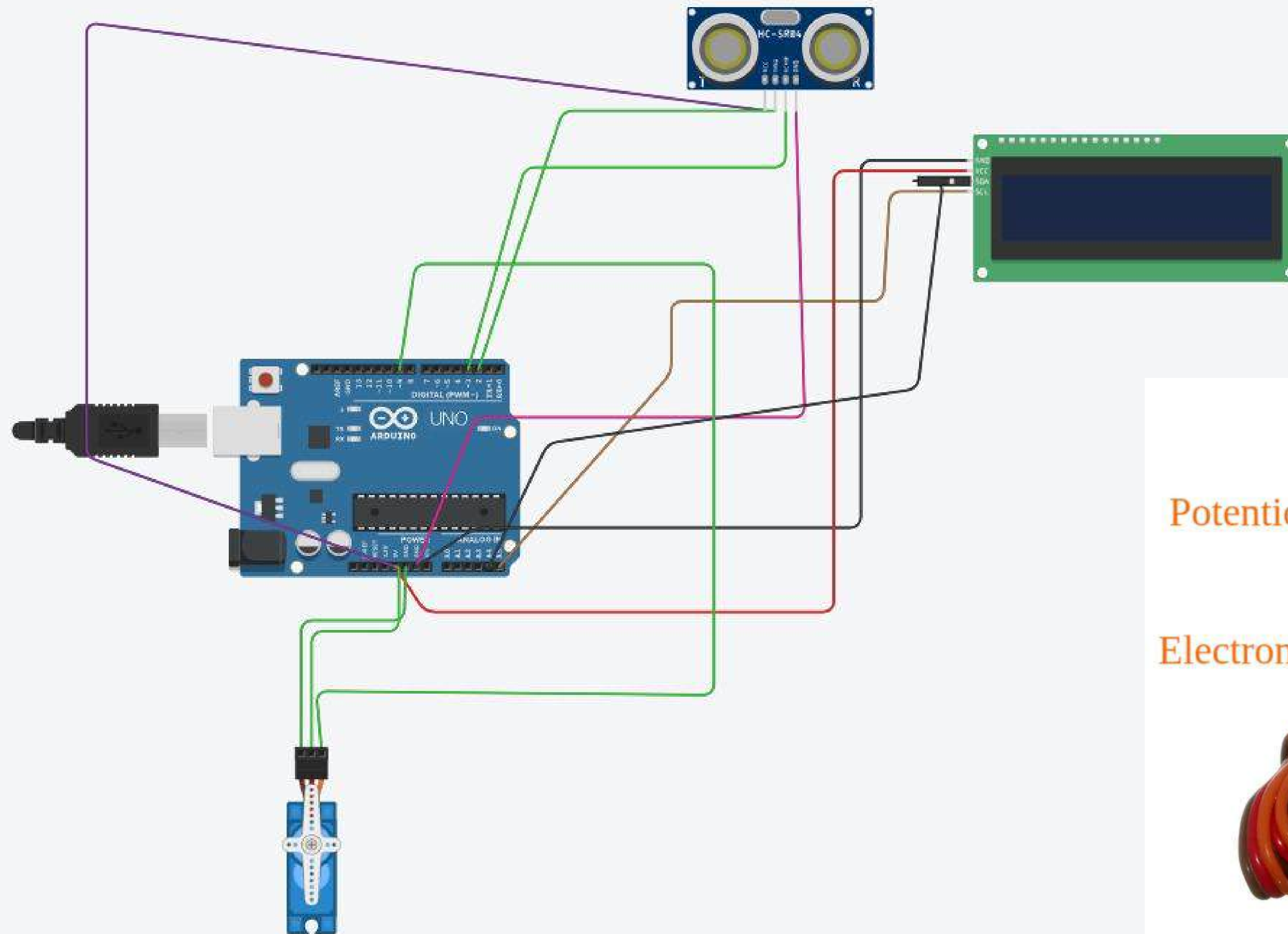
```

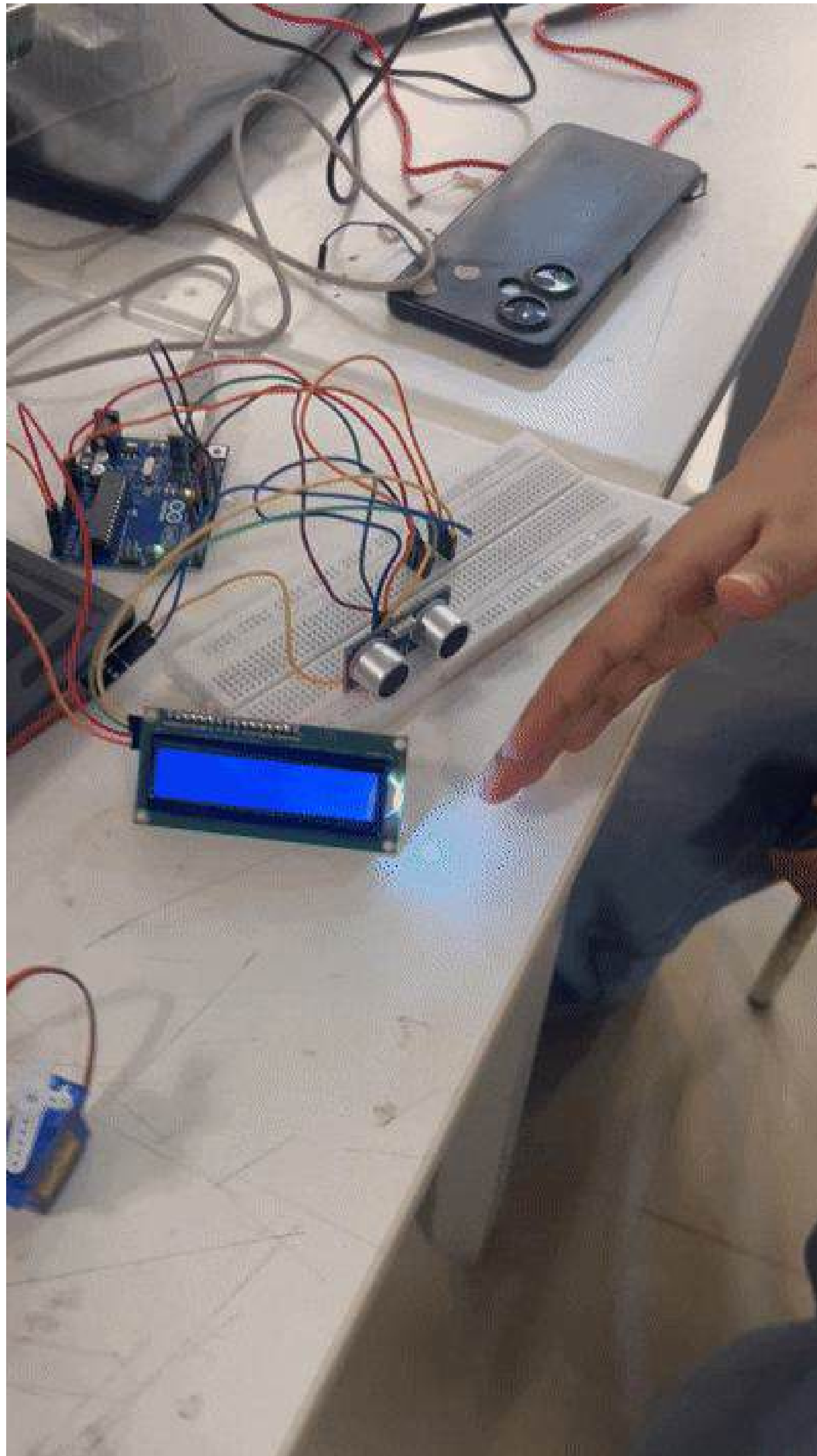


Our Learnings

- We learned to integrate the HC-SR04 ultrasonic sensor with an LCD display to measure and show distances in real time.
- The ultrasonic sensor works by sending sound pulses from the trigger pin and receiving them at the echo pin, with the time measured using pulseIn().
- The Arduino processes this time to calculate the distance and sends the result to the LCD for display.
- We understood how to use the LiquidCrystal library to control the LCD, manage cursor positions, and print custom messages.

Ultrasonic sensor with LCD and microservomotor





```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>

Servo    myservo;
int      trigPin = 2;
int      echoPin = 3;
long     duration;
int      distance;
LiquidCrystal_I2C lcd(0x27,20,4); // set the LCD address to 0x27 for a 20x4 display

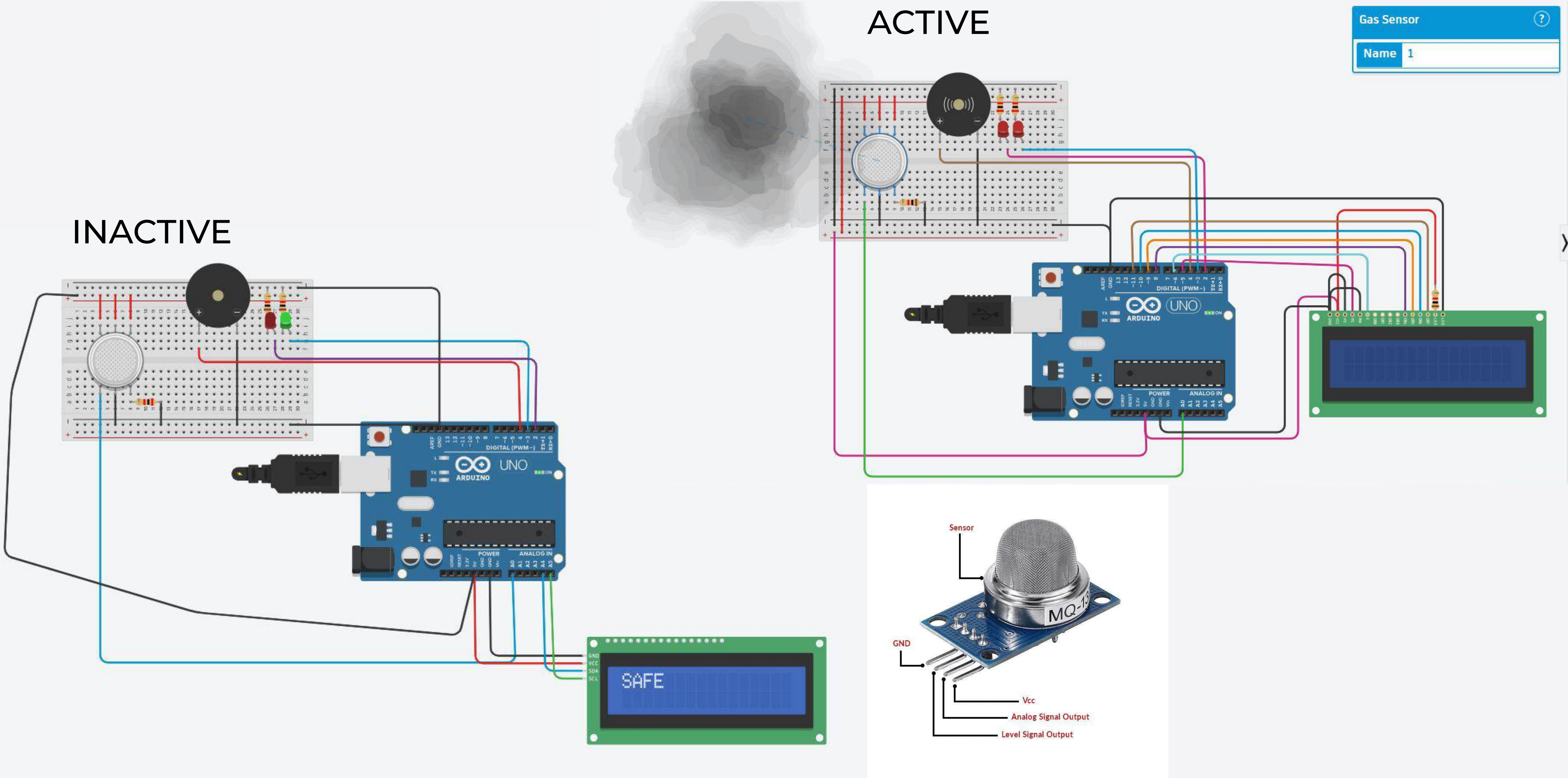
void      setup()    {
    lcd.init(); // initialize the LCD
    lcd.backlight();
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    myservo.attach(9); // Servo on pin 9
}

void      loop()     {
    lcd.clear();
    // ✓ Ultrasonic reading
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration * 0.034 / 2;
    // ✓ Display distance on LCD
    lcd.setCursor(0,0);
    lcd.print("Distance: ");
    lcd.print(distance);
    lcd.print(" cm");
    // ✓ Map distance to servo angle (0-180)
    int angle = map(distance, 0, 100, 180, 0);
    angle = constrain(angle, 0, 180); // Make sure it stays in range
    myservo.write(angle); // Move servo according to distance
    delay(200); // small delay for stability
}
```

Our Learnings

- We learned to integrate the HC-SR04 ultrasonic sensor with an LCD display to measure and show distances in real time.
- The ultrasonic sensor works by sending sound pulses from the trigger pin and receiving them at the echo pin, with the time measured using pulseIn().
- The Arduino processes this time to calculate the distance and sends the result to the LCD for display.
- We understood how to use the LiquidCrystal library to control the LCD, manage cursor positions, and print custom messages.

Arduino with LCD monitor, buzzer and gas sensor





```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int buzzer = 9;    // Buzzer connected to pin 9
int pir = 7;       // PIR sensor connected to pin 7
int sensorState = 0;

LiquidCrystal_I2C lcd(0x27, 16, 2); // LCD (16x2) at I2C
address 0x27

void setup() {
  pinMode(buzzer, OUTPUT);
  pinMode(pir, INPUT);
  lcd.init();
  lcd.backlight();
  Serial.begin(9600);
}

void loop() {
  sensorState = digitalRead(pir);

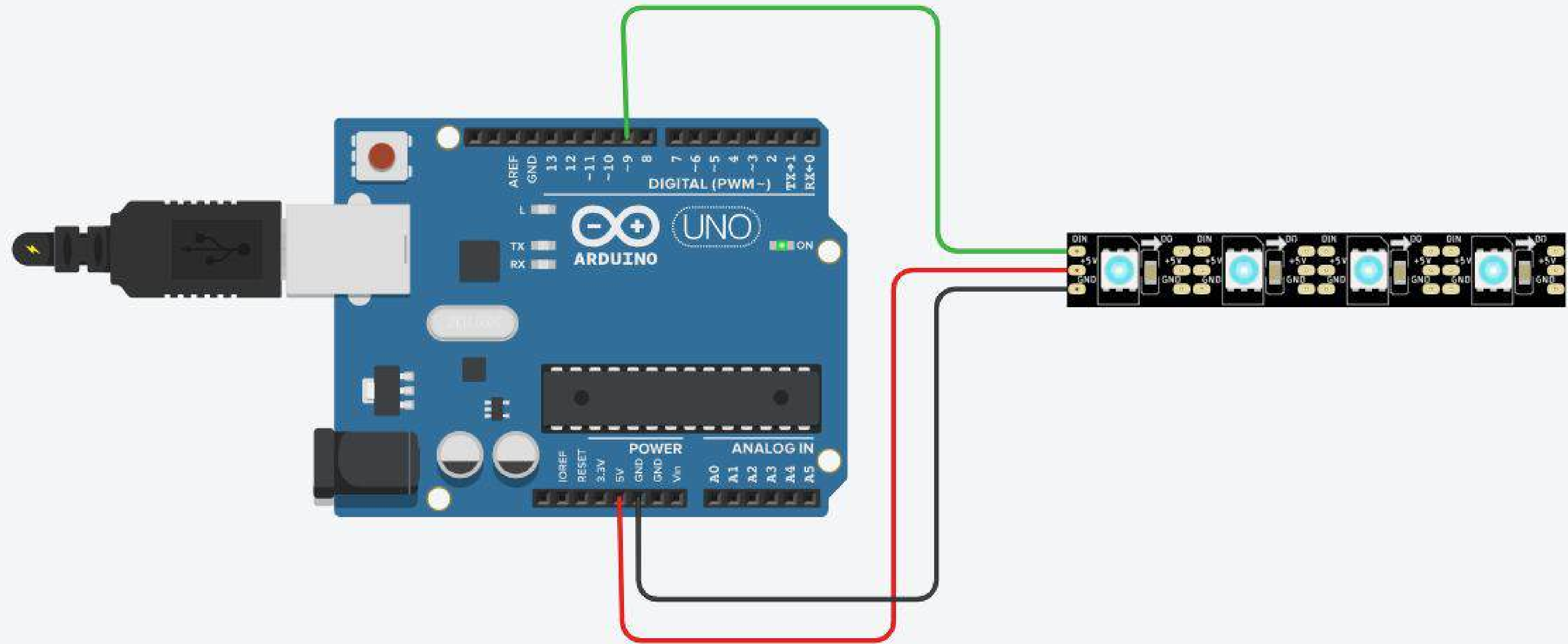
  if (sensorState == HIGH) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Motion Detected");
    Serial.println("Motion Detected!");
    tone(buzzer, 2000);    // Play tone
    delay(5000);          // Keep buzzer on for 5 sec
  } else {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("No Motion");
    Serial.println("No Motion");
    noTone(buzzer);       // Stop buzzer
  }

  delay(200);
}
```

Our Learnings:

- Learned how to interface a gas sensor (MQ series) with Arduino UNO.
- Understood how the sensor detects gas concentration and outputs analog values.
- Used LEDs and buzzer as indicators for gas detection levels.
- Learned to display sensor readings on an LCD screen using the LiquidCrystal library.
- Gained experience in wiring multiple components (sensor, buzzer, LEDs, LCD) on a breadboard.
- Observed the importance of threshold values for triggering alerts.
- Practiced real-time monitoring of environmental parameters with Arduino.

Arduino with Neopixel





```
#include <Adafruit_NeoPixel.h>

// === Pin Definitions ===
const int trigPin = 8;
const int echoPin = 7;
const int neopixelPin = 9;
const int numPixels = 12;

long duration;
int distance;

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(numPixels, neopixelPin, NEO_GRB + NEO_KHZ800);

int fadeSpeed = 5; // Controls speed of fade transition

// Predefined pure RGB colors
const int colors[3][3] = {
  {255, 0, 0}, // Red
  {0, 0, 255}, // Blue
  {255, 255, 0} // Yellow
};

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);

  pixels.begin();
  pixels.show();
  randomSeed(analogRead(0));
}

void loop() {
  // === Measure distance using Ultrasonic ===
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  if (distance <= 50) {
    // === Fade to white ===
    for (int i = 0; i <= 255; i += fadeSpeed) {
      for (int j = 0; j < numPixels; j++) {
        pixels.setPixelColor(j, pixels.Color(i, i, i)); // White
      }
      pixels.show();
      delay(10);
    }
  } else {
    // === Choose a random pure color (R, B, or Y) ===
    int colorIndex = random(0, 3);
    int r = colors[colorIndex][0];
    int g = colors[colorIndex][1];
    int b = colors[colorIndex][2];

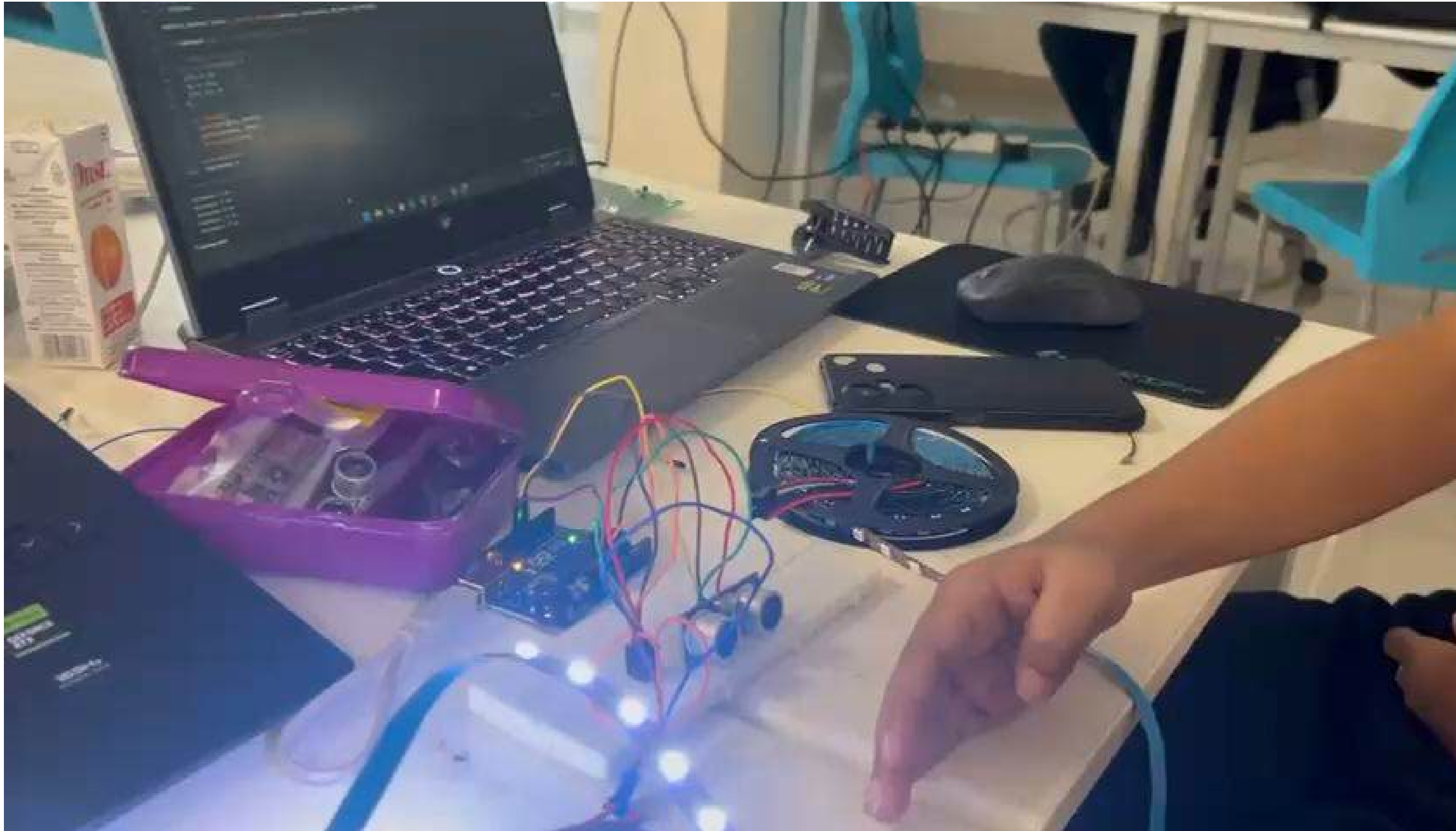
    // === Smooth fade to selected color ===
    for (int i = 0; i <= 255; i += fadeSpeed) {
      int fr = (r * i) / 255;
      int fg = (g * i) / 255;
      int fb = (b * i) / 255;
      for (int j = 0; j < numPixels; j++) {
        pixels.setPixelColor(j, pixels.Color(fr, fg, fb));
      }
      pixels.show();
      delay(10);
    }
  }

  delay(200); // Pause before next distance check
}
```

Our Learnings

- Learned how to interface Neopixel LED strips with Arduino UNO.
- Understood the wiring: 5V (red), GND (black), and Data (green) connection to pin 6.
- Gained familiarity with the Adafruit Neopixel library for controlling LEDs.
- Explored how to set colors, brightness, and animations through code.
- Observed how code logic directly affects LED patterns (loops, delays, RGB values).
- Recognized the importance of power supply stability when working with LED strips.
- Practiced uploading and debugging Arduino sketches for hardware control.
- Improved understanding of digital output control for sequential lighting effects.

Ultrasonic sensor with neopixel



Our Learnings:

- Learned how to interface an ultrasonic sensor with Arduino for distance measurement.
- Understood the working of NeoPixel LEDs and their RGB color control.
- Implemented distance-based lighting effects (fade to white when close, random colors when far).
- Practiced wiring multiple components (sensor + LED strip) on a breadboard.
- Improved skills in Arduino coding with libraries (Adafruit NeoPixel).
- Gained hands-on experience in real-time sensor-based interactions.

```
#include <Adafruit_NeoPixel.h>

// === Pin Definitions ===
const int trigPin = 8;
const int echoPin = 7;
const int neopixelPin = 9;
const int numPixels = 12;

long duration;
int distance;

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(numPixels, neopixelPin, NEO_GRB + NEO_KHZ800);

int fadeSpeed = 5; // Controls speed of fade transition

// Predefined pure RGB colors
const int colors[3][3] = {
  {255, 0, 0}, // Red
  {0, 0, 255}, // Blue
  {255, 255, 0} // Yellow
};

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);

  pixels.begin();
  pixels.show();
  randomSeed(analogRead(0));
}

void loop() {
  // === Measure distance using Ultrasonic ===
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;

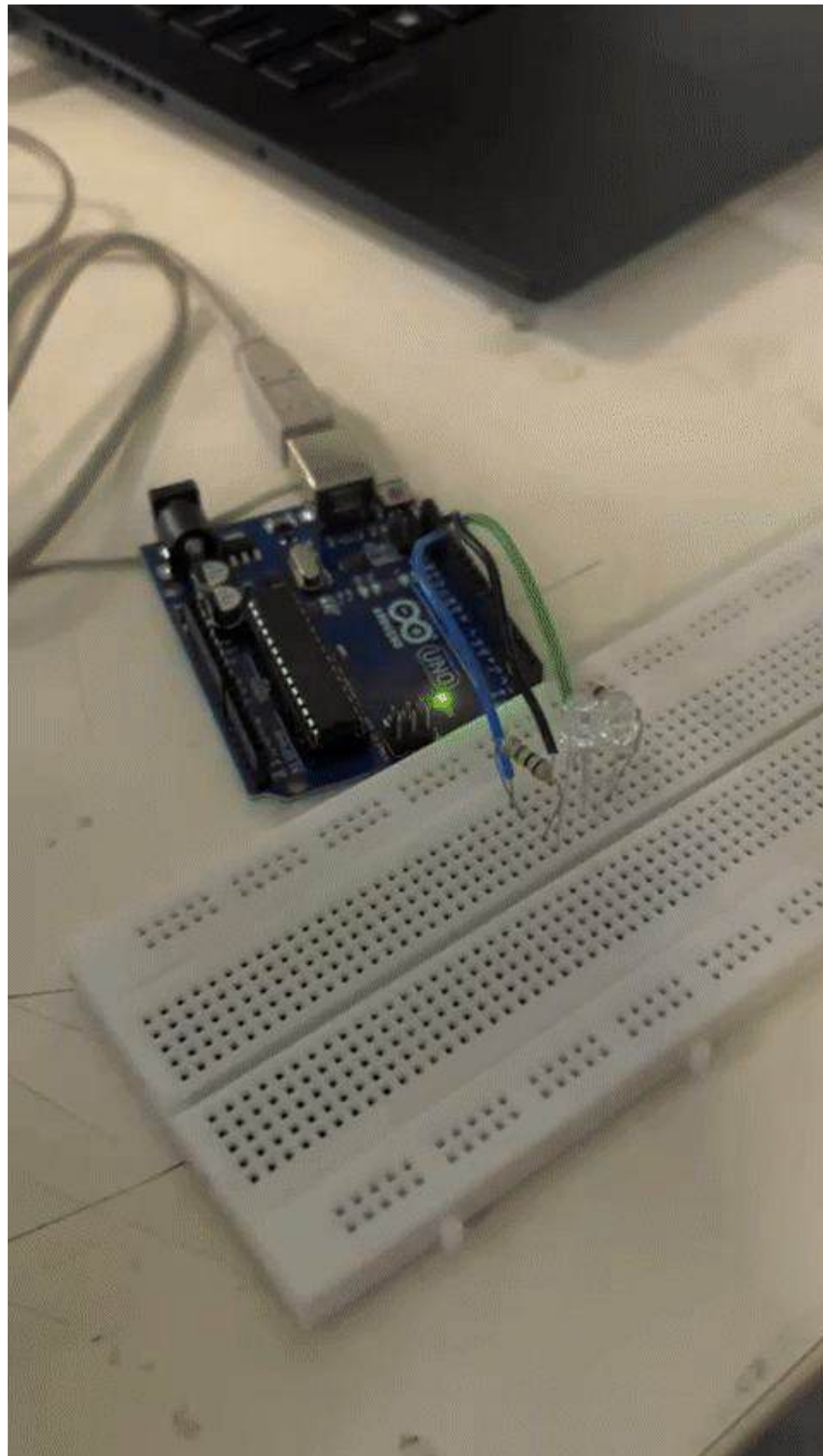
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  if (distance <= 50) {
    // === Fade to white ===
    for (int i = 0; i <= 255; i += fadeSpeed) {
      for (int j = 0; j < numPixels; j++) {
        pixels.setPixelColor(j, pixels.Color(i, i, i)); // White
      }
      pixels.show();
      delay(10);
    }
  } else {
    // === Choose a random pure color (R, B, or Y) ===
    int colorIndex = random(0, 3);
    int r = colors[colorIndex][0];
    int g = colors[colorIndex][1];
    int b = colors[colorIndex][2];

    // === Smooth fade to selected color ===
    for (int i = 0; i <= 255; i += fadeSpeed) {
      int fr = (r * i) / 255;
      int fg = (g * i) / 255;
      int fb = (b * i) / 255;
      for (int j = 0; j < numPixels; j++) {
        pixels.setPixelColor(j, pixels.Color(fr, fg, fb));
      }
      pixels.show();
      delay(10);
    }
  }

  delay(200); // Pause before next distance check
}
```


2 LED alternate blinking



```
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
}

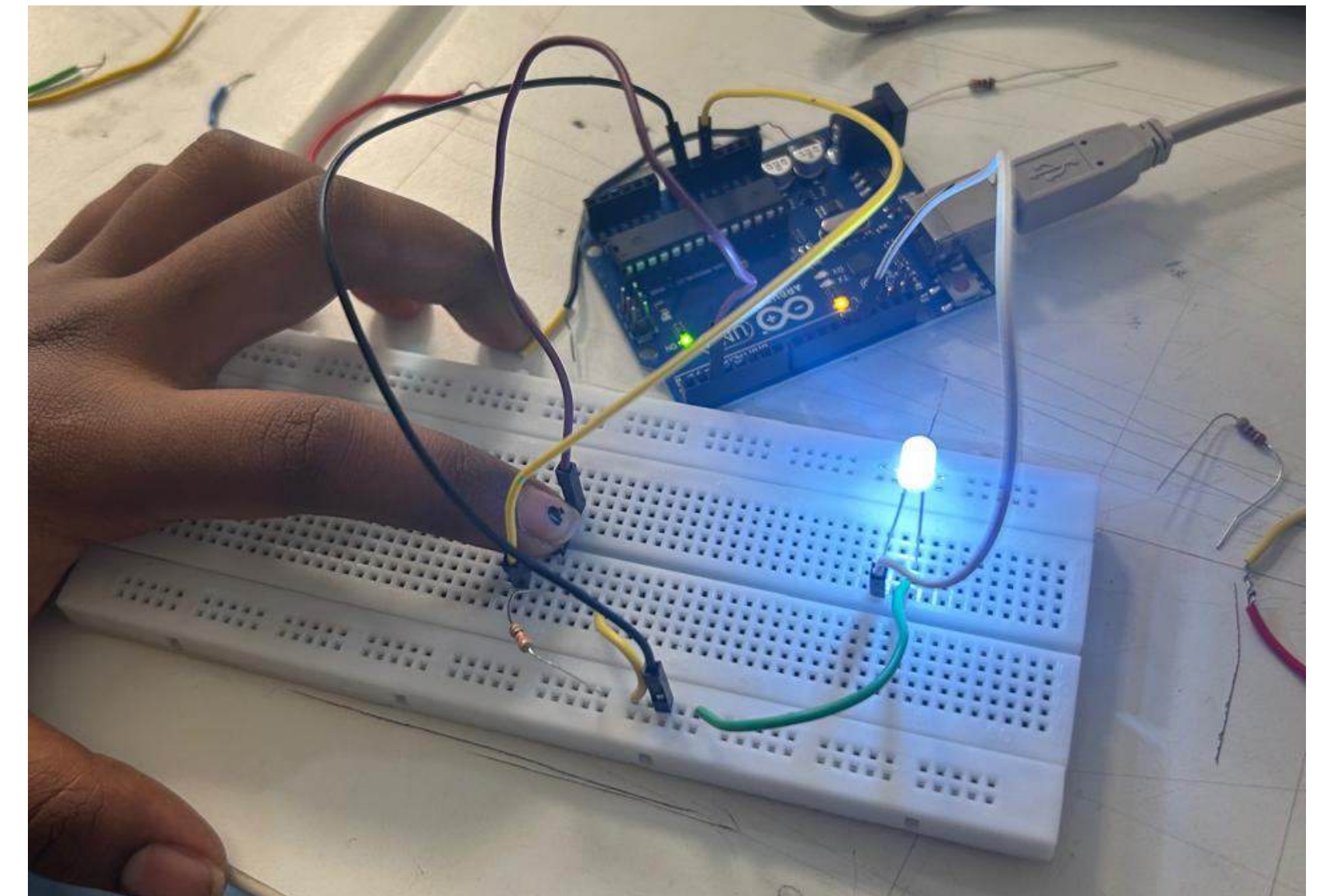
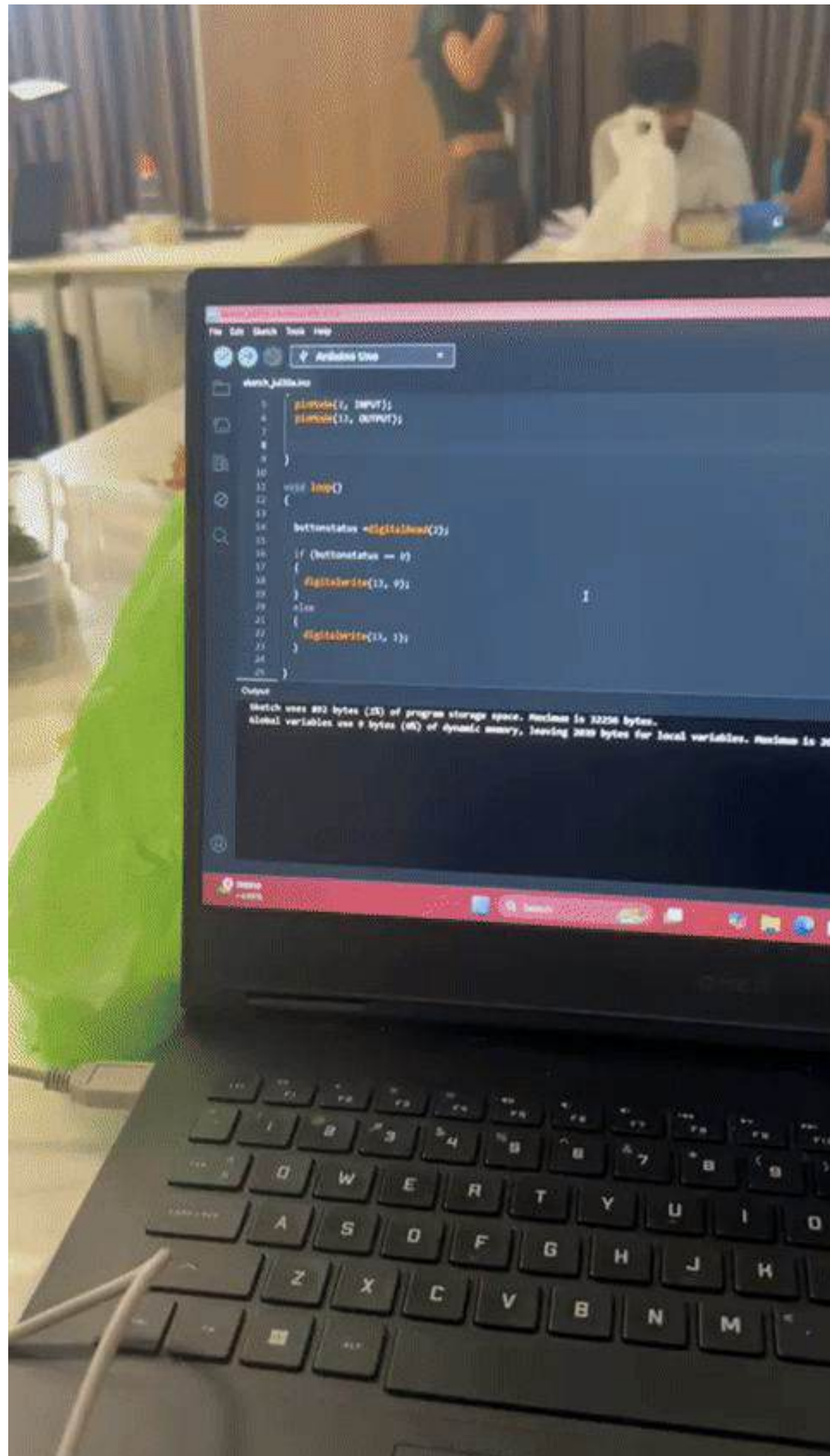
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);

  digitalWrite(12, HIGH);
  delay(1000);
  digitalWrite(12, LOW);
  delay(1000);
}
```

Our Learnings:

- Learned how to connect multiple LEDs to Arduino and control them via digital pins.
- Understood the use of pinMode() to set pins as outputs.
- Practiced using digitalWrite() with HIGH and LOW to turn LEDs on and off.
- Explored how delay() creates time gaps to control blinking patterns.
- Observed how to make LEDs blink alternately by sequencing the code.
- Strengthened understanding of basic Arduino programming structure (setup & loop).

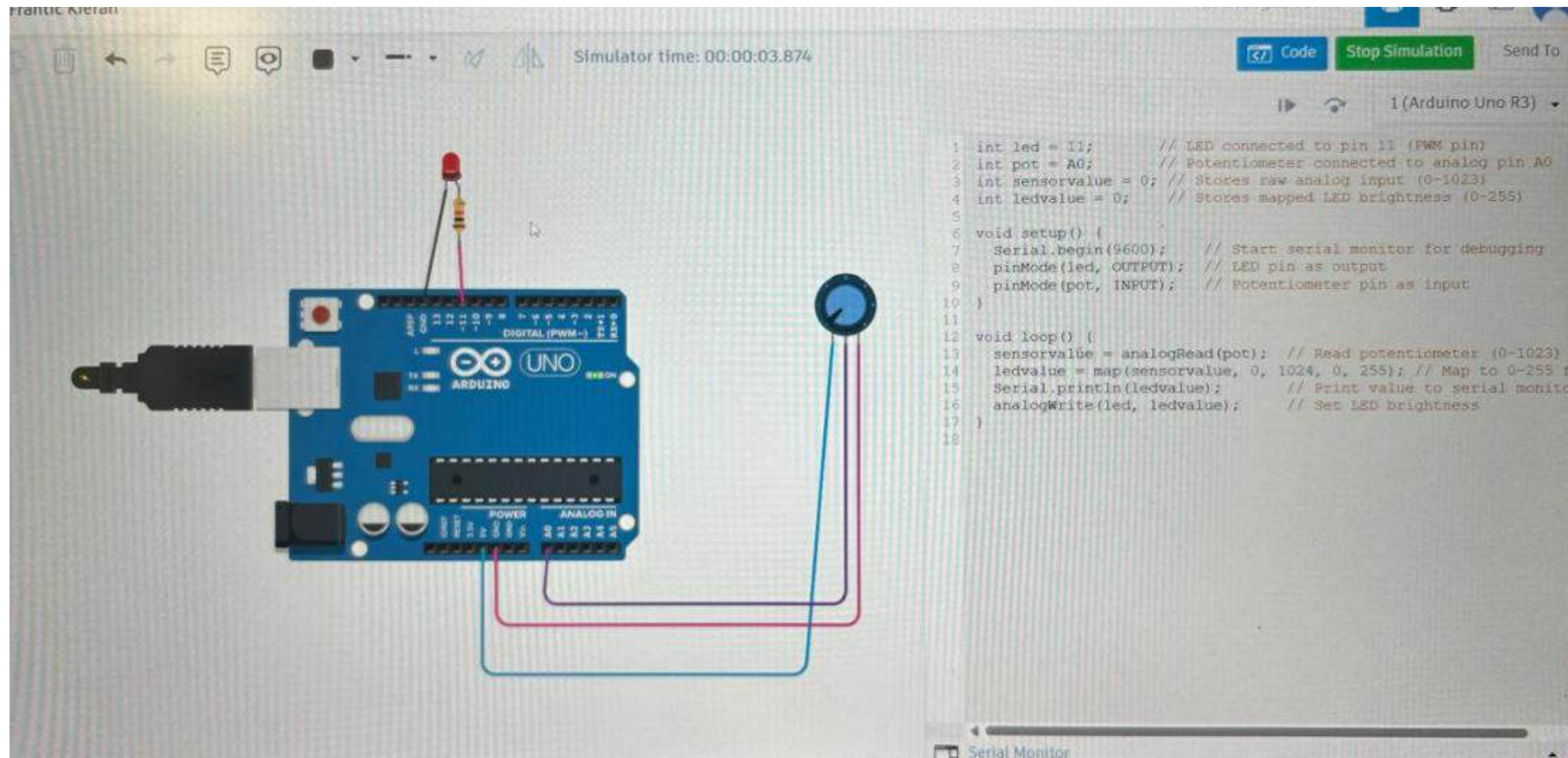
Push button with LED



Our Learnings:

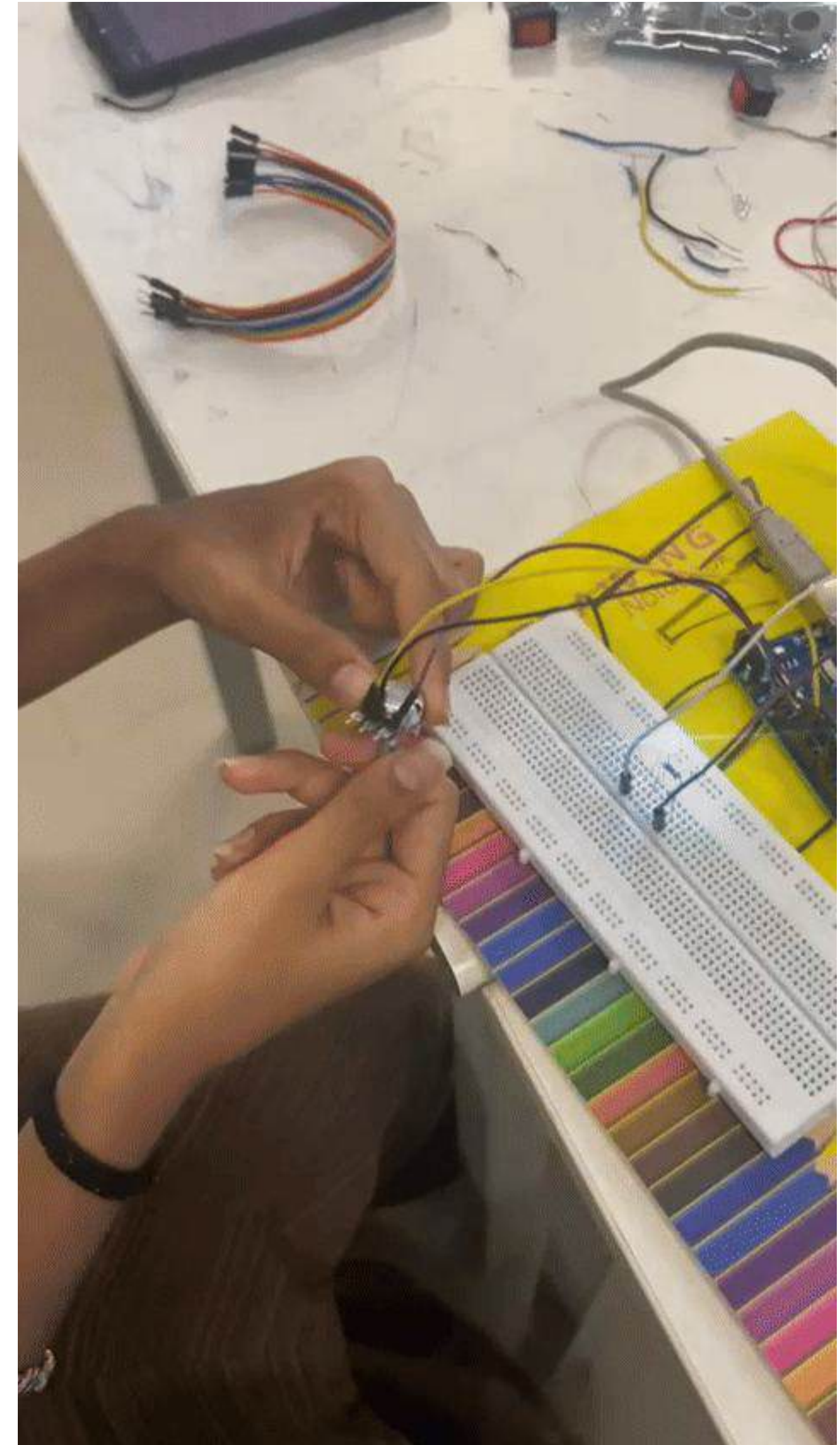
- Learned how to interface a push button with Arduino UNO.
- Understood the use of `digitalRead()` to detect button press status.
- Practiced using conditional statements (if-else) to control LED behavior.
- Observed how a button press can turn an LED ON or OFF.
- Gained insight into the importance of input (button) and output (LED) interaction.
- Strengthened understanding of basic digital logic in Arduino circuits.

Potentiometer with LED

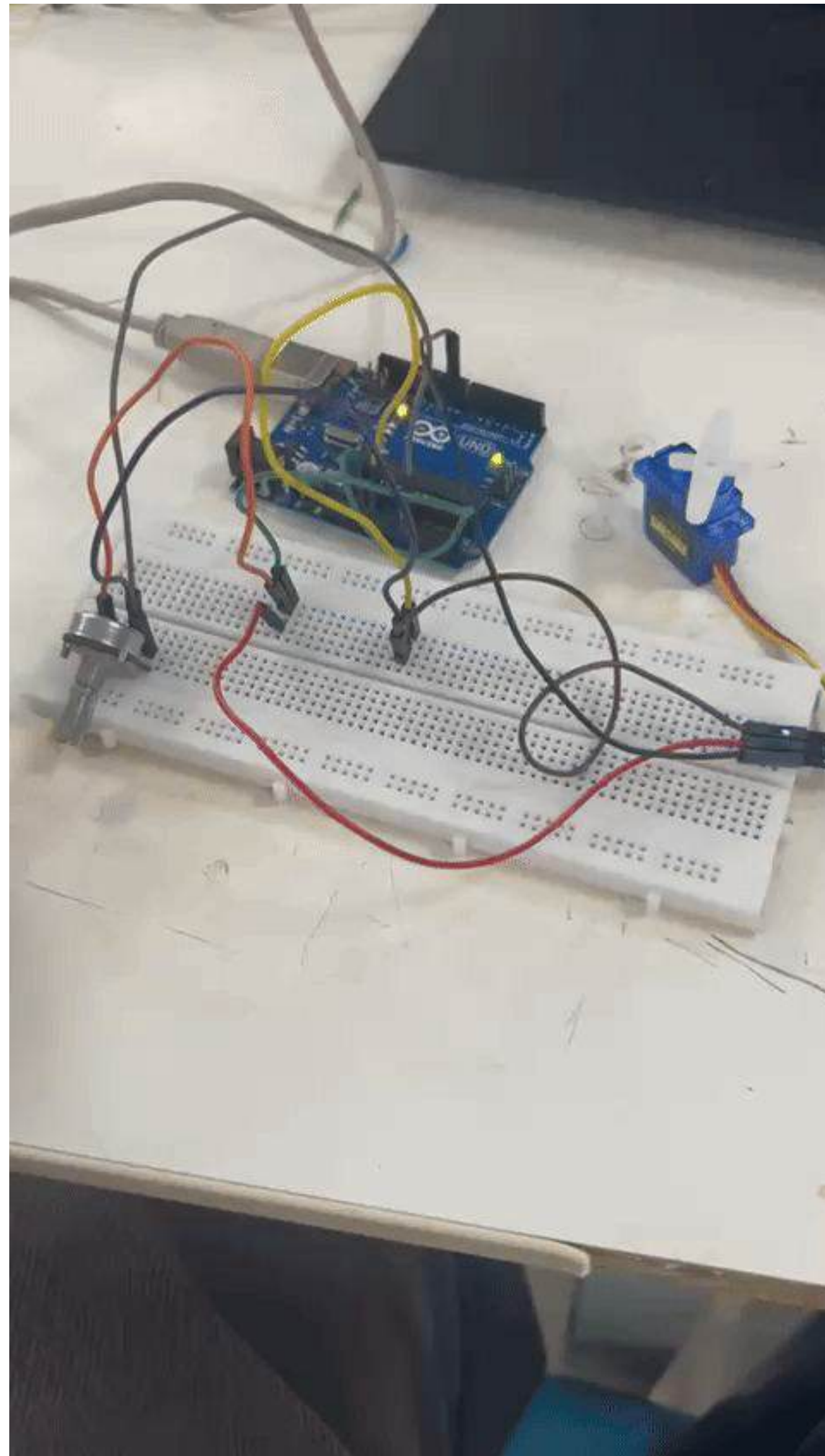


Our Learnings:

- Learned how to interface a potentiometer with Arduino UNO.
- Understood how to read analog values (0-1023) using `analogRead()`.
- Practiced mapping analog input values to LED brightness (0-255) using `map()`.
- Gained hands-on experience with PWM (Pulse Width Modulation) for controlling LED intensity.
- Observed how turning the potentiometer changes LED brightness gradually.
- Learned to use `analogWrite()` for analog output.
- Practiced using Serial Monitor for debugging and observing sensor values.



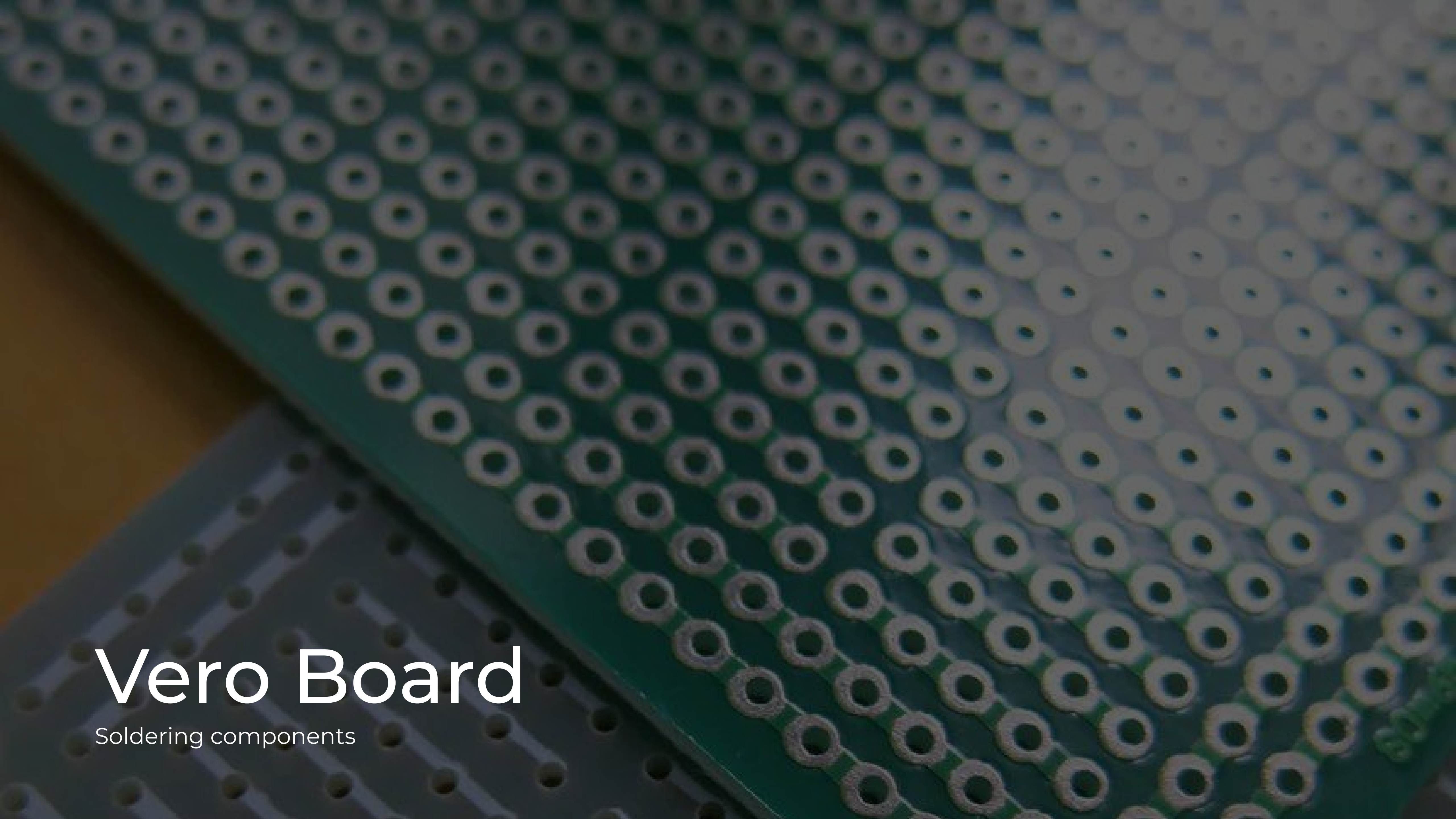
Potentiometer with servomotor



```
sketch  Tools  Help
Arduino Uno
nk.ino
1  #include <Servo.h>
2
3  Servo myservo; // create servo object to control a servo
4
5  int potpin = 0; // analog pin used to connect the potentiometer
6  int val;       // variable to read the value from the analog pin
7
8  void setup() {
9      myservo.attach(9); // attaches the servo on pin 9 to the servo object
10 }
11
12 void loop() {
13     val = analogRead(potpin); // reads the value of the potentiometer
14     val = map(val, 0, 1023, 0, 180); // scale it to use it with the servo (0 to 180)
15     myservo.write(val); // sets the servo position
16     delay(15); // waits for the servo to get there
17 }
```

Our Learnings:

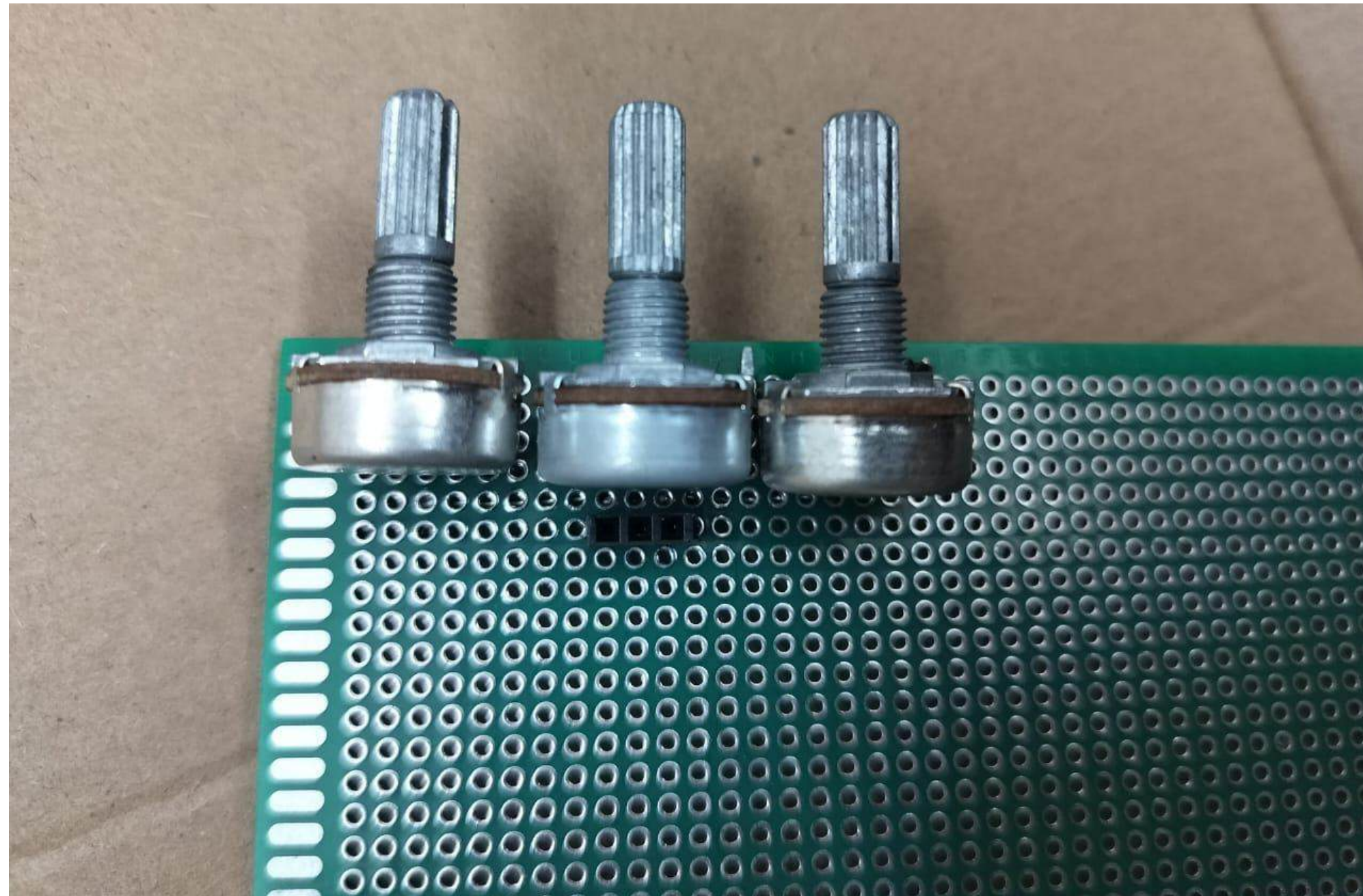
- Learned how to interface a potentiometer and servo motor with Arduino UNO.
- Understood how to use `analogRead()` to read variable resistance values from the potentiometer.
- Practiced using `map()` function to scale values from 0–1023 (potentiometer range) to 0–180 (servo angle).
- Learned how to control servo position smoothly using potentiometer input.
- Gained hands-on experience with the Servo library (`Servo.h`) for easier control.
- Observed real-time analog input to mechanical movement conversion.

A close-up, angled view of a green Veroboard, a type of single-sided printed circuit board. The board is covered in a dense grid of circular holes, which are used for soldering electronic components. The board is shown at an angle, with another similar board visible underneath it in the lower-left corner. The lighting is soft, highlighting the texture of the green solder mask and the metallic edges of the holes.

Veroboard

Soldering components

3 Potentiometer on vero board

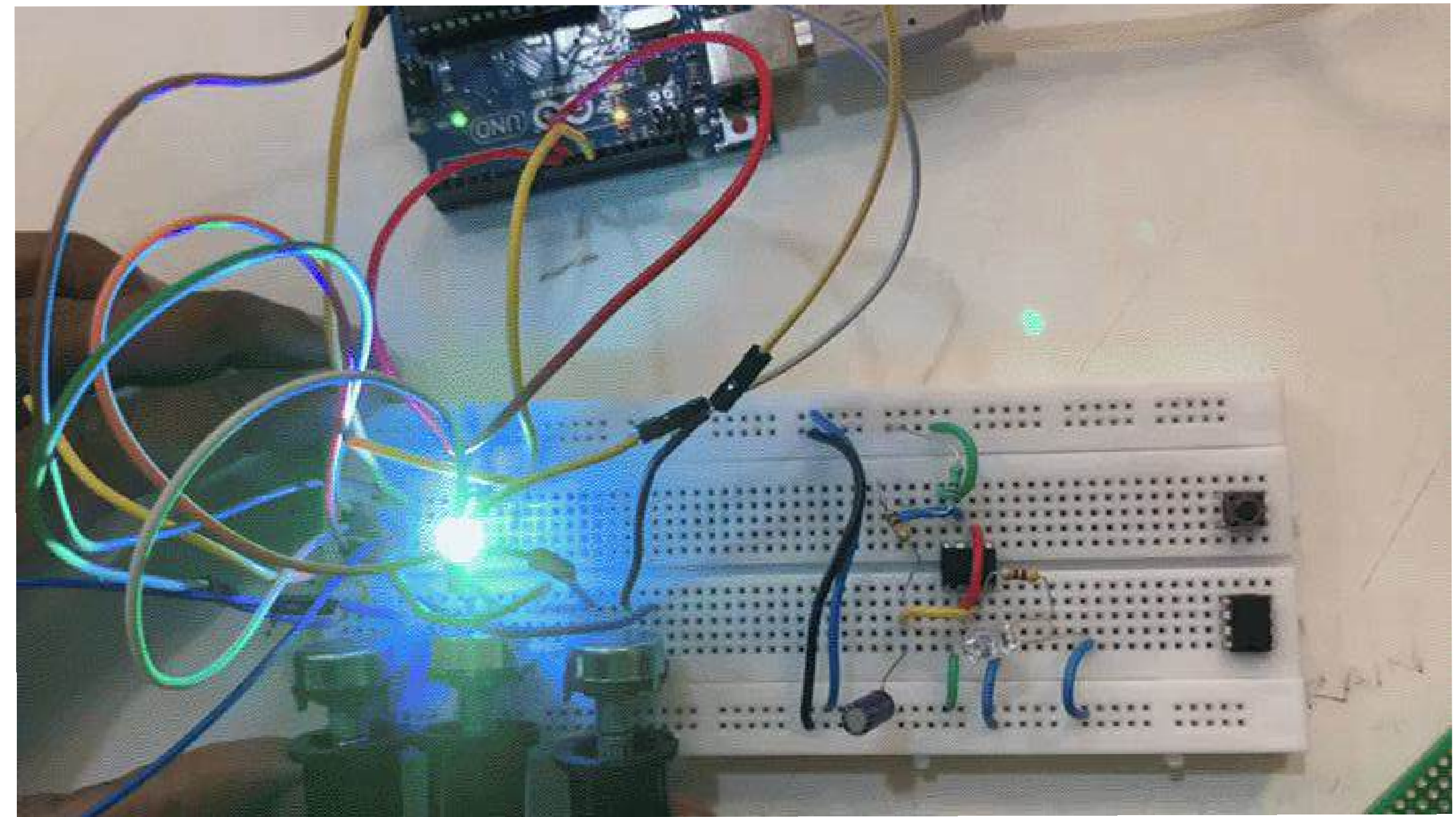


Small turns in potentiometer position caused noticeable changes in **LED color intensity**.

Soldered vero board connections are more reliable than breadboard

The **three potentiometers** provided independent control of **Red, Green, and Blue channels**.

Proper wiring and grounding were essential to avoid flickering or unstable outputs.



3 POTIENTOMETER WITH RGB tested on the breadboard

Our Learnings

- Learned how to mount potentiometers on a vero board for stable connections.
- Understood how potentiometers can control RGB LED brightness by varying resistance.
- Gained practice in breadboard prototyping before soldering on a vero board.
- Learned the role of analog input pins on Arduino/ESP32 for reading potentiometer values.
- Understood how multiple inputs can combine to control a single output (RGB color mixing).

3 LED with 3 [250 ohm] resistors

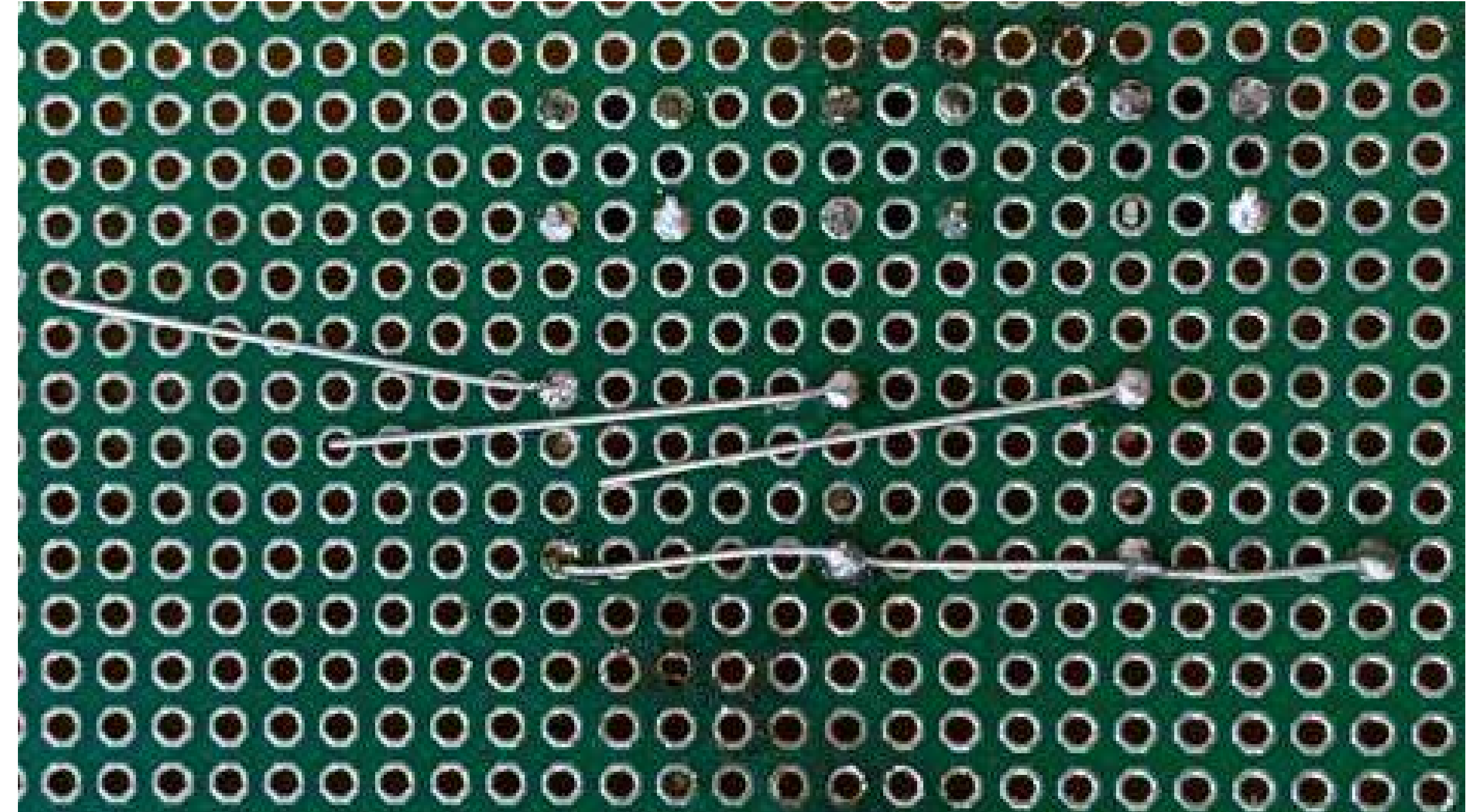
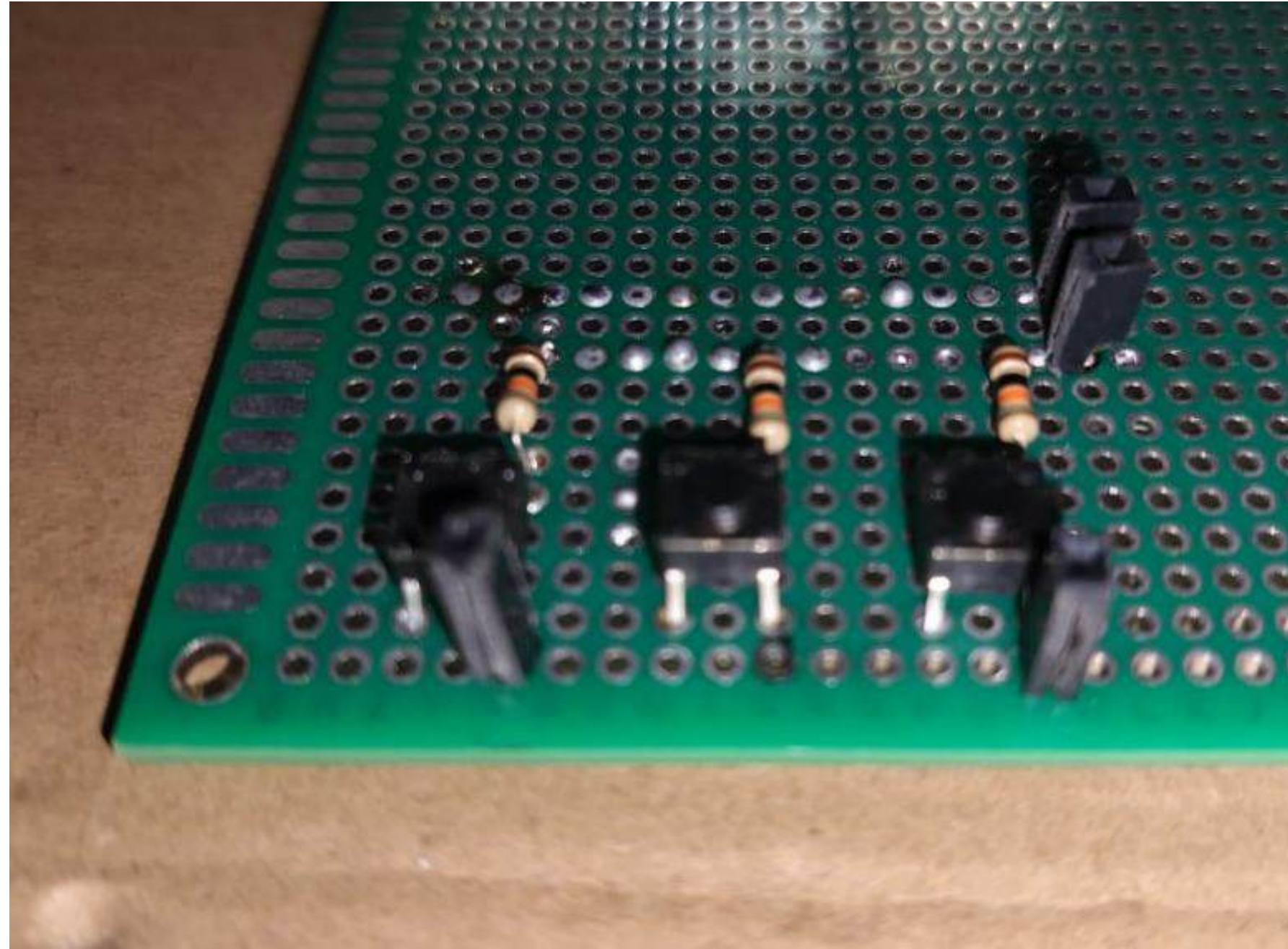


The brightness was safe and controlled, avoiding overheating.
Direct connections without resistors could have burnt the LEDs quickly.
Vero board mounting gave a stable and neat layout compared to breadboard wiring.

Our Learnings

- Learned that multiple LED needs its higher power resistor to limit current and prevent damage.
- Understood how resistor value affects LED brightness (250Ω gives moderate brightness).
- Gained practice in soldering LEDs and resistors on a vero board for permanent use.
- Learned the importance of correct polarity (anode/cathode) when connecting LEDs

3 Push buttons with 1K ohm resistance



Backside of soldering

The buttons provided stable input signals because of the 1K Ω pull-down resistors.

No false triggering was noticed when the circuit is pressed or released.

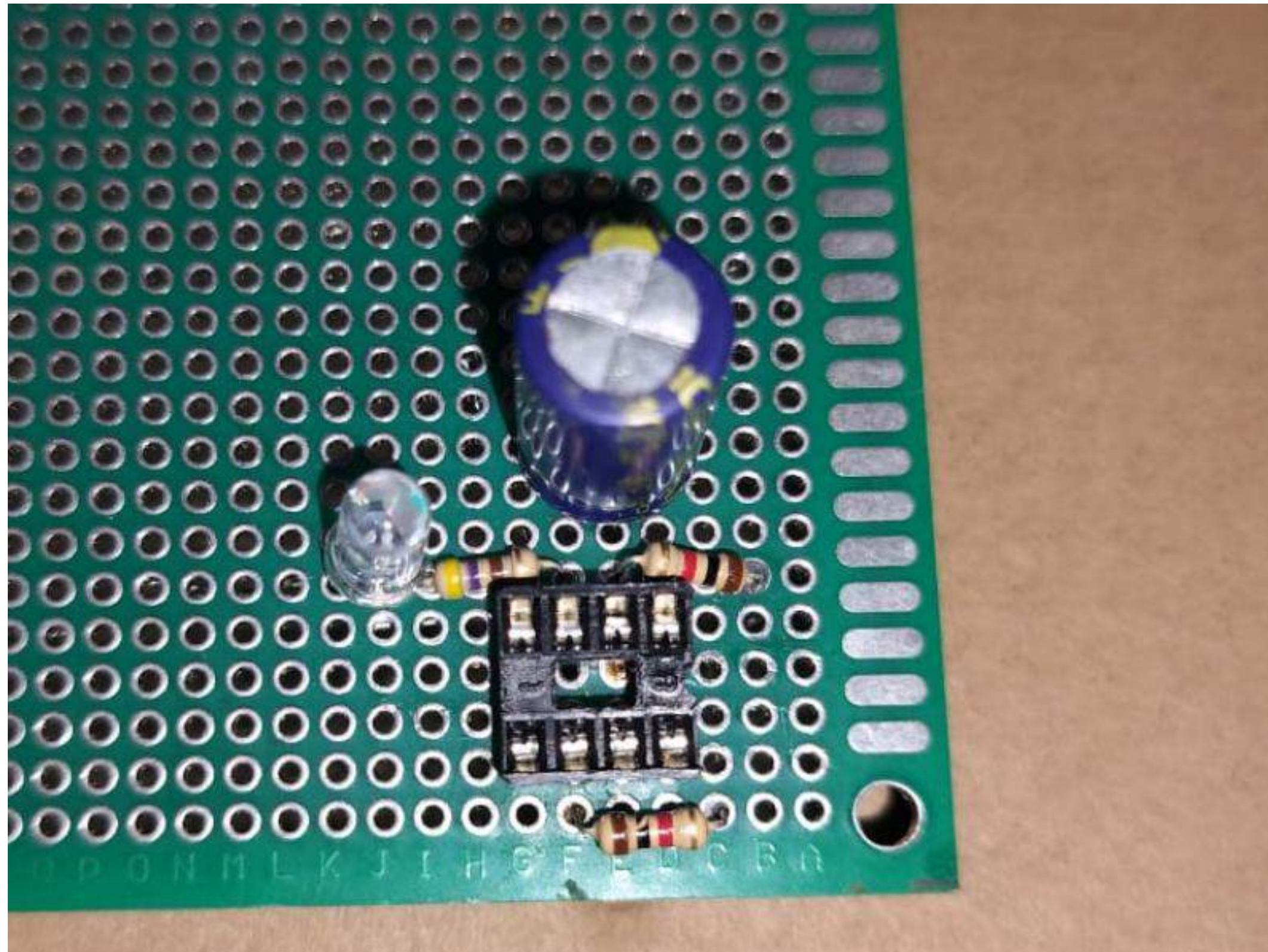
The placement on the veroboard makes it easy to expand (e.g., adding LEDs, buzzers, or microcontroller inputs).

Proper soldering ensured good continuity without short circuits.

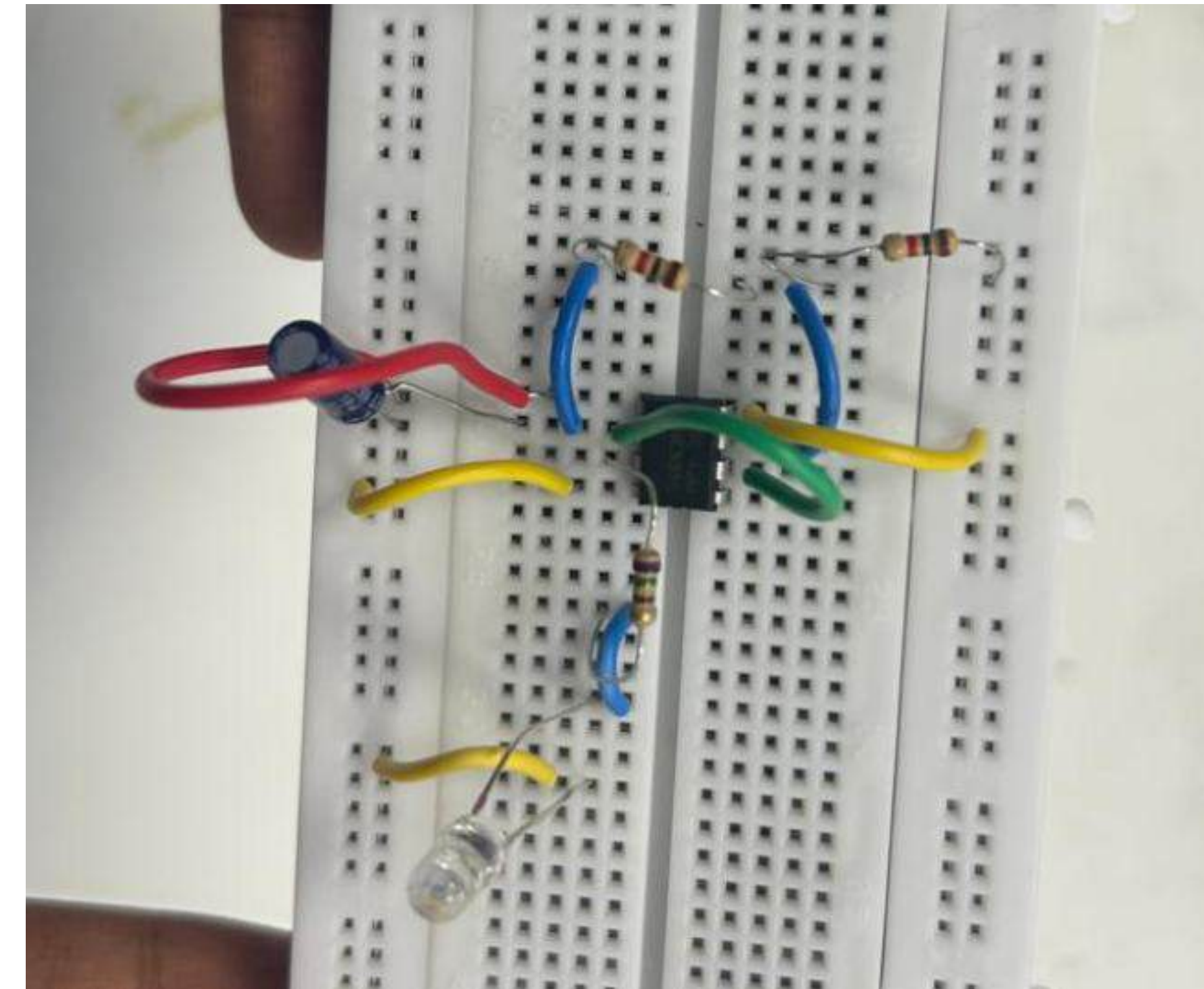
Our Learnings

- Learned how to interface push buttons with pull-down resistors (1K Ω used to avoid floating states).
- Understood that without resistors, input pins may read random HIGH/LOW values (floating).
- Practiced arranging and soldering buttons neatly on a veroboard.
- Learned the difference between normally open (NO) and normally closed (NC) button configurations.

IC-555



The LED connected to the 555 IC was blinking at a steady rate, confirming that the astable circuit worked.
By changing the resistor or capacitor values, the blinking speed (frequency) changed.
The breadboard version made it easier to test different component values quickly.
The vero board version provided a stronger, permanent circuit but required careful soldering to avoid shorts.



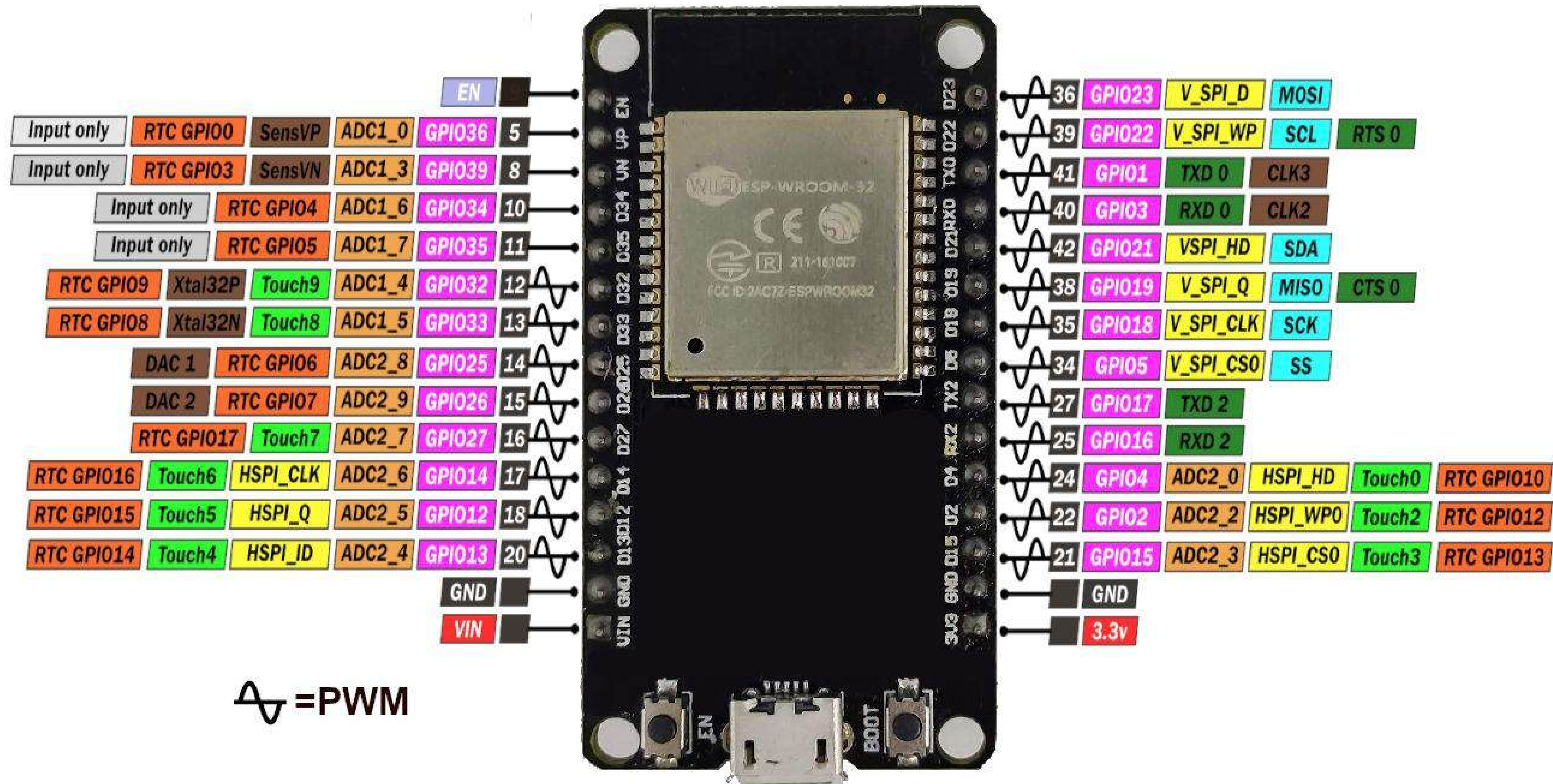
Our Learnings

- Learned the pin configuration of the 555 timer IC (VCC, GND, Trigger, Threshold, Control, Discharge, Output, Reset).
- Understood the Astable mode (for generating square waves / blinking LED) and Monostable mode (for timer applications).
- Learned how resistors and capacitors connected to the 555 IC control time delay and frequency.
- Practiced circuit building on both vero board (permanent soldered version) and breadboard (testing/temporary version).

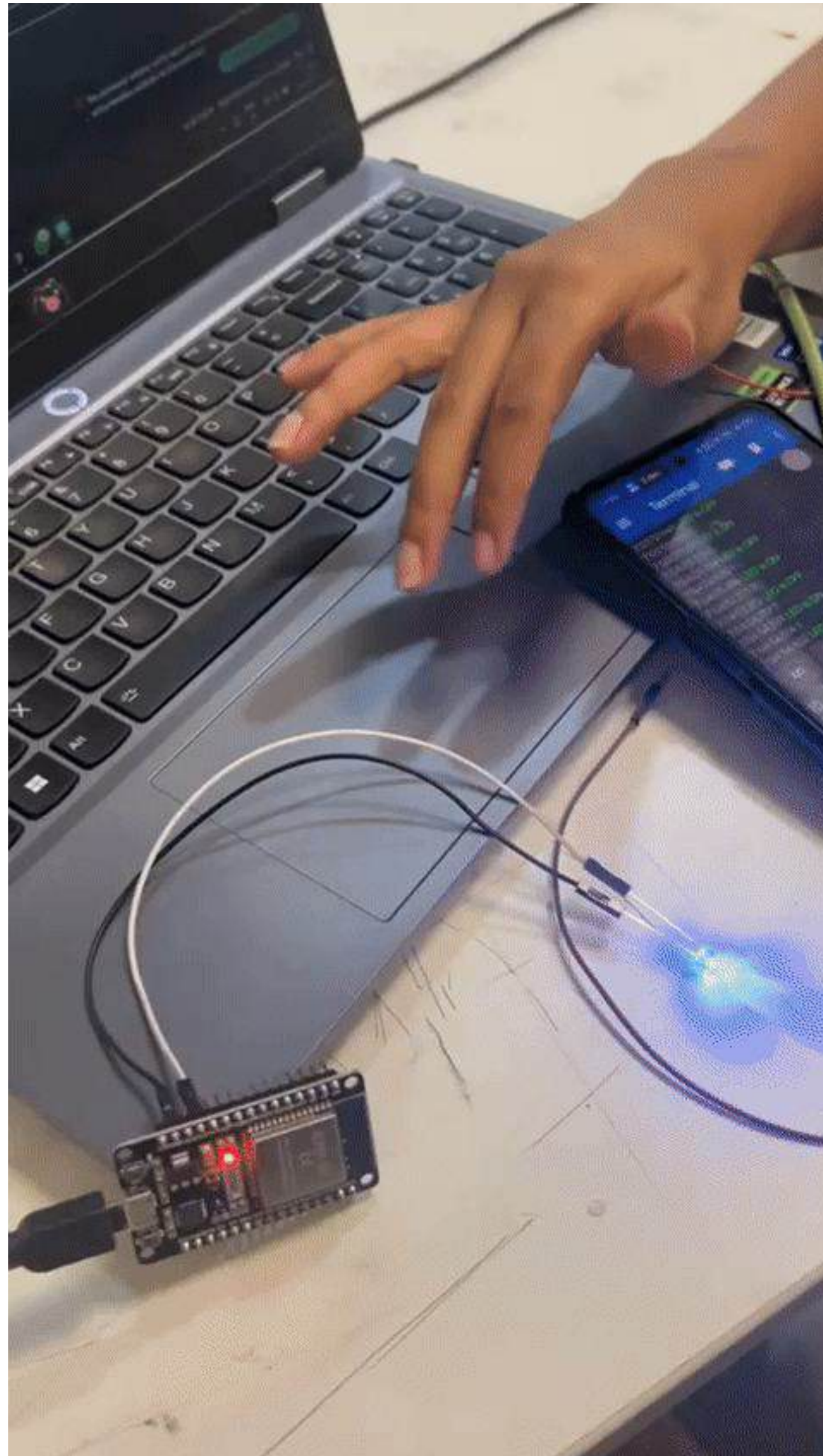
A close-up, angled view of an ESP-32 Devkit V1 development board. The board is populated with various components including a USB Type-C port, a micro-USB port, a push button, several integrated circuits, and numerous surface-mount components. The board is populated with various components including a USB Type-C port, a micro-USB port, a push button, several integrated circuits, and numerous surface-mount components. The board is populated with various components including a USB Type-C port, a micro-USB port, a push button, several integrated circuits, and numerous surface-mount components. The board is populated with various components including a USB Type-C port, a micro-USB port, a push button, several integrated circuits, and numerous surface-mount components.

ESP- 32 Devkit V1 (with bluetooth)

Pinout diagram of the ESP32 DevKit board.



ESP-32 connected with 1 LED bulb via bluetooth



ESP 32 is connected to an led, it can be controlled by serial bluetooth terminal app

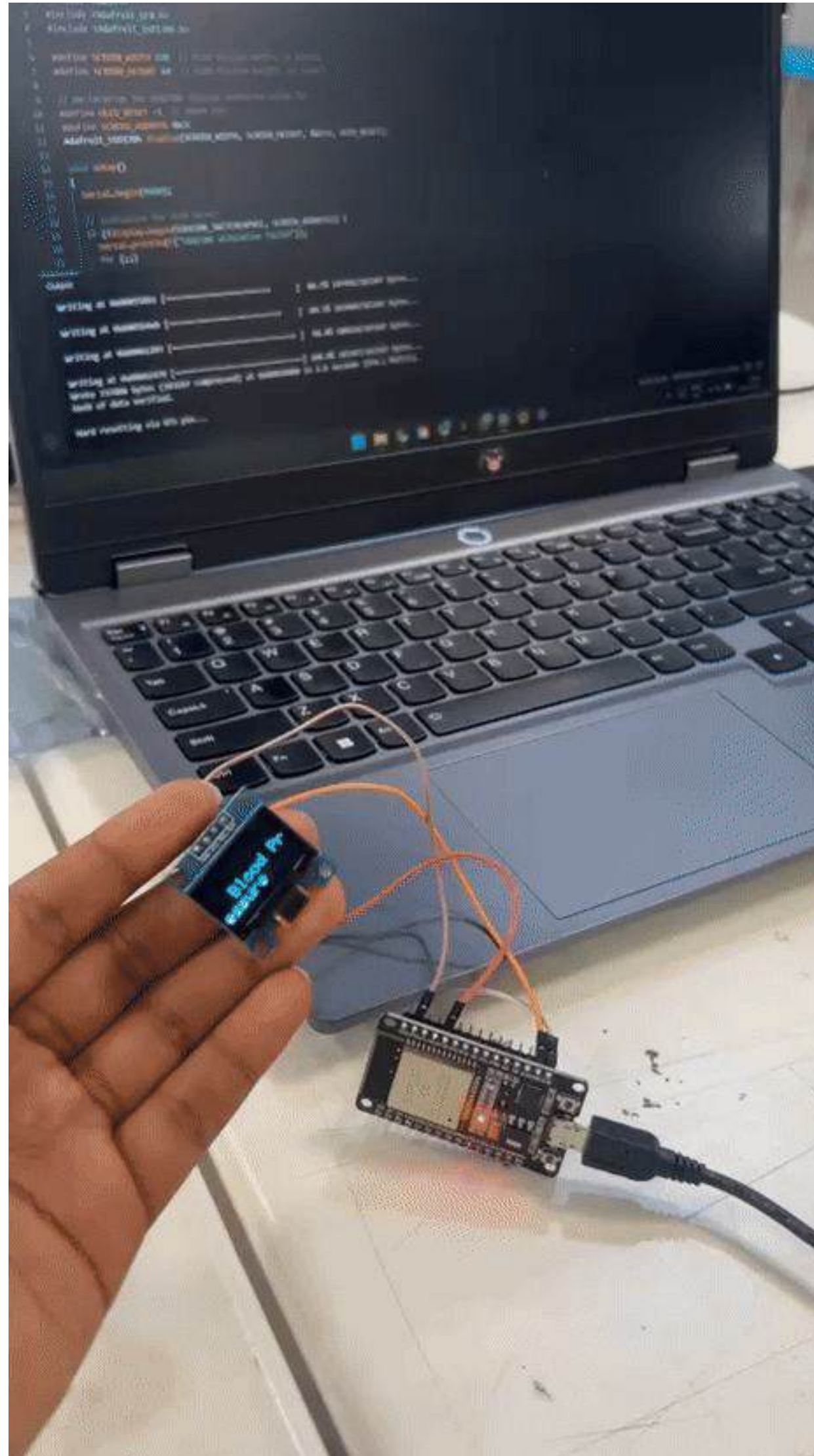
```
5 #error Bluetooth is not enabled! Please run make menuconfig to enable it
6 #endif
7
8 // Create BluetoothSerial object
9 BluetoothSerial SerialBT;
10
11 // Define LED pin
12 const int ledPin = 12; // GPIO4
13 void setup() {
14     // Initialize LED pin as output
15     pinMode(ledPin, OUTPUT);
16     digitalWrite(ledPin, LOW); // LED off initially
17
18     // Initialize Serial for debugging
19     Serial.begin(115200);
20
21     // Initialize Bluetooth with device name
22     SerialBT.begin("ESP32-BT-tc"); // Bluetooth device name
23     Serial.println("Bluetooth Started! Pair with ESP32-BT-tc");
24 }
25
26 void loop() {
27     // Check if data is available from Bluetooth
28     if (SerialBT.available()) {
29         char incomingChar = SerialBT.read();
30
31         // Ignore newline (\n) and carriage return (\r)
32         if (incomingChar == '\n' || incomingChar == '\r') {
33             return; // Skip processing these characters
34         }
35     }
36 }
```

Coding on arduino ide

Our learnings:

- Learned to pair ESP32 with phone via Bluetooth.
- Controlled LED using commands from app.
- Programmed ESP32 with Arduino IDE (BluetoothSerial).
- Understood GPIO setup and hardware response.

ESP-32 with OLED display



```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET -1 // Reset pin
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
&Wire, OLED_RESET);

void setup()
{
  Serial.begin(9600);

  // initialize the OLED object
  if (!display.begin(SSD1306_SWITCHCAPVCC,
SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ; // Don't proceed, loop forever
  }

  // Clear the buffer.
  display.clearDisplay();

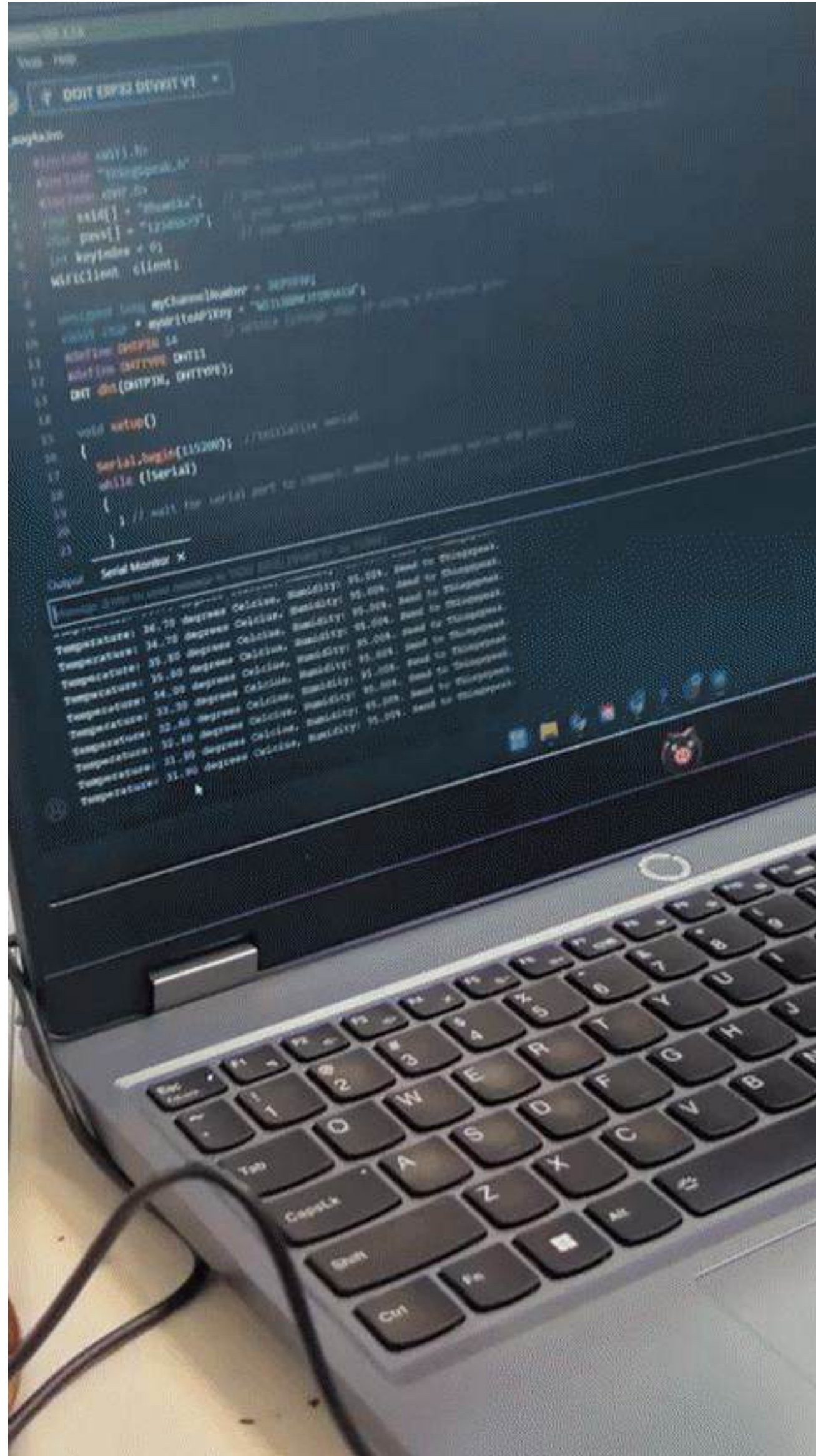
  // Display Text
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(28, 28);
  display.println("Blood Pressure");
  display.display();
  delay(2000);
  display.clearDisplay();
}

void loop()
{
}
```

Our learnings:

- OLED Integration: Learned how to connect and initialize an OLED screen with the ESP32 using the Adafruit_SSD1306 and Adafruit_GFX libraries.
- Display Control: Gained experience in writing text, adjusting font size, positioning text with `setCursor()`, and refreshing the display buffer.
- Serial + Visual Feedback: Understood how to combine serial monitor debugging with on-screen feedback for clearer system status.
- Scalability: Realized that an OLED can show sensor readings, messages, or status indicators

ESP-32 with DHT sensor



```
#include <WiFi.h>
#include "ThingSpeak.h" // always include thingspeak header file after other header files and custom macros
#include <DHT.h>
char ssid[] = "mayank"; // your network SSID (name)
char pass[] = "mayank123"; // your network password
int keyIndex = 0; // your network key Index number (needed only for WEP)
WiFiClient client;

unsigned long myChannelNumber = 3025027;
const char * myWriteAPIKey = "6E7GYYSYY7QEFCU1";
#define DHTPIN 14 // GPIO14 (Change this if using a different pin)
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
  Serial.begin(115200); //Initialize serial
  while (!Serial)
  {
    ; // wait for serial port to connect. Needed for Leonardo native USB port only
  }

  dht.begin();
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client); // Initialize ThingSpeak
}

void loop()
{
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Connect or reconnect to WiFi
  if(WiFi.status() != WL_CONNECTED)
  {
    Serial.print("Attempting to connect to SSID: ");

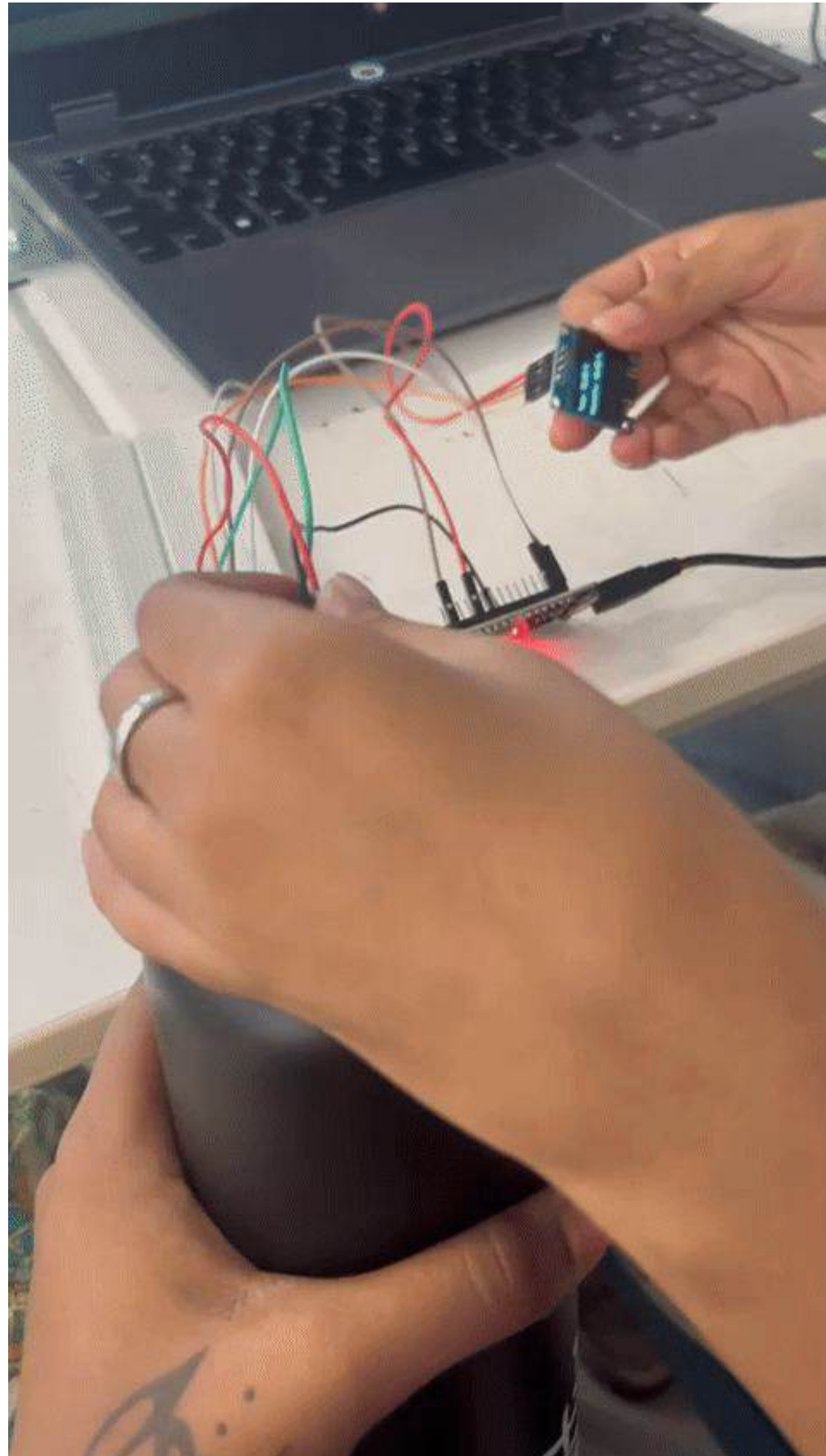
    while(WiFi.status() != WL_CONNECTED)
    {
      WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this line if using open or WEP network
      Serial.print(".");
      delay(5000);
    }
    Serial.println("\nConnected.");
  }
  ThingSpeak.setField(1, t);
  ThingSpeak.setField(2, h);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" degrees Celcius, Humidity: ");
  Serial.print(h);
  Serial.println("% Send to Thingspeak.");

  delay(15000); // Wait 15 seconds to update the channel again
}
```

Our learnings:

- DHT Sensor: Measures temperature and humidity.
- ESP32 + Wi-Fi: Connects to Wi-Fi for IoT applications.
- ThingSpeak Integration: Sends sensor data to cloud for monitoring.
- Serial Monitor: Displays real-time temperature & humidity readings.
- Practical Use: Foundation for smart environment monitoring in projects.

ESP-32 with OLED display and DHT sensor



```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "DHT.h"

// OLED setup
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_ADDRESS 0x3C // I2C address
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// DHT setup
#define DHTPIN 14 // DHT data pin connected to GPIO14
#define DHTTYPE DHT11 // DHT11 selected here

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();

  // OLED begin
  if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed!"));
    for (;;) // Loop forever if OLED fails
  }

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.println("DHT11 Sensor Init...");
  display.display();
  delay(2000);
}

void loop() {
  float humidity = dht.readHumidity();
  float tempC = dht.readTemperature(); // Default is Celsius

  // Check for failed reads
  if (isnan(humidity) || isnan(tempC)) {
    Serial.println("Failed to read from DHT11 sensor!");
    display.clearDisplay();
    display.setCursor(0, 10);
    display.println("Sensor Error!");
    display.display();
    delay(2000);
    return;
  }

  // Clear screen and display values
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  display.setCursor(0, 10);
  display.print("Temp: ");
  display.print(tempC);
  display.println(" C");

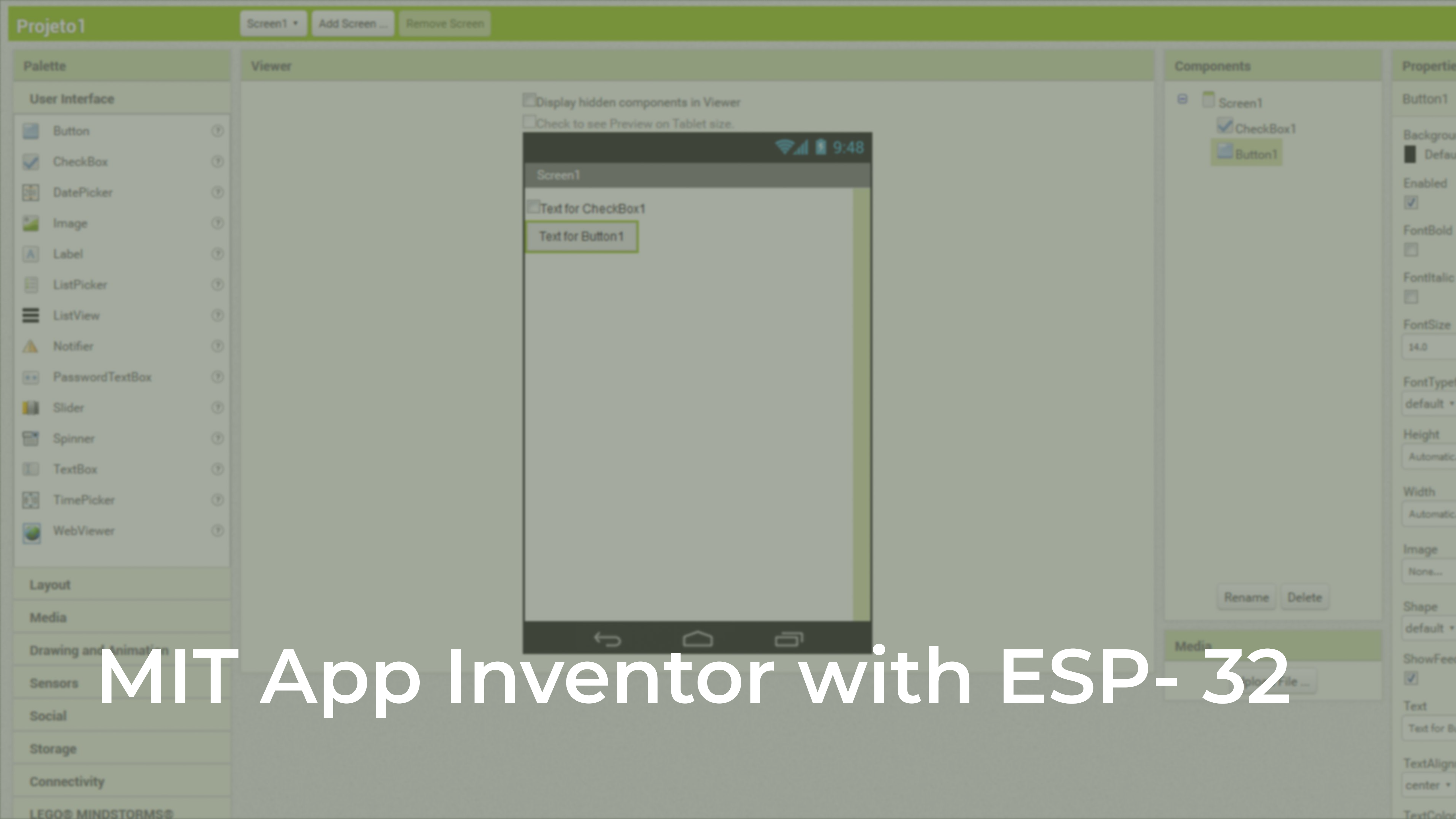
  display.setCursor(0, 30);
  display.print("Humidity: ");
  display.print(humidity);
  display.println(" %");

  display.display();

  delay(500); // Update every 2 seconds
}
```

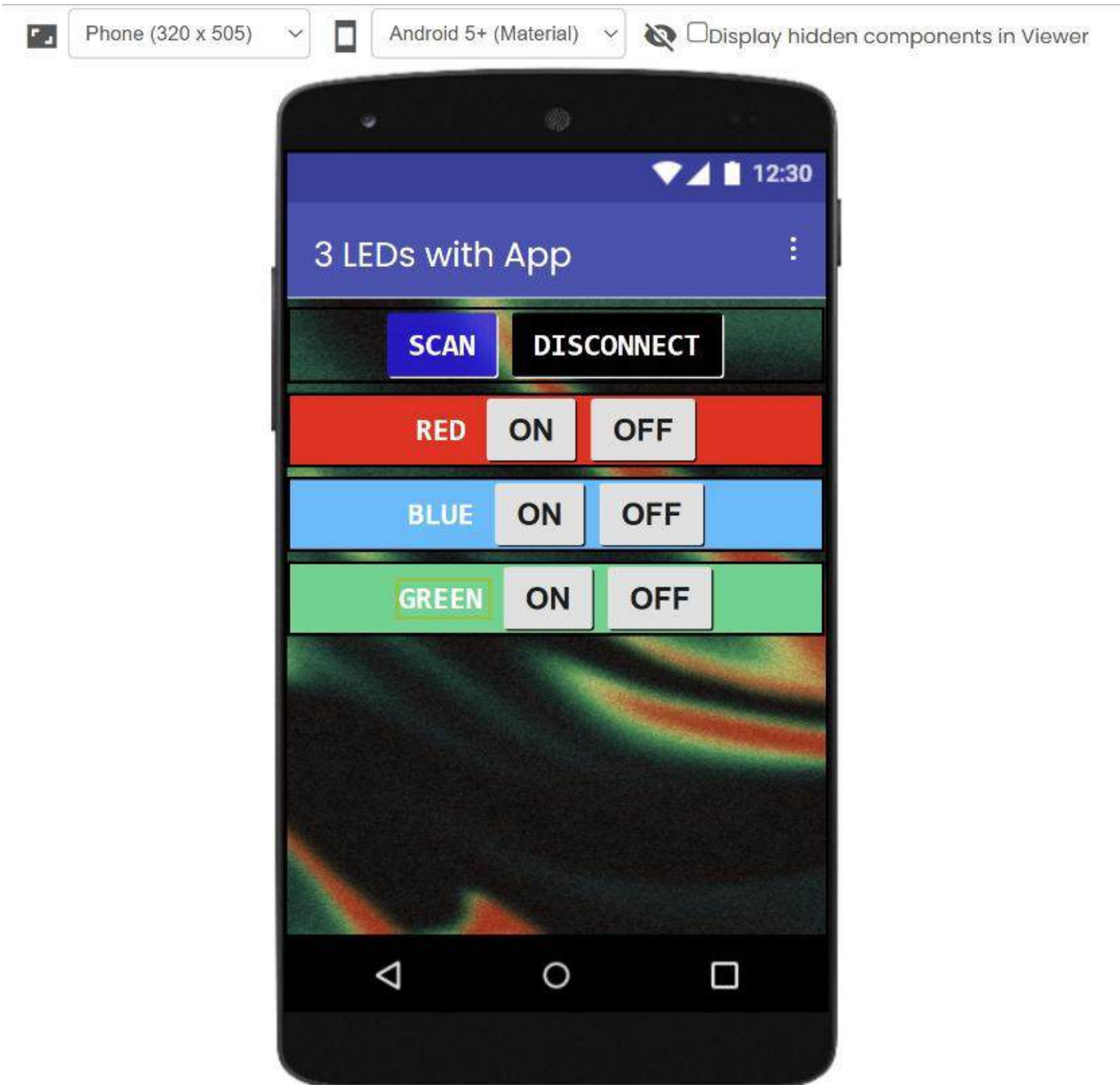
Our learnings:

- Learned how to interface ESP-32 with an OLED display and DHT11/DHT22 sensor.
- Understood how to use Adafruit libraries (Adafruit_GFX, Adafruit_SSD1306) for displaying text on the OLED.
- Practiced connecting sensors via I²C protocol and configuring pins.
- Learned to read temperature and humidity values using the DHT sensor.
- Implemented error handling when sensor values fail to read.
- Gained skills in displaying dynamic data (temperature °C, humidity %) on OLED in real time.
- Observed how ESP-32 can combine sensing and display functions, making it useful for IoT applications.



MIT App Inventor with ESP-32

RGB LED with MIT app



```
when ListPicker1.BeforePicking
do set ListPicker1.Elements to BluetoothClient1.AddressesAndNames

when ListPicker1.AfterPicking
do if call BluetoothClient1.Connect
    address ListPicker1.Selection
    then set ListPicker1.Elements to BluetoothClient1.AddressesAndNames

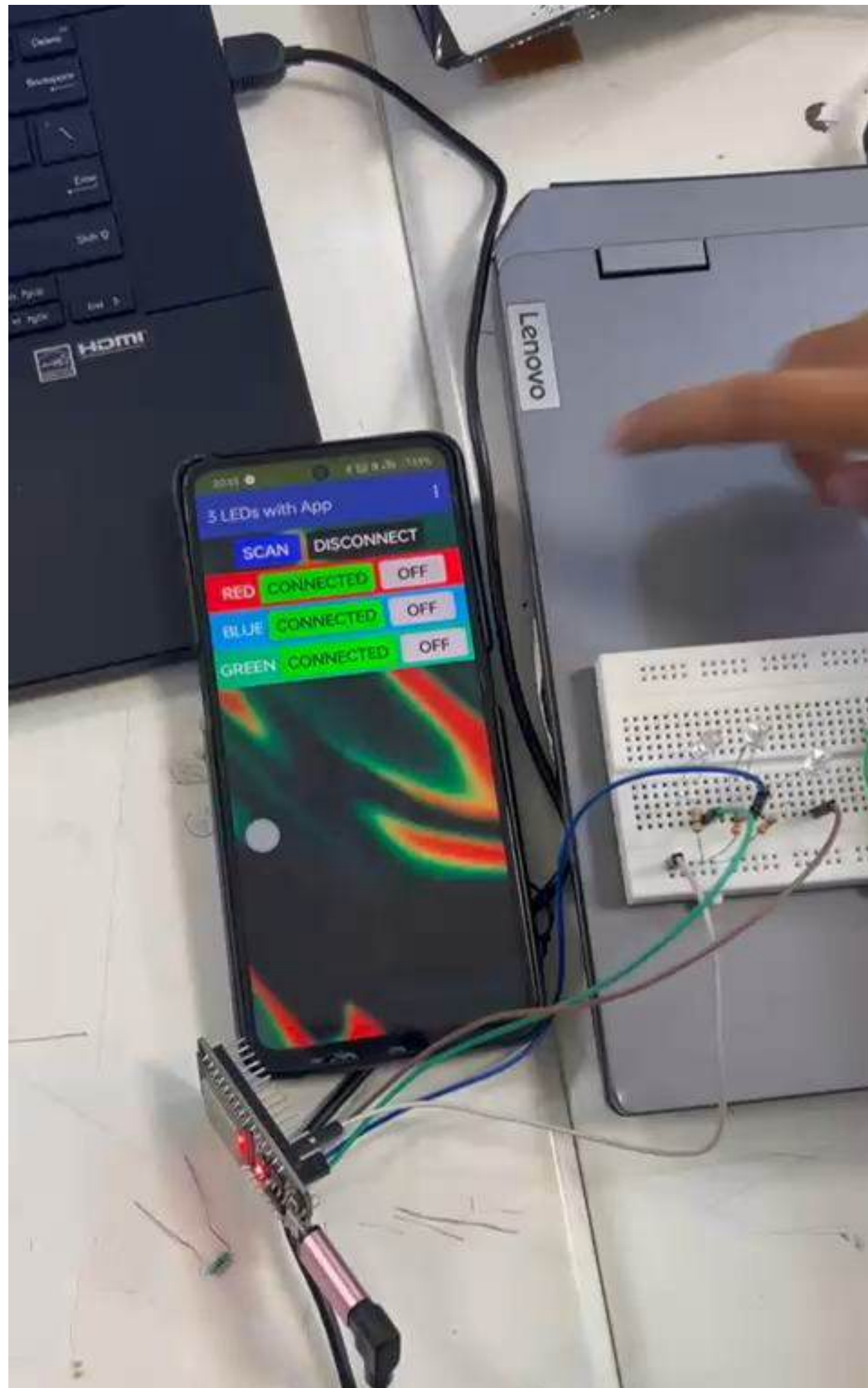
when Button2.Click
do call BluetoothClient1.Disconnect

when Clock1.Timer
do if BluetoothClient1.IsConnected
    then set Button3.Text to "CONNECTED"
    set Button3.BackgroundColor to green
    if not BluetoothClient1.IsConnected
    then set Button3.Text to "NOT CONNECTED"
    set Button3.BackgroundColor to red
    if BluetoothClient1.IsConnected
    then set Button5.Text to "CONNECTED"
    set Button5.BackgroundColor to green
    if not BluetoothClient1.IsConnected
    then set Button5.Text to "NOT CONNECTED"
    set Button5.BackgroundColor to red
    if BluetoothClient1.IsConnected
    then set Button7.Text to "CONNECTED"
    set Button7.BackgroundColor to green
    if not BluetoothClient1.IsConnected
    then set Button7.Text to "NOT CONNECTED"
    set Button7.BackgroundColor to red

when Button4.Click
do call BluetoothClient1.SendText
    text "1"

when Button6.Click
do call BluetoothClient1.SendText
    text "2"

when Button8.Click
do call BluetoothClient1.SendText
    text "3"
```

Our Learnings

- Understood the wiring differences between common cathode and common anode RGB LEDs.
- Implemented BluetoothSerial on ESP32 to wirelessly receive commands.
- Designed a toggle-based control system for individual LED colors.
- Gained experience in state tracking for maintaining LED ON/OFF status.
- Improved debugging by sending feedback messages over Bluetooth.

```
#include <BluetoothSerial.h>

BluetoothSerial SerialBT;

// Pins for RGB LED anodes (common anode → VCC)
const int redLED = 12;
const int greenLED = 14;
const int blueLED = 27;

// Track LED states
bool redState = false;
bool greenState = false;
bool blueState = false;

void setup() {
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(blueLED, OUTPUT);

  // Turn all LEDs OFF initially (HIGH = OFF for common anode)
  digitalWrite(redLED, HIGH);
  digitalWrite(greenLED, HIGH);
  digitalWrite(blueLED, HIGH);

  Serial.begin(115200);
  SerialBT.begin("ESP32-RGB"); // Bluetooth name
  Serial.println("Bluetooth ready. Send 1 (Red), 2 (Green), 3 (Blue) to toggle.");
}

void loop() {
  if (SerialBT.available()) {
    char incomingChar = SerialBT.read();

    if (incomingChar == '\n' || incomingChar == '\r') return;

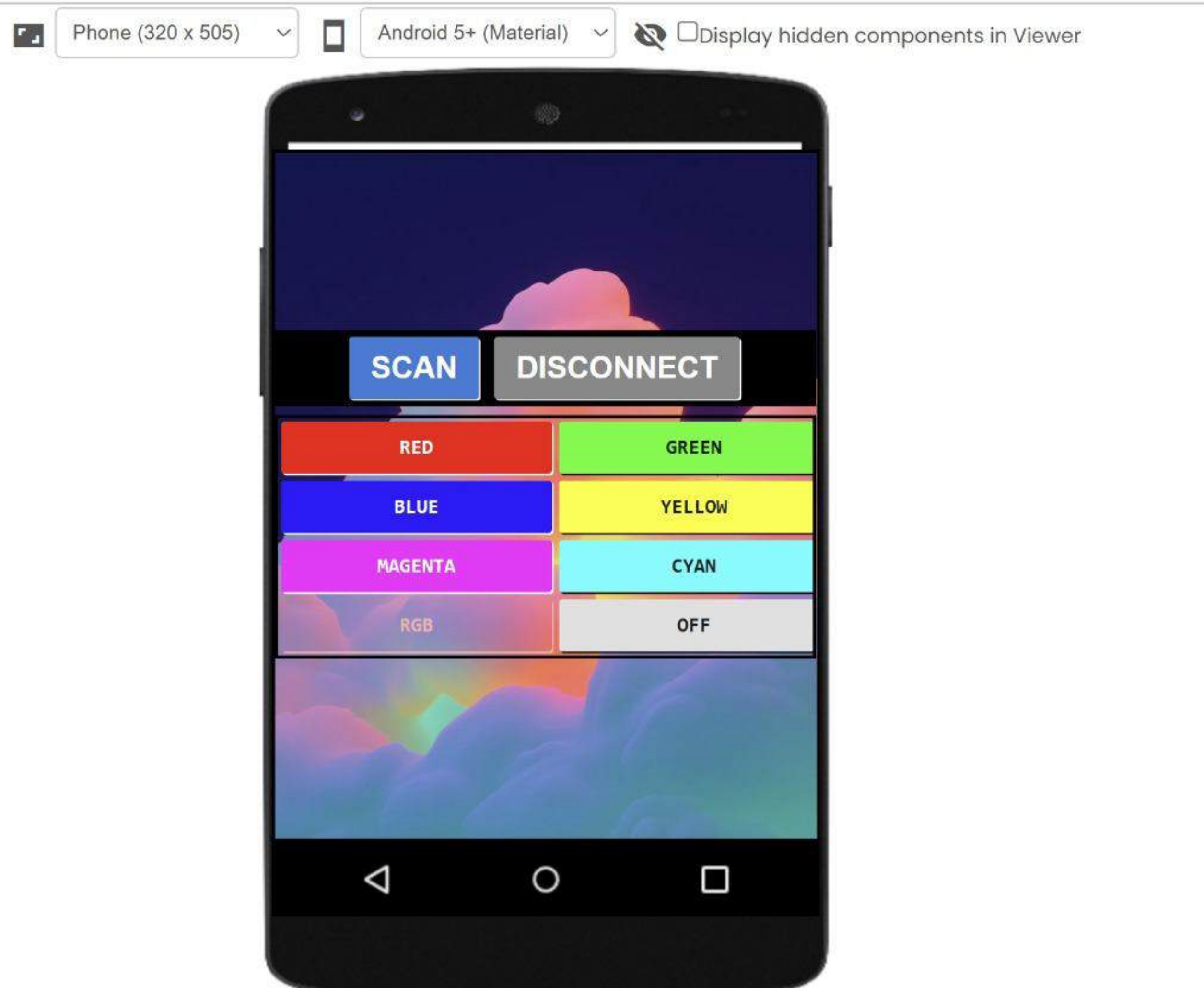
    switch (incomingChar) {
      case '1': // Toggle Red
        redState = !redState;
        digitalWrite(redLED, redState ? LOW : HIGH); // LOW = ON for common anode
        SerialBT.println(redState ? "Red ON" : "Red OFF");
        break;

      case '2': // Toggle Green
        greenState = !greenState;
        digitalWrite(greenLED, greenState ? LOW : HIGH);
        SerialBT.println(greenState ? "Green ON" : "Green OFF");
        break;

      case '3': // Toggle Blue
        blueState = !blueState;
        digitalWrite(blueLED, blueState ? LOW : HIGH);
        SerialBT.println(blueState ? "Blue ON" : "Blue OFF");
        break;

      default:
        SerialBT.println("Send 1=Red, 2=Green, 3=Blue");
        break;
    }
  }
}
```


Neopixel with bluetooth



```
when ListPicker1.BeforePicking
do
  set ListPicker1.Elements to BluetoothClient1.AddressesAndNames

when ListPicker1.AfterPicking
do
  if call BluetoothClient1.Connect
    address ListPicker1.Selection
  then
    set ListPicker1.Elements to BluetoothClient1.AddressesAndNames

when Clock1.Timer
do
  if BluetoothClient1.IsConnected
  then
    set Button3.Text to "CONNECTED"
    set Button3.BackgroundColor to #00FF00
  if not BluetoothClient1.IsConnected
  then
    set Button3.Text to "NOT CONNECTED"
    set Button3.BackgroundColor to #FF0000

when Button11.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "RED"

when Button10.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "GREEN"

when Button12.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "BLUE"

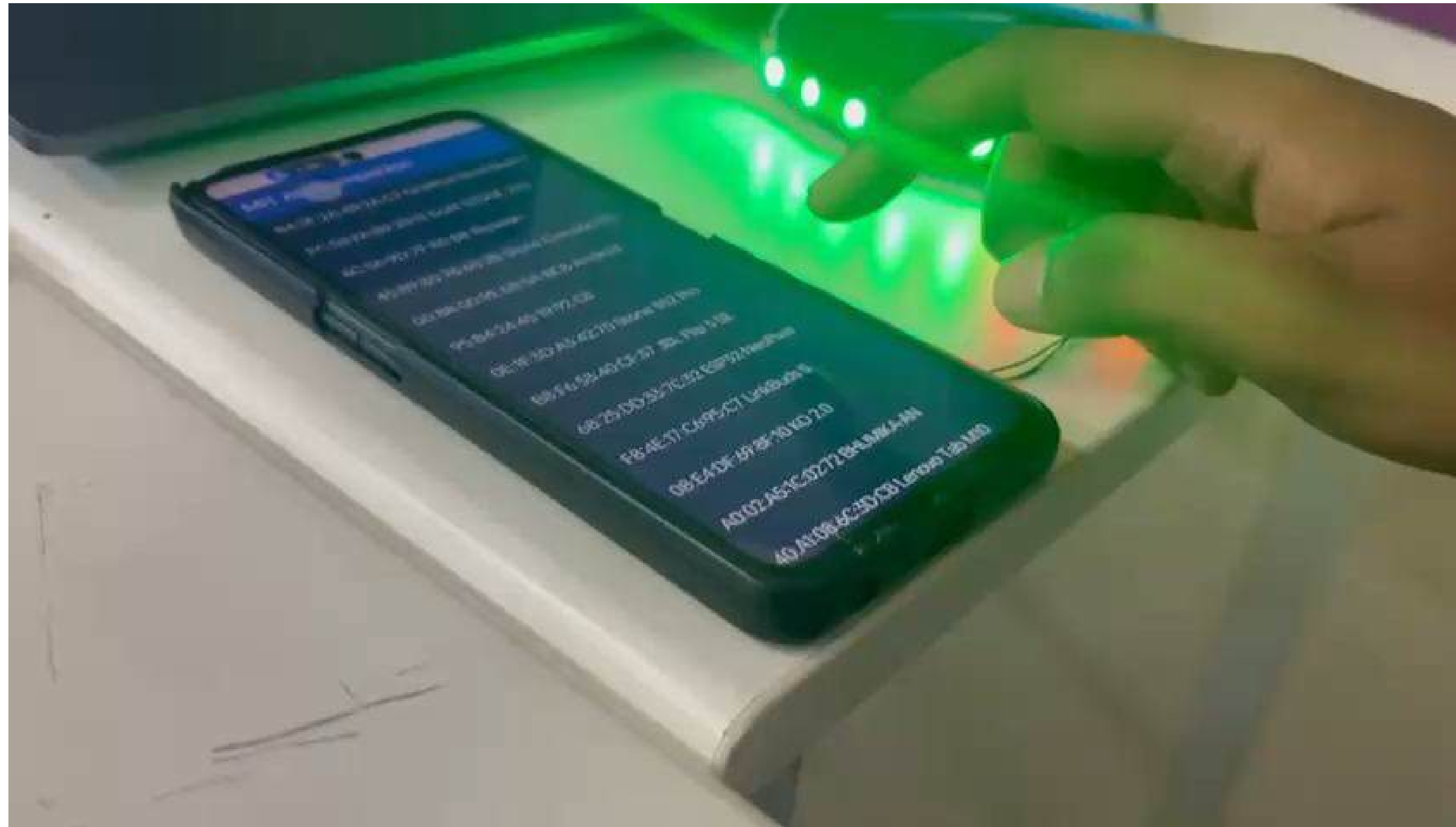
when Button13.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "YELLOW"

when Button14.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "MAGENTA"

when Button15.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "CYAN"

when Button16.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "WHITE"

when Button16.Click
do
  if BluetoothClient1.IsConnected
  then
    call BluetoothClient1.SendText
    text "OFF"
```

Our Learnings from this code:

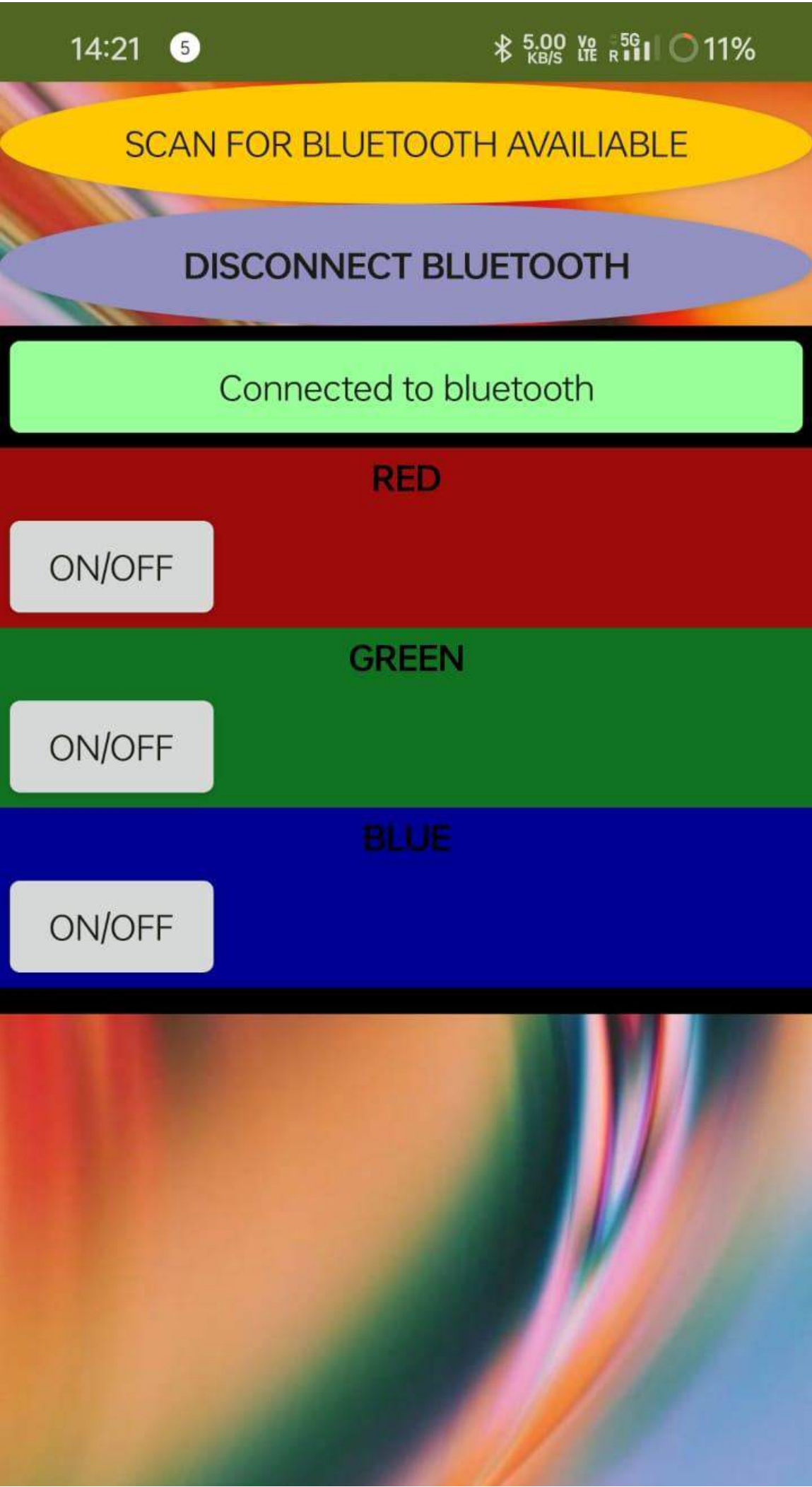
- Learned how to control NeoPixel LEDs with an ESP32 using Bluetooth.
- Understood how to send color commands from a phone and change LEDs instantly.
- Implemented a timer to return to animation mode when no commands are received.
- Discovered that breaking the code into smaller functions makes it easier to read and modify.

```

1 #include <Adafruit_NeoPixel.h>
2 #include <BluetoothSerial.h>
3
4 #define LED_PIN 5           // GPIO connected to NeoPixel DIN
5 #define NUM_LEDS 8         // Number of LEDs in strip or ring
6 #define ANIMATION_TIMEOUT 5000 // ms before returning to animation mode
7
8 Adafruit_NeoPixel strip(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);
9 BluetoothSerial SerialBT;
10
11 unsigned long lastCommandTime = 0;
12 bool animationMode = true;
13
14 // ===== SETUP =====
15 void setup() {
16   strip.begin();
17   strip.show(); // Initialize all pixels off
18   Serial.begin(115200);
19   SerialBT.begin("ESP32-NeoPixel");
20   Serial.println("Bluetooth ready. Waiting for commands...");
21 }
22
23 // ===== LOOP =====
24 void loop() {
25   if (SerialBT.available()) {
26     String cmd = SerialBT.readStringUntil('\n');
27     cmd.trim();
28     handleCommand(cmd);
29     lastCommandTime = millis();
30     animationMode = false;
31   }
32
33   // Return to animation after timeout
34   if (!animationMode && millis() - lastCommandTime > ANIMATION_TIMEOUT) {
35     animationMode = true;
36   }
37
38   if (animationMode) {
39     runAnimation();
40   }
41 }
42
43 // ===== HANDLE BLUETOOTH COMMAND =====
44 void handleCommand(String cmd) {
45   Serial.println("Command: " + cmd);
46   uint32_t color = strip.Color(0, 0, 0);
47
48   if (cmd.equalsIgnoreCase("RED")) color = strip.Color(255, 0, 0);
49   else if (cmd.equalsIgnoreCase("GREEN")) color = strip.Color(0, 255, 0);
50   else if (cmd.equalsIgnoreCase("BLUE")) color = strip.Color(0, 0, 255);
51   else if (cmd.equalsIgnoreCase("YELLOW")) color = strip.Color(255, 255, 0);
52   else if (cmd.equalsIgnoreCase("MAGENTA")) color = strip.Color(255, 0, 255);
53   else if (cmd.equalsIgnoreCase("CYAN")) color = strip.Color(0, 255, 255);
54   else if (cmd.equalsIgnoreCase("WHITE")) color = strip.Color(255, 255, 255);
55   else if (cmd.equalsIgnoreCase("OFF")) color = strip.Color(0, 0, 0);
56
57   for (int i = 0; i < strip.numPixels(); i++) {
58     strip.setPixelColor(i, color);
59   }
60   strip.show();
61 }
62
63 // ===== ANIMATION MODE =====
64 void runAnimation() {
65   colorWipe(strip.Color(255, 0, 0), 500); // Red
66   colorWipe(strip.Color(0, 255, 0), 500); // Green
67   colorWipe(strip.Color(0, 0, 255), 500); // Blue
68 }
69
70 void colorWipe(uint32_t color, int wait) {
71   for (int i = 0; i < strip.numPixels(); i++) {
72     strip.setPixelColor(i, color);
73     strip.show();
74     delay(wait);
75
76     // Break animation immediately if command received
77     if (SerialBT.available()) return;
78   }
79 }

```


3 LED control with MIT



```
when ListPicker7 .BeforePicking
do set ListPicker7 .Elements to BluetoothClient1 .AddressesAndNames

when ListPicker7 .AfterPicking
do if call BluetoothClient1 .Connect
    address ListPicker7 .Selection
then set ListPicker7 .Elements to BluetoothClient1 .AddressesAndNames

when Button7 .Click
do call BluetoothClient1 .Disconnect

when Clock1 .Timer
do if BluetoothClient1 .IsConnected
then set Button8 .Text to "Connected to bluetooth"
    set Button8 .BackgroundColor to green
if not BluetoothClient1 .IsConnected
then set Button8 .Text to "Connected to bluetooth"
    set Button8 .BackgroundColor to red
if BluetoothClient1 .IsConnected
then set Button10 .Text to "Connected to bluetooth"
    set Button10 .BackgroundColor to green
if not BluetoothClient1 .IsConnected
then set Button10 .Text to "Connected to bluetooth"
    set Button10 .BackgroundColor to red
if BluetoothClient1 .IsConnected
then set Button12 .Text to "Connected to bluetooth"
    set Button12 .BackgroundColor to green
if not BluetoothClient1 .IsConnected
then set Button12 .Text to "Connected to bluetooth"
    set Button12 .BackgroundColor to red
```

```
when Button9 .Click
do call BluetoothClient1 .SendText
    text 5

when Button11 .Click
do call BluetoothClient1 .SendText
    text 6

when Button13 .Click
do call BluetoothClient1 .SendText
    text 7
```



```
#include <BluetoothSerial.h>

BluetoothSerial SerialBT;

// Pins for RGB LED cathodes
const int redLED = 12;
const int greenLED = 14;
const int blueLED = 27;

// Track LED states
bool redState = false;
bool greenState = false;
bool blueState = false;

void setup() {
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(blueLED, OUTPUT);

  // Turn all LEDs OFF initially (HIGH for common anode)
  digitalWrite(redLED, HIGH);
  digitalWrite(greenLED, HIGH);
  digitalWrite(blueLED, HIGH);

  Serial.begin(115200);
  SerialBT.begin("ESP32-RGB"); // Bluetooth name
  Serial.println("Bluetooth ready. Send 1 (Red), 2 (Green), 3 (Blue) to toggle.");
}

void loop() {
  if (SerialBT.available()) {
    char incomingChar = SerialBT.read();

    if (incomingChar == '\n' || incomingChar == '\r') return;

    switch (incomingChar) {
      case '5': // Toggle Red
        redState = !redState;
        digitalWrite(redLED, redState ? LOW : HIGH); // LOW = ON
        SerialBT.println(redState ? "Red ON" : "Red OFF");
        break;

      case '6': // Toggle Green
        greenState = !greenState;
        digitalWrite(greenLED, greenState ? LOW : HIGH);
        SerialBT.println(greenState ? "Green ON" : "Green OFF");
        break;

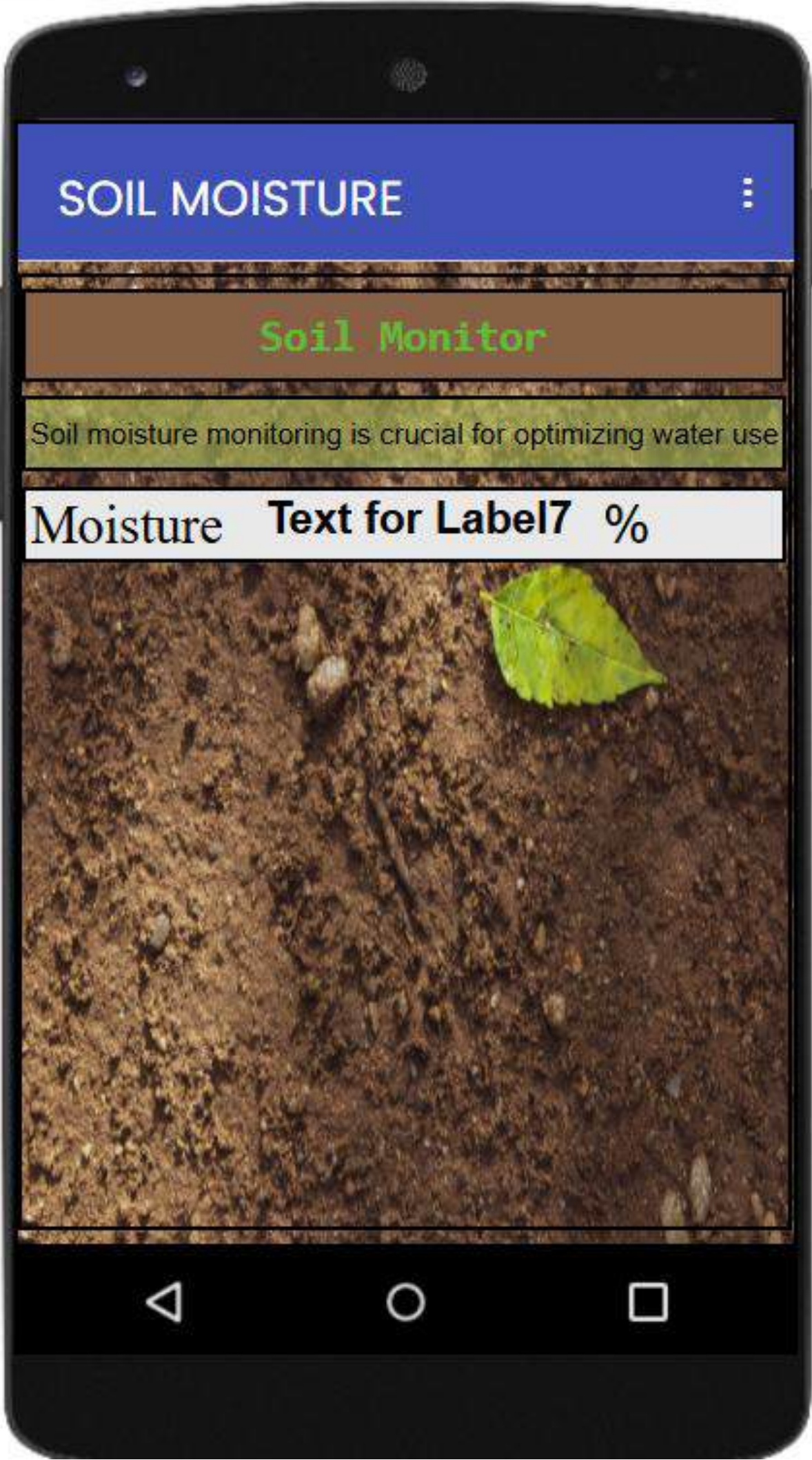
      case '7': // Toggle Blue
        blueState = !blueState;
        digitalWrite(blueLED, blueState ? LOW : HIGH);
        SerialBT.println(blueState ? "Blue ON" : "Blue OFF");
        break;

      default:
        SerialBT.println("Send 1=Red, 2=Green, 3=Blue");
        break;
    }
  }
}
```

Our Learnings:

- Learned how to control NeoPixel LEDs with an ESP32 using Bluetooth.
- Understood how to send color commands from a phone and change LEDs instantly.
- Implemented a timer to return to animation mode when no commands are received.
- Discovered that breaking the code into smaller functions makes it easier to read and modify.

Soil Moisture



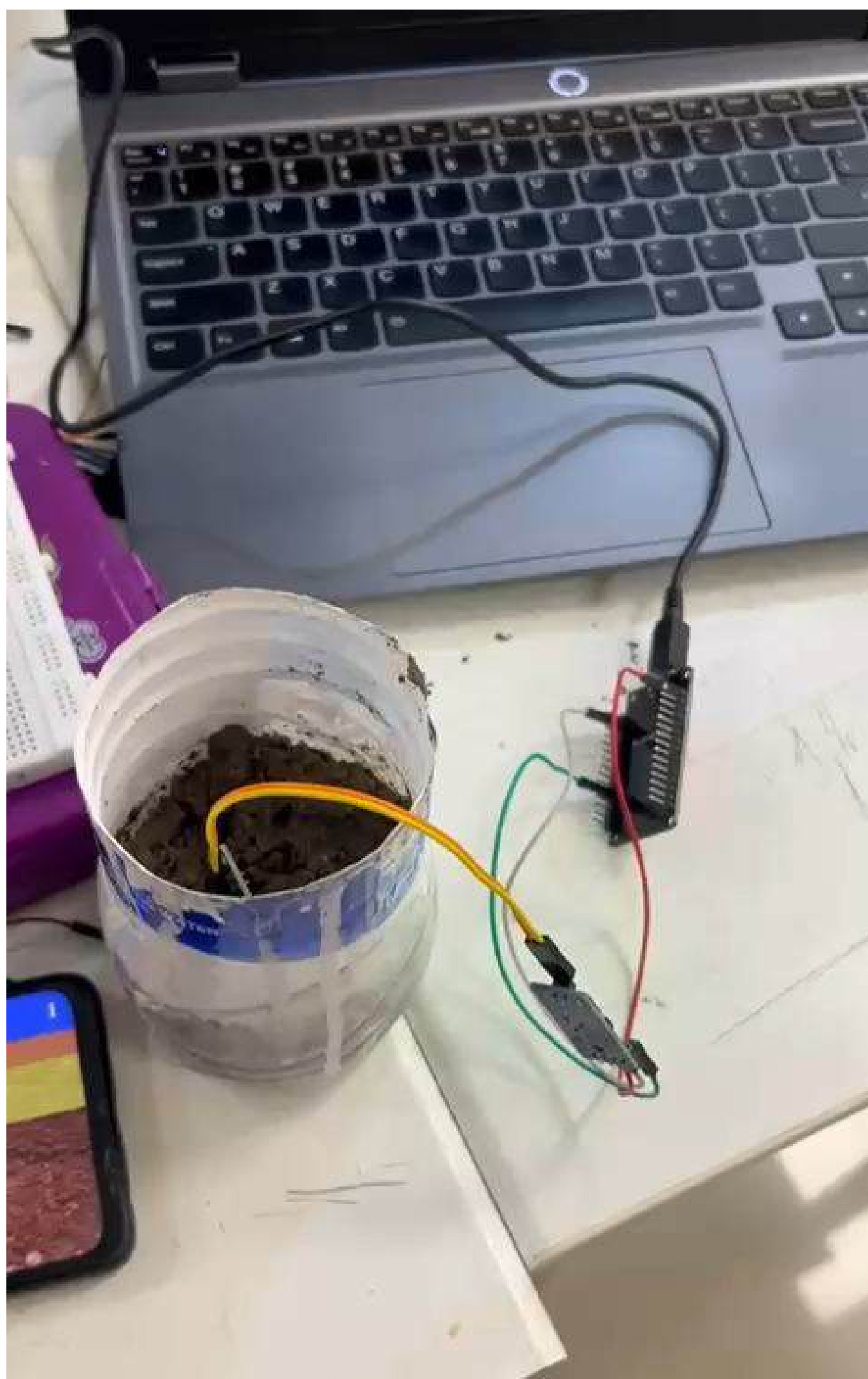
initialize global value to 0

initialize global link to "https://api.thingspeak.com/channels/3028215/feed..."

```
when Web1 GotText
  url responseCode responseType responseContent
  do
    if
      then
        initialize local json to call Web1 JsonTextDecode
        in
          set global value to look up in pairs key "field1" pairs get json notFound "not found"
          get responseContent
```

```
when Clock1 Timer
  do
    call dht
```

```
to dht
  do
    set Web1 Url to get global link
    call Web1 Get
```

Our Learnings

- Understood how to interface an analog soil moisture sensor with ESP32.
- Learned to map raw ADC values (0–4095) into percentage for better readability.
- Implemented Wi-Fi connection and reconnection logic to maintain IoT data flow.
- Sent sensor readings to ThingSpeak for cloud-based monitoring.
- Practiced loop timing control to follow platform update rules and reduce network load.

```
#include <WiFi.h>
#include "ThingSpeak.h" // always include thingspeak header file after other

// Update with your new WiFi credentials
char ssid[] = "mayank"; // your WiFi SSID
char pass[] = "mayank123"; // your WiFi password

WiFiClient client;

// Updated ThingSpeak details
unsigned long myChannelNumber = 3028215; // Your new channel ID
const char * myWriteAPIKey = "EGBHF8TTGWPCRVT6"; // Replace with your new

// Analog pin connected to soil moisture sensor
#define SOIL_PIN 34

void setup()
{
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client); // Initialize ThingSpeak
}

void loop()
{
  // Read soil moisture value (0–4095)
  int soilRaw = analogRead(SOIL_PIN);

  // Convert to percentage (optional scaling)
  int moisturePercent = map(soilRaw, 4095, 0, 0, 100);

  // Reconnect to WiFi if disconnected
  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.print("Connecting to WiFi...");
    while (WiFi.status() != WL_CONNECTED)
    {
      WiFi.begin(ssid, pass);
      delay(5000);
      Serial.print(".");
    }
    Serial.println("Connected!");
  }

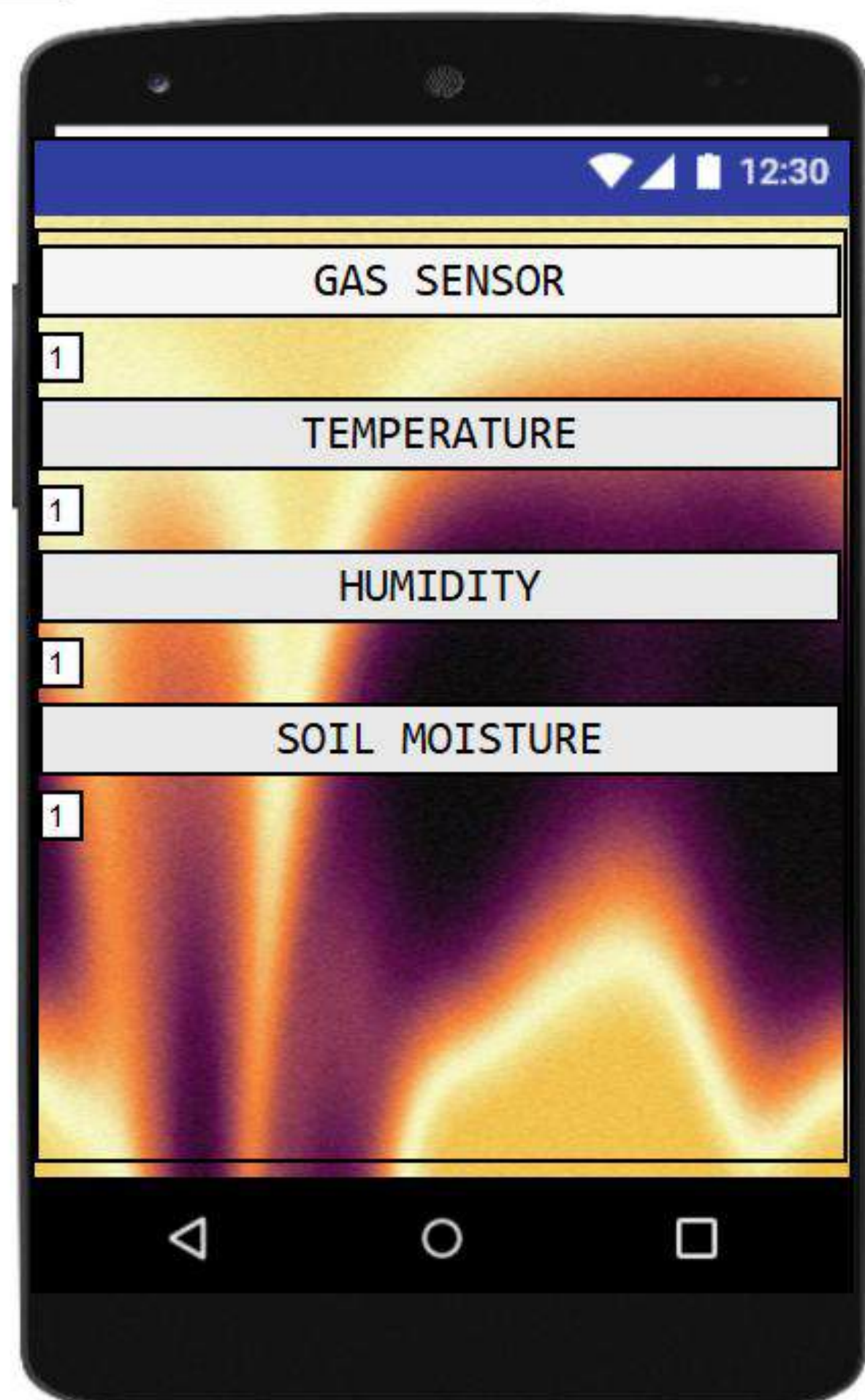
  // Update to ThingSpeak
  ThingSpeak.setField(1, moisturePercent);
  int status = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

  if (status == 200) {
    Serial.print("Soil Moisture: ");
    Serial.print(moisturePercent);
    Serial.println("% sent to ThingSpeak.");
  } else {
    Serial.print("Error sending data: ");
    Serial.println(status);
  }

  delay(1000); // Wait 15 seconds before sending next update
}
```


DHT, Soil moisture and Gas sensors

Phone (320 x 505) Android 5+ (Material) ☐ Display hidden components in Viewer

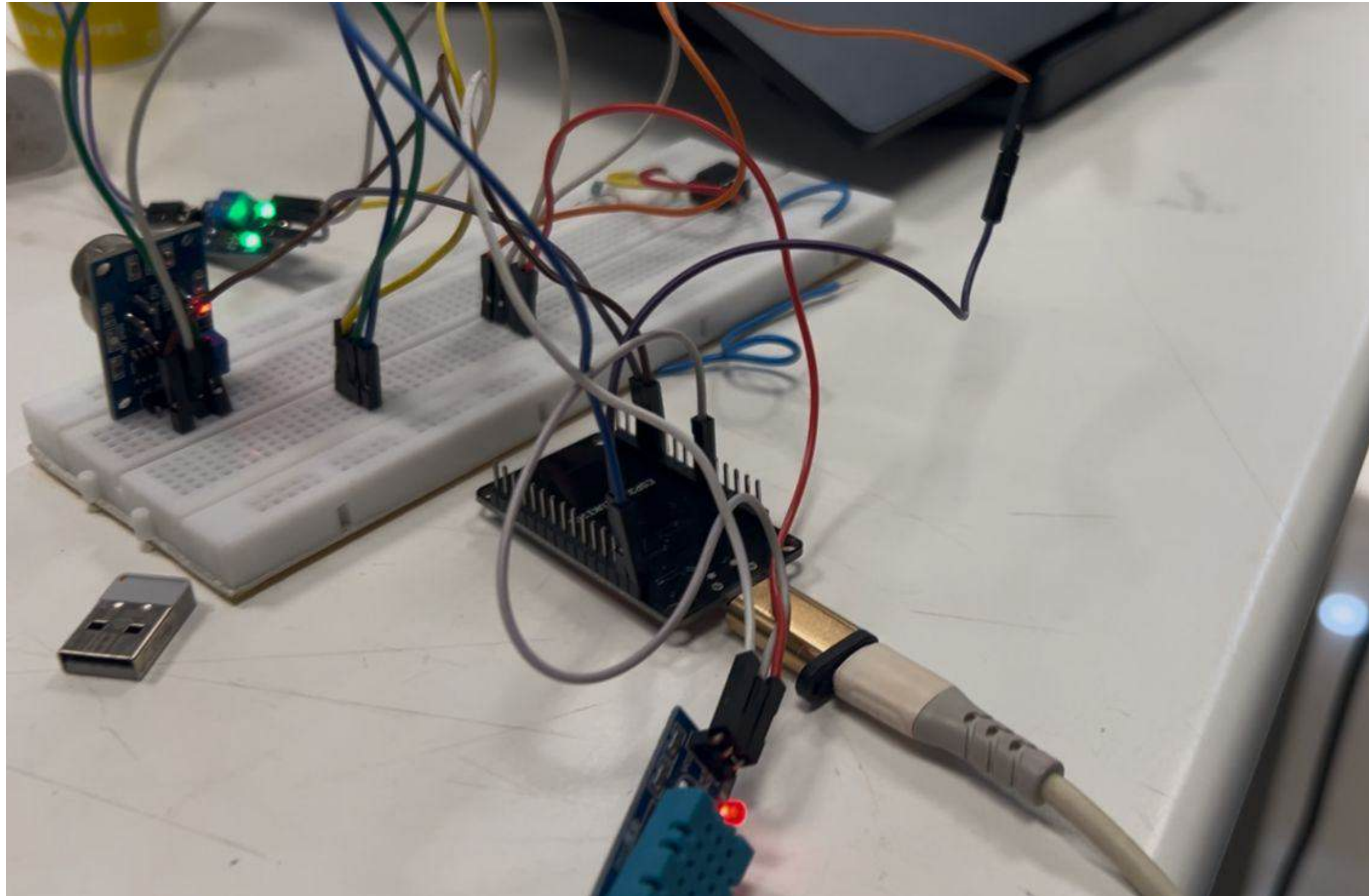


Non-visible components

```
initialize global value to 0
initialize global value1 to 0
initialize global value2 to 0
initialize global value3 to 0
```

```
initialize global link to https://api.thingspeak.com/channels/3025802/feed...
```

```
when Web1 GotText
  url responseCode responseType responseContent
  do
    if get responseCode == 200
      then
        initialize local json to call Web1 JsonTextDecode
        jsonText get responseContent
        in
          set global value to look up in pairs key field1
          pairs get json
          notFound not found
          set global value1 to look up in pairs key field2
          pairs get json
          notFound not found
          set global value2 to look up in pairs key field3
          pairs get json
          notFound not found
          set global value3 to look up in pairs key field4
          pairs get json
          notFound not found
        to dht
        do
          set Web1 Uri to get global link
          call Web1 Get
        when Clock1 Timer
        do
          set Label14 Text to get global value
          set Label15 Text to get global value
          set Label16 Text to get global value
          set Label17 Text to get global value
```

Our Learnings

- Successfully interfaced multiple sensors (DHT11, soil moisture, MQ gas sensor) with ESP32.
- Implemented Wi-Fi connectivity for real-time IoT data transfer.
- Used ThingSpeak API to upload sensor readings for remote monitoring.
- Learned to map and calibrate sensor data into meaningful percentage values.
- Implemented automatic reconnection logic to handle Wi-Fi drops.
- Understood the importance of data update intervals to comply with IoT platform limits.

```
#include <WiFi.h>
#include "ThingSpeak.h"
#include "DHT.h"

// WiFi settings
char ssid[] = "mayank";
char pass[] = "mayank123";
WiFiClient client;

// ThingSpeak channel details
unsigned long myChannelNumber = 3028215;
const char * myWriteAPIKey = "YOUR_API_KEY_HERE"; // Replace with actual key

// Pin definitions
#define DHTPIN 14 // DHT data pin
#define DHTTYPE DHT11 // or DHT22
#define SOIL_PIN 34 // Analog pin for soil moisture sensor
#define GAS_PIN 35 // Analog pin for MQ gas sensor

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client); // Initialize ThingSpeak
}

void loop() {
  // Read DHT sensor
  float temp = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Read soil moisture (0-4095)
  int soilRaw = analogRead(SOIL_PIN);
  int soilPercent = map(soilRaw, 4095, 0, 0, 100); // Calibrate as needed

  // Read gas sensor (0-4095)
  int gasValue = analogRead(GAS_PIN); // Higher = more gas detected

  // Reconnect to WiFi if disconnected
  if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
      WiFi.begin(ssid, pass);
      delay(3000);
      Serial.print(".");
    }
    Serial.println("Connected!");
  }

  // Print to Serial
  Serial.println("---- Sensor Readings ----");
  Serial.print("Temperature: "); Serial.print(temp); Serial.println(" C");
  Serial.print("Humidity: "); Serial.print(humidity); Serial.println(" %");
  Serial.print("Soil Moisture: "); Serial.print(soilPercent); Serial.println(" %");
  Serial.print("Gas Sensor (Raw): "); Serial.println(gasValue);

  // Send to ThingSpeak
  ThingSpeak.setField(1, temp);
  ThingSpeak.setField(2, humidity);
  ThingSpeak.setField(3, soilPercent);
  ThingSpeak.setField(4, gasValue);

  int status = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  if (status == 200) {
    Serial.println("Data sent to ThingSpeak.");
  } else {
    Serial.print("ThingSpeak Error: "); Serial.println(status);
  }

  delay(15000); // ThingSpeak free tier allows updates every 15 seconds
}
```


DHT Sensor



```
initialize global Value to 0
```

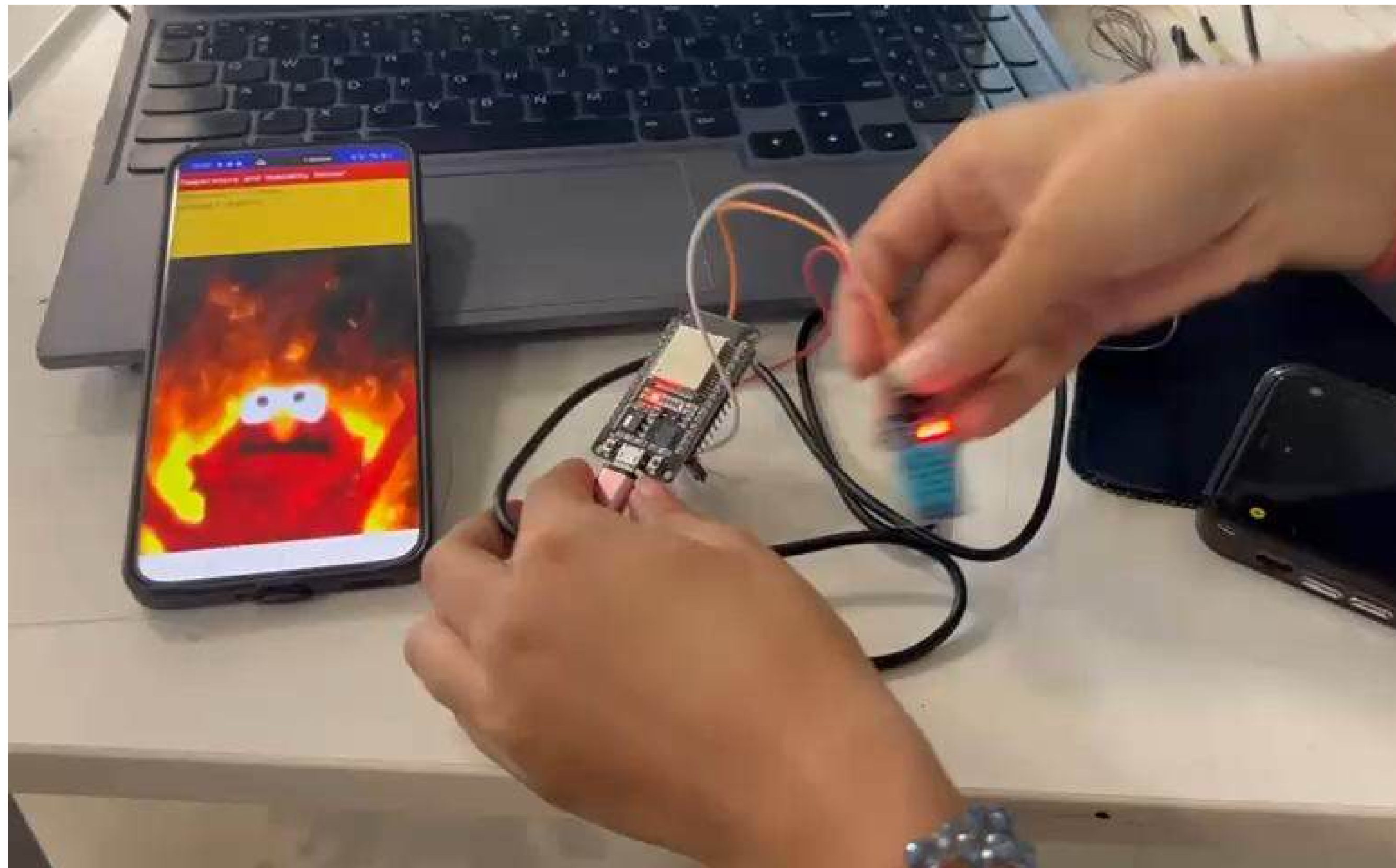
```
initialize global Value2 to 0
```

```
initialize global link to "https://api.thingspeak.com/channels/3025889/feed..."
```

```
when Web1 GotText
  url responseCode responseType responseContent
  do
    if
      get responseCode = 200
    then
      initialize local json to call Web1 .JsonTextDecode
      in
        set global Value to look up in pairs key "field1"
        pairs get json
        notFound "not found"
        set global Value2 to look up in pairs key "field2"
        pairs get json
        notFound "not found"
```

```
when Clock1 Timer
  do
    call dht
    set Label7 Text to get global Value
    set Label5 Text to get global Value2
```

```
to dht
  do
    set Web1 Url to get global link
    call Web1 .Get
```



Our Learnings

- Learned how to interface a DHT11 temperature and humidity sensor with the ESP32.
- Used the ThingSpeak API to upload multiple sensor fields (temperature and humidity).
- Implemented Wi-Fi reconnection logic to ensure continuous data transmission.
- Practiced working with sensor libraries (DHT.h) for simplified reading.
- Understood the importance of delays in IoT data uploads to meet API limits.

```
#include <WiFi.h>
#include "ThingSpeak.h" // always include thingspeak header after other headers
#include <DHT.h>

char ssid[] = "mayank"; // your network SSID (name)
char pass[] = "mayank123"; // your network password
int keyIndex = 0; // only needed for WEP

WiFiClient client;

unsigned long myChannelNumber = 3025889;
const char * myWriteAPIKey = "408KVQI11N0HYKS3";

#define DHTPIN 14 // GPIO14 (change if using a different pin)
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  while (!Serial) { ; } // wait for serial port (Leonardo only)

  dht.begin();
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client); // initialize ThingSpeak
}

void loop() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Connect or reconnect to WiFi
  if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    while (WiFi.status() != WL_CONNECTED) {
      WiFi.begin(ssid, pass);
      Serial.print(".");
      delay(5000);
    }
    Serial.println("\nConnected.");
  }

  ThingSpeak.setField(1, t);
  ThingSpeak.setField(2, h);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

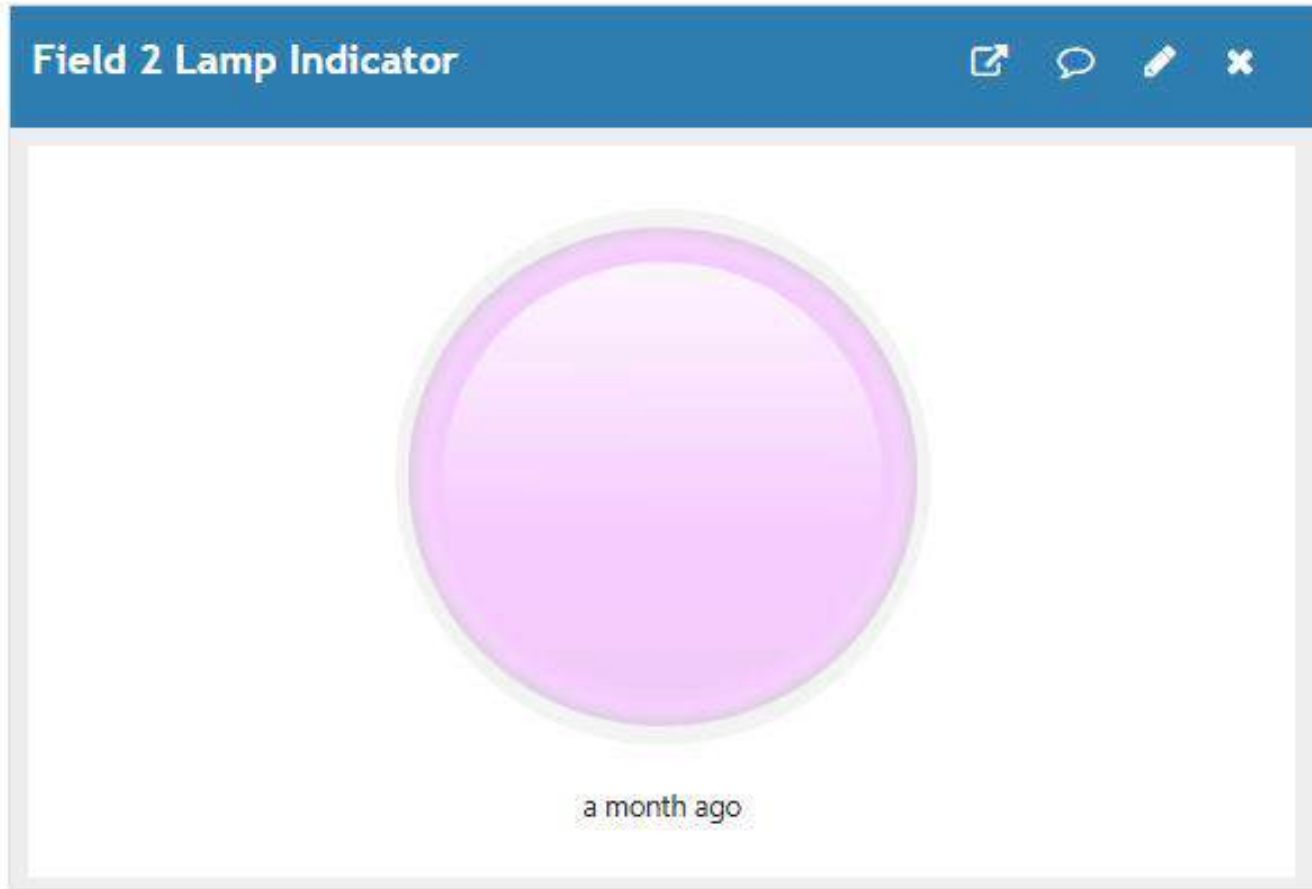
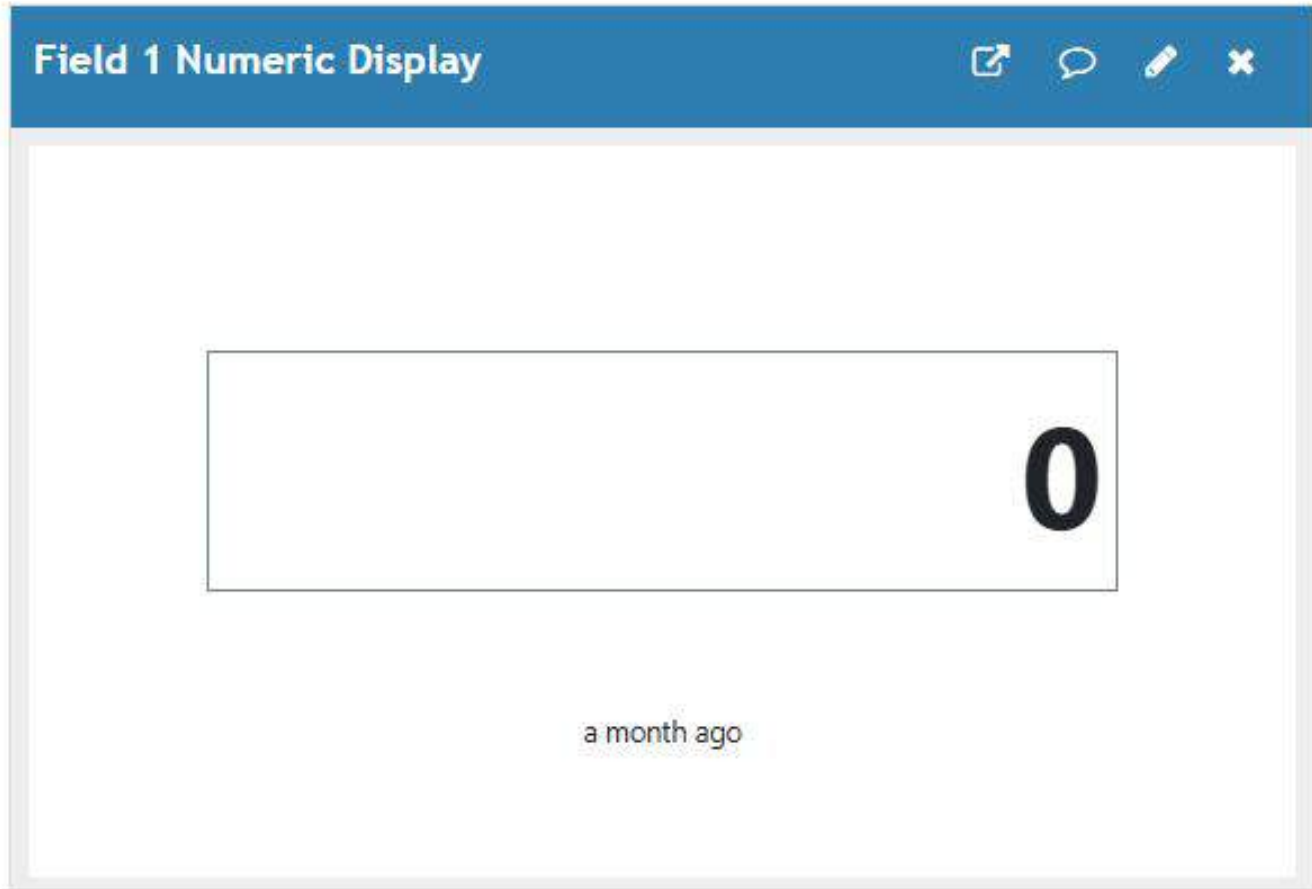
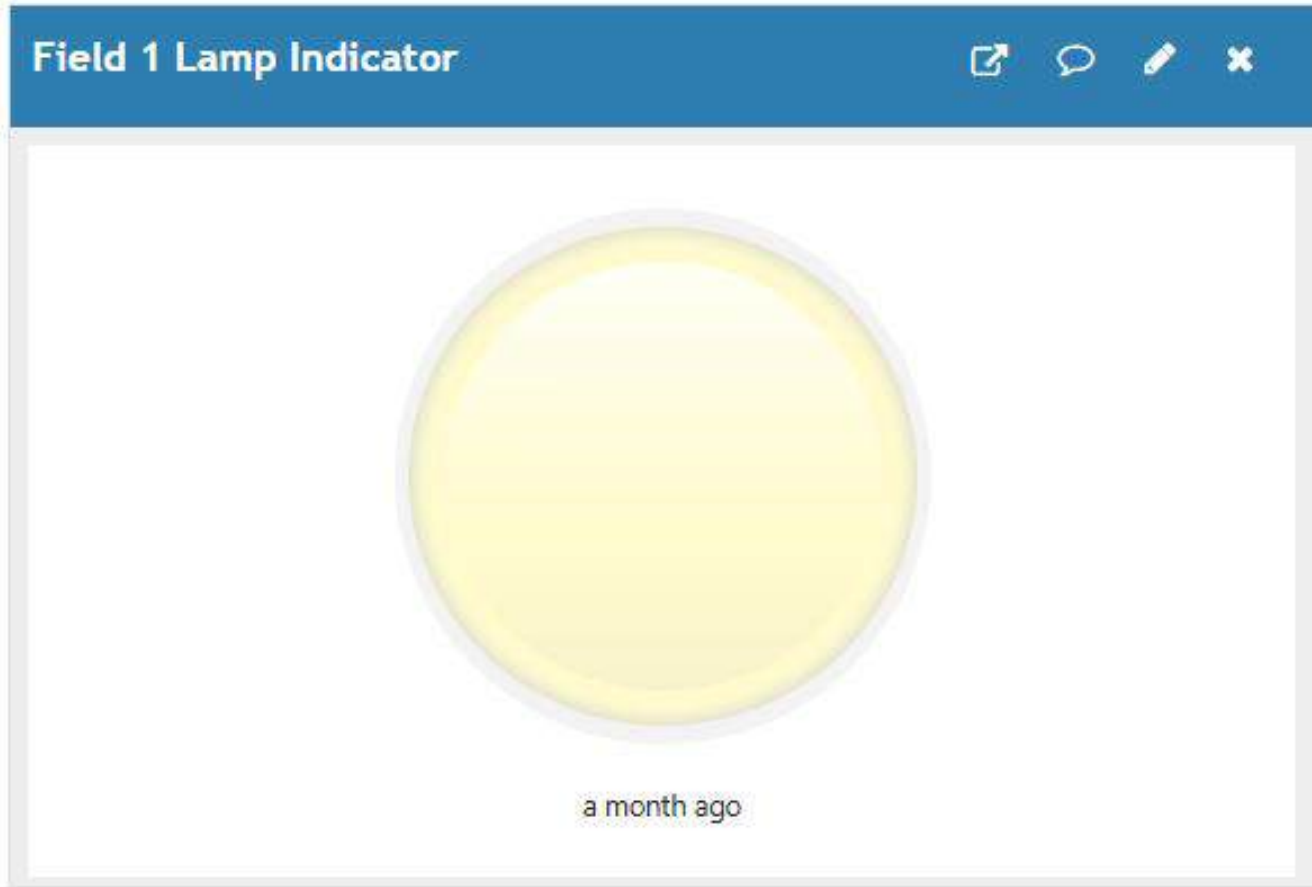
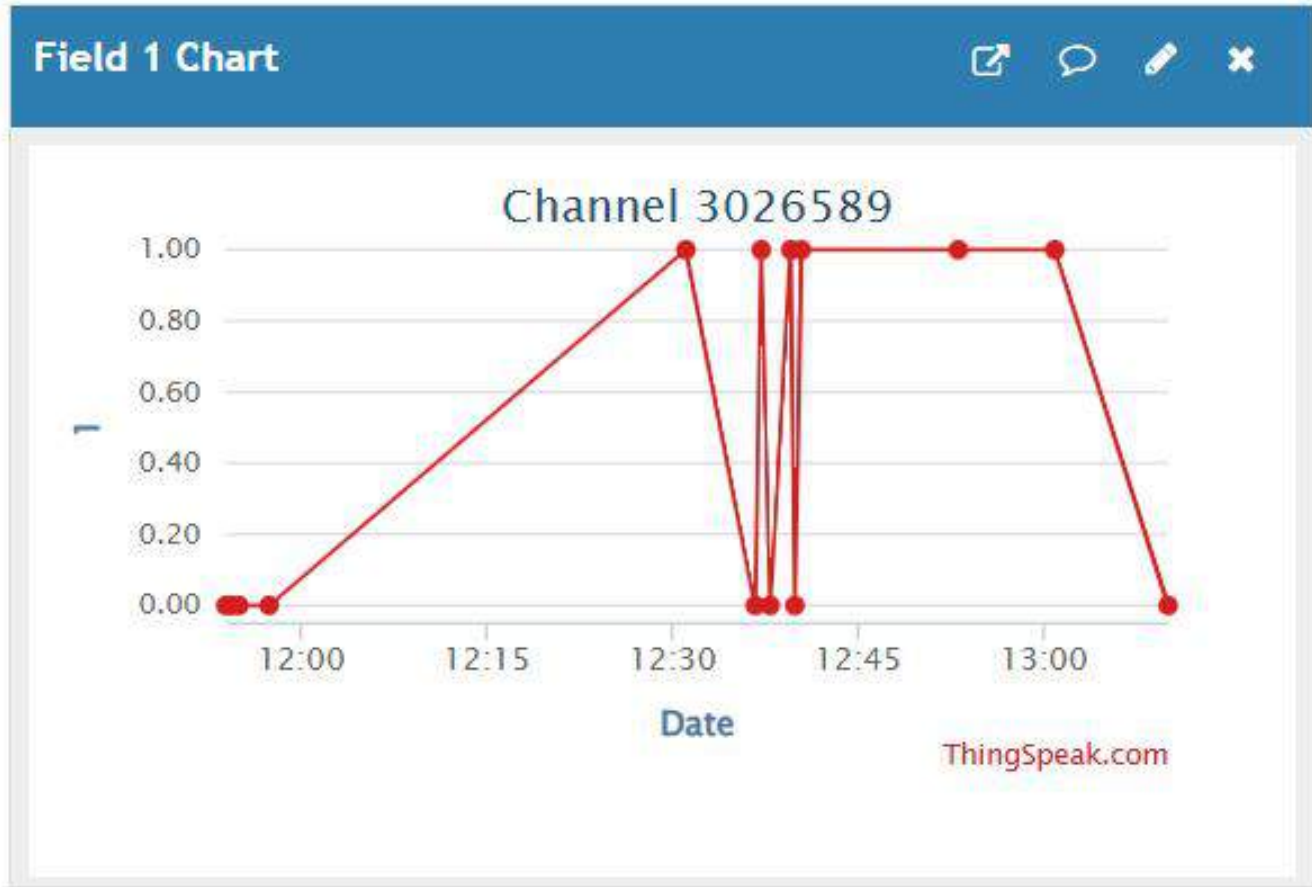
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print("°C, Humidity: ");
  Serial.print(h);
  Serial.println("%. Sent to ThingSpeak.");

  delay(15000); // wait 15 seconds before updating again
}
```




Thingspeak webserver

LED, ESP-32 and bluetooth app



```
V1
writeSingleField.ino secrets.h
5 #error Bluetooth is not enabled! Please run make menuconfig to enable it
6 #endif
7
8 // Create BluetoothSerial object
9 BluetoothSerial SerialBT;
10
11 // Define LED pin
12 const int ledPin = 12; // GPIO4
13 void setup() {
14     // Initialize LED pin as output
15     pinMode(ledPin, OUTPUT);
16     digitalWrite(ledPin, LOW); // LED off initially
17
18     // Initialize Serial for debugging
19     Serial.begin(115200);
20
21     // Initialize Bluetooth with device name
22     SerialBT.begin("ESP32-BT-tc"); // Bluetooth device name
23     Serial.println("Bluetooth Started! Pair with ESP32-BT-tc");
24 }
25
26 void loop() {
27     // Check if data is available from Bluetooth
28     if (SerialBT.available()) {
29         char incomingChar = SerialBT.read();
30
31         // Ignore newline (\n) and carriage return (\r)
32         if (incomingChar == '\n' || incomingChar == '\r') {
33             return; // Skip processing these characters
34         }
35     }
36 }
37
38 // Write data to Field 1
39 void writeField1(int value) {
40     SerialBT.write(value);
41 }
42
43 // Write data to Field 2
44 void writeField2(int value) {
45     SerialBT.write(value);
46 }
47
48 // Main loop
49 void loop() {
50     // Read data from Field 1
51     int value1 = SerialBT.read();
52
53     // Read data from Field 2
54     int value2 = SerialBT.read();
55
56     // Write data to Field 1
57     writeField1(value1);
58
59     // Write data to Field 2
60     writeField2(value2);
61
62     // Toggle LED
63     digitalWrite(ledPin, !digitalRead(ledPin));
64
65     delay(1000);
66 }
```

Output Serial Monitor gdb-server X

Waiting for gdb server to start...[2025-08-04T11:15:48.371Z] SERVER CONSOLE DE
:\\Users\\adwai\\AppData\\Local\\Arduino15\\packages\\esp32\\tools\\openocd-
ai\\AppData\\Local\\Temp\\.arduinoIDE-unsaved202574-23508-2myq9c.76ibh\\Writ
nood-helpers.tcl" -f board/esp32-wrover-kit-3.3v.cfg

ThingSpeak dashboard created for visualization
Time-series graph shows LED state over time
Circular indicator displays current LED status

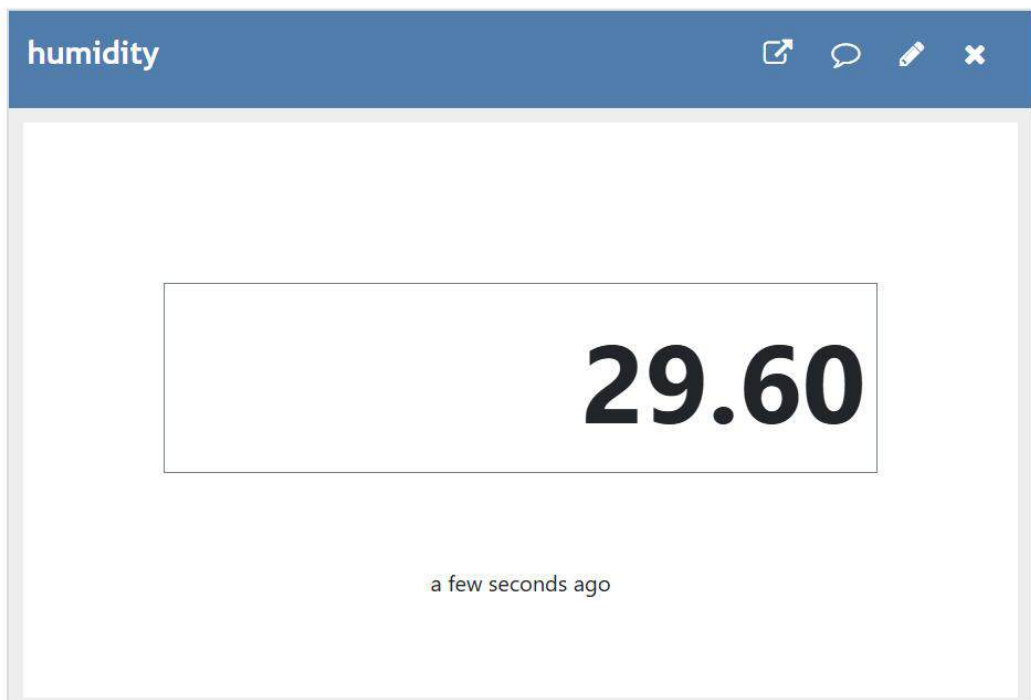
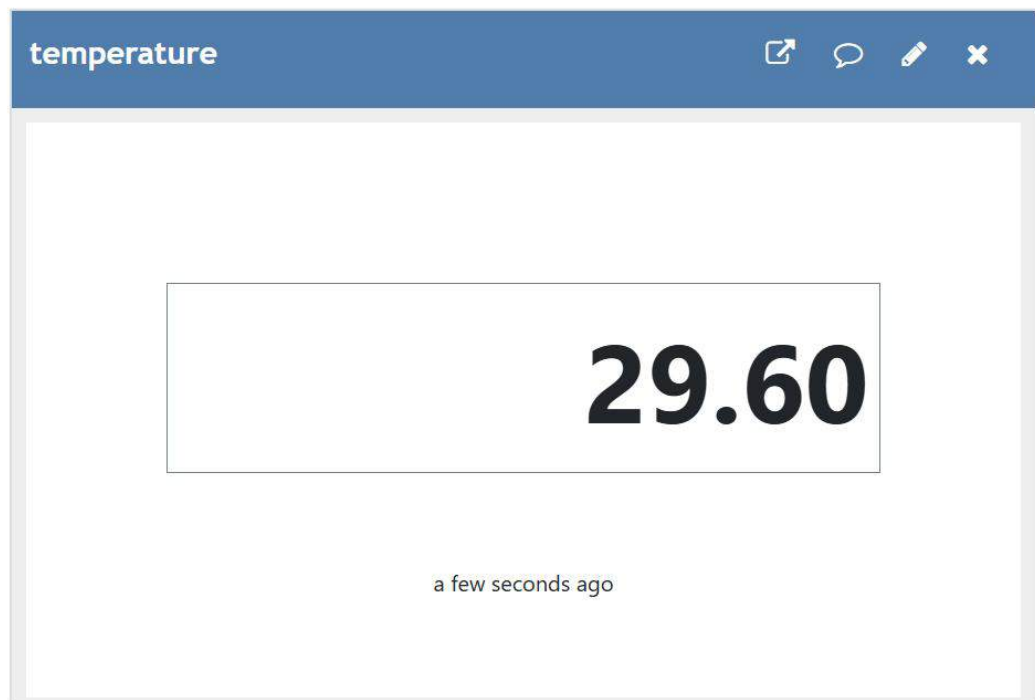
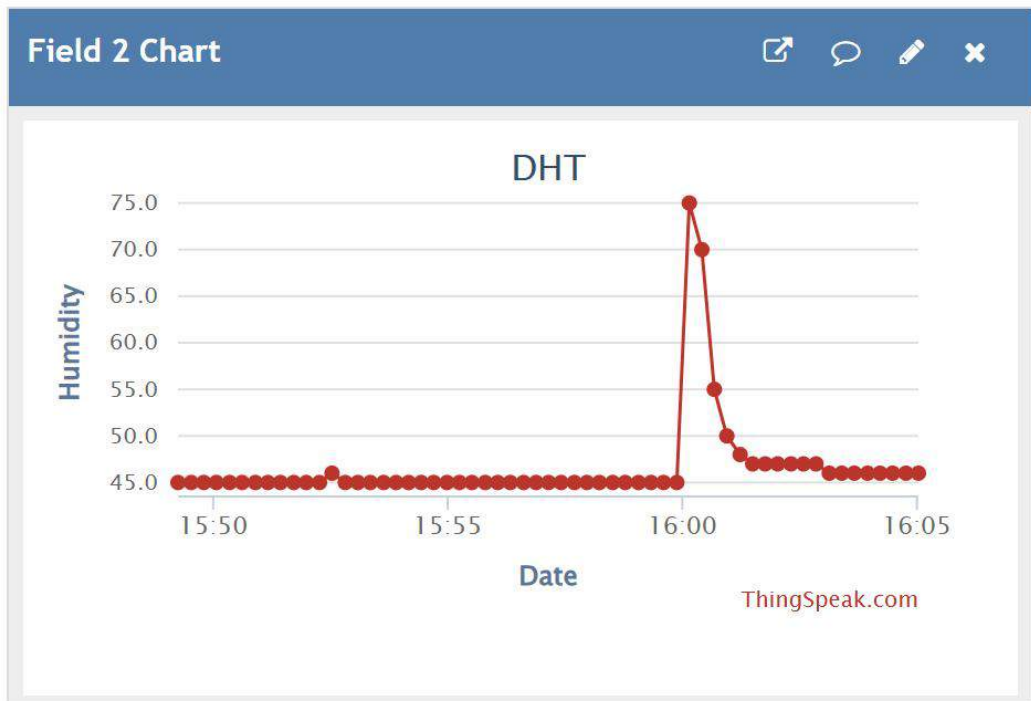
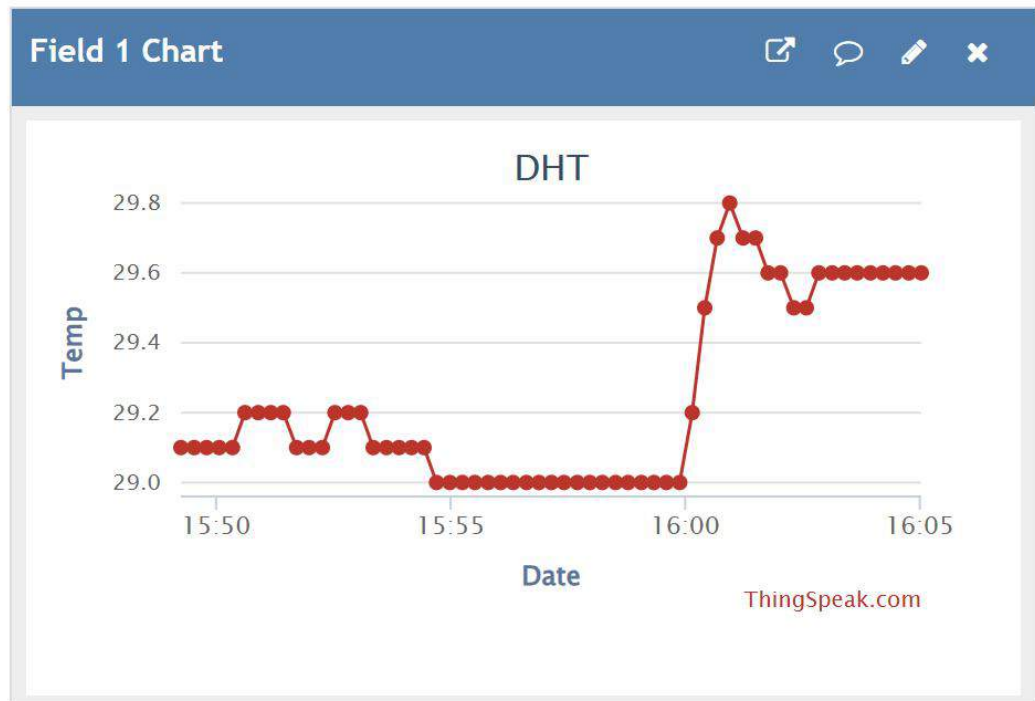
Output

Our Learnings:

1. Understood how to control an LED via Bluetooth using ESP32.
2. Learned to send LED status data from ESP32 to ThingSpeak.
3. Explored ThingSpeak visualizations:
 - Time-series chart for LED state history.
 - Lamp indicator for real-time ON/OFF status.
 - Numeric display for binary LED values (0/1).
4. Understood the importance of API keys and channel fields in ThingSpeak.
5. Observed how Bluetooth app control + IoT cloud dashboard integrate into one system.
6. Gained hands-on experience in building a basic IoT application with feedback and monitoring.



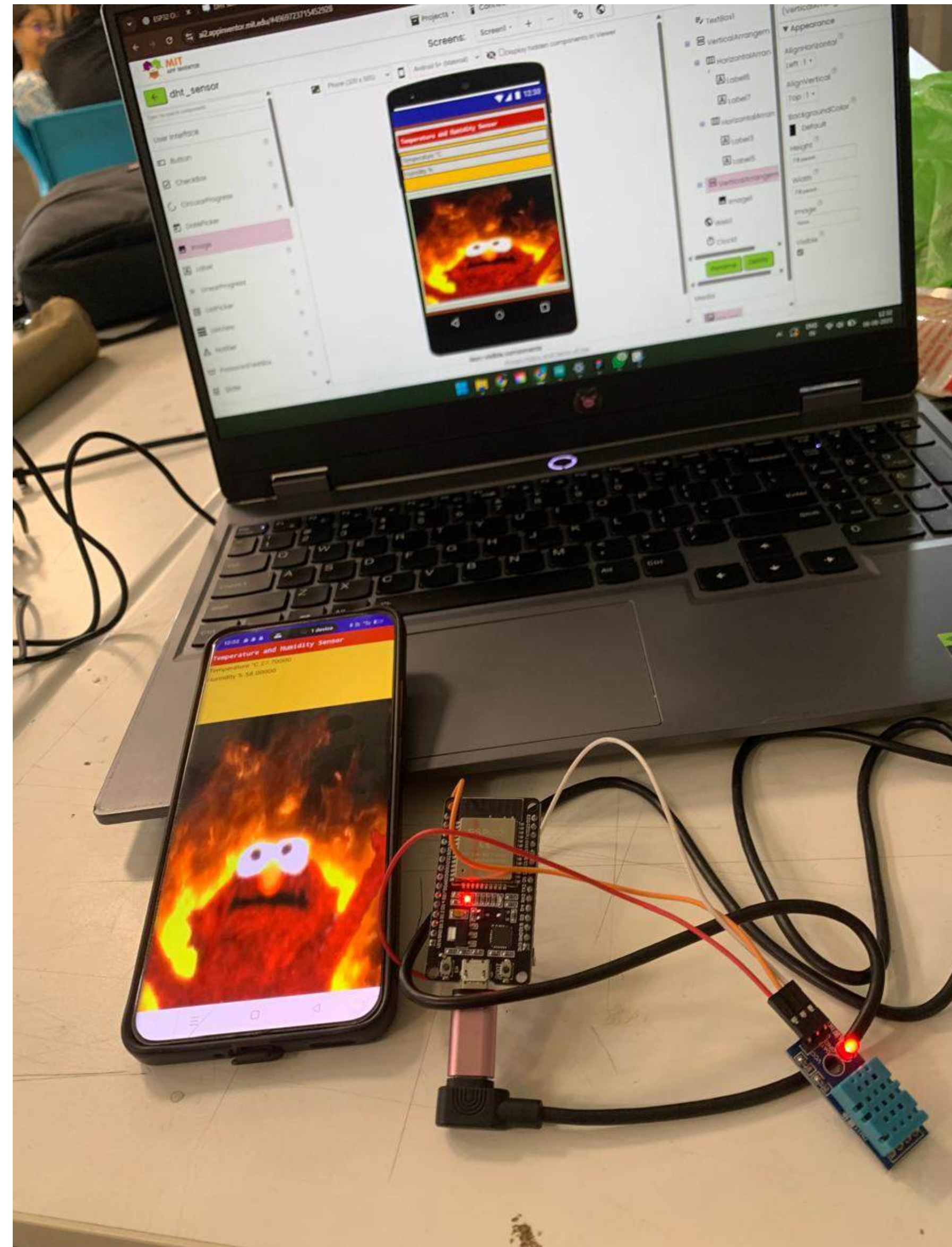
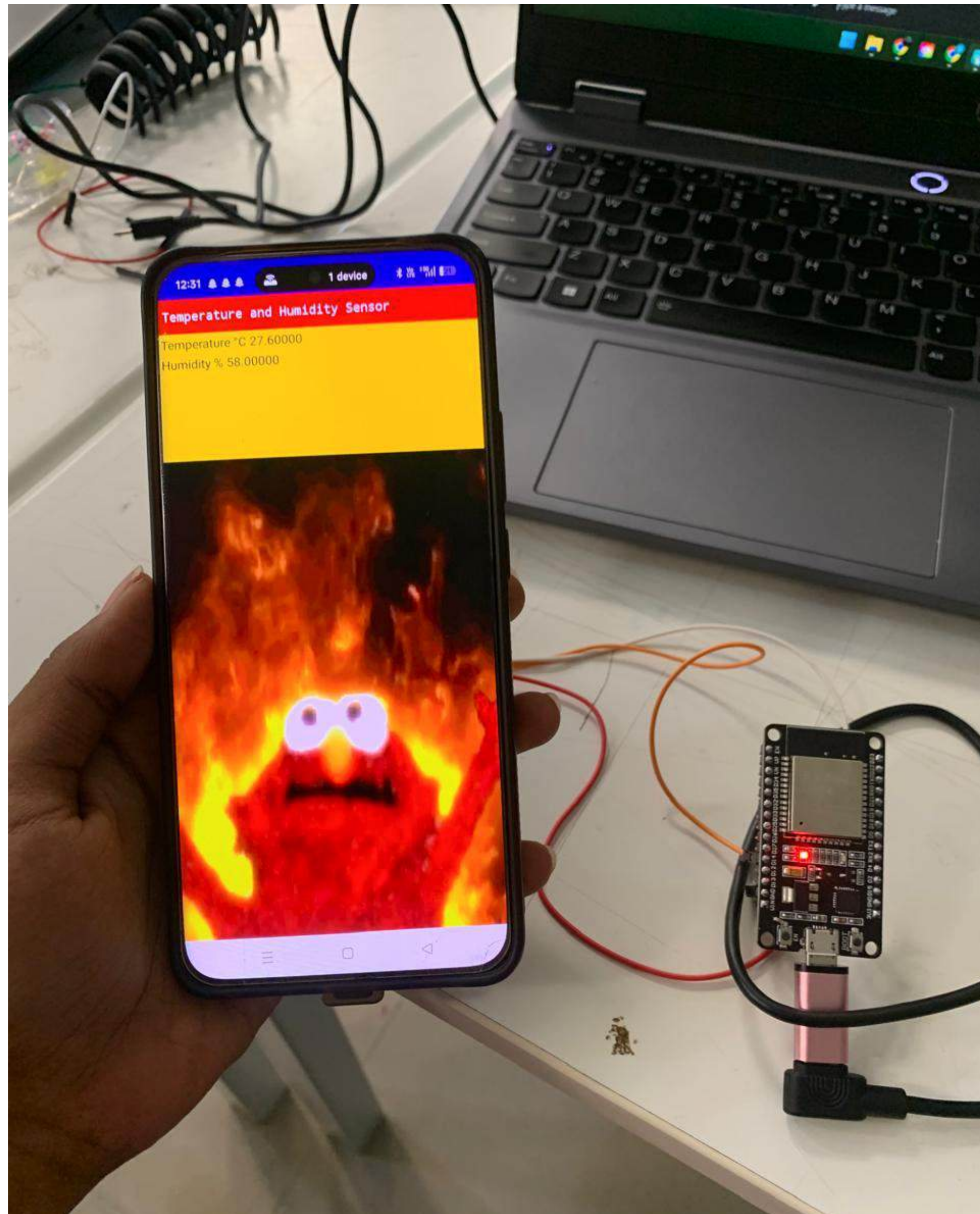
DHT sensor (thingspeak)



1. ThingSpeak requires an API key and a simple HTTP request to upload data.
2. ThingSpeak provides ready-made graphs, dashboards, and basic analytics that simplify monitoring.
3. ESP32 connects to WiFi easily, but needs stable power for reliable operation.
4. Discovered that ThingSpeak can run basic analytics with MATLAB, which could help in future projects for alerts or automation.
5. The DHT sensor needs a minimum 2-second delay between readings.

```
WriteSingleField | Arduino IDE 2.3.6
File Edit Sketch Tools Help
ψ DOIT ESP32 DEVKIT V1
WriteSingleField.ino secrets.h
1 #include <WiFi.h>
2 #include "ThingSpeak.h" // always include thingspeak header file after other header files and custom macros
3 #include <DHT.h>
4 char ssid[] = "mayank"; // your network SSID (name)
5 char pass[] = "mayank123"; // your network password
6 int keyIndex = 0; // your network key Index number (needed only for WEP)
7 WiFiClient client;
8
9 unsigned long myChannelNumber = 3025901;
10 const char * myWriteAPIKey = "KFDZTC9REL9E2R0U";
11 #define DHTPIN 14 // GPIO14 (Change this if using a different pin)
12 #define DHTTYPE DHT11
13 DHT dht(DHTPIN, DHTTYPE);
14
15 void setup()
16 {
17   Serial.begin(115200); //Initialize serial
18   while (!Serial)
19   {
20     ; // wait for serial port to connect. Needed for Leonardo native USB port only
21   }
22
23   dht.begin();
24   WiFi.mode(WIFI_STA);
25   ThingSpeak.begin(client); // Initialize ThingSpeak
26 }
27 void loop()
28 {
29   float h = dht.readHumidity();
30   float t = dht.readTemperature();
31
32   // Connect or reconnect to WiFi
33   if(WiFi.status() != WL_CONNECTED)
34   {
35     Serial.print("Attempting to connect to SSID: ");
36
37     while(WiFi.status() != WL_CONNECTED)
38     {
39       WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this line if using open or WEP network
40       Serial.print(".");
41       delay(5000);
42     }
43     Serial.println("\nConnected.");
44   }
45   ThingSpeak.setField(1, t);
46   ThingSpeak.setField(2, h);
47   ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
48   Serial.print("Temperature: ");
49   Serial.print(t);
50   Serial.print(" degrees Celcius, Humidity: ");
51   Serial.print(h);
52   Serial.println("%. Send to Thingspeak.");
53
54   delay(15000); // Wait 15 seconds to update the channel again
55 }
56
Output Serial Monitor X
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM9')
Temperature: 29.70 degrees Celcius, Humidity: 46.00%. Send to Thingspeak.
Temperature: 29.80 degrees Celcius, Humidity: 46.00%. Send to Thingspeak.
Temperature: 29.80 degrees Celcius, Humidity: 46.00%. Send to Thingspeak.
Temperature: 29.70 degrees Celcius, Humidity: 46.00%. Send to Thingspeak.
```

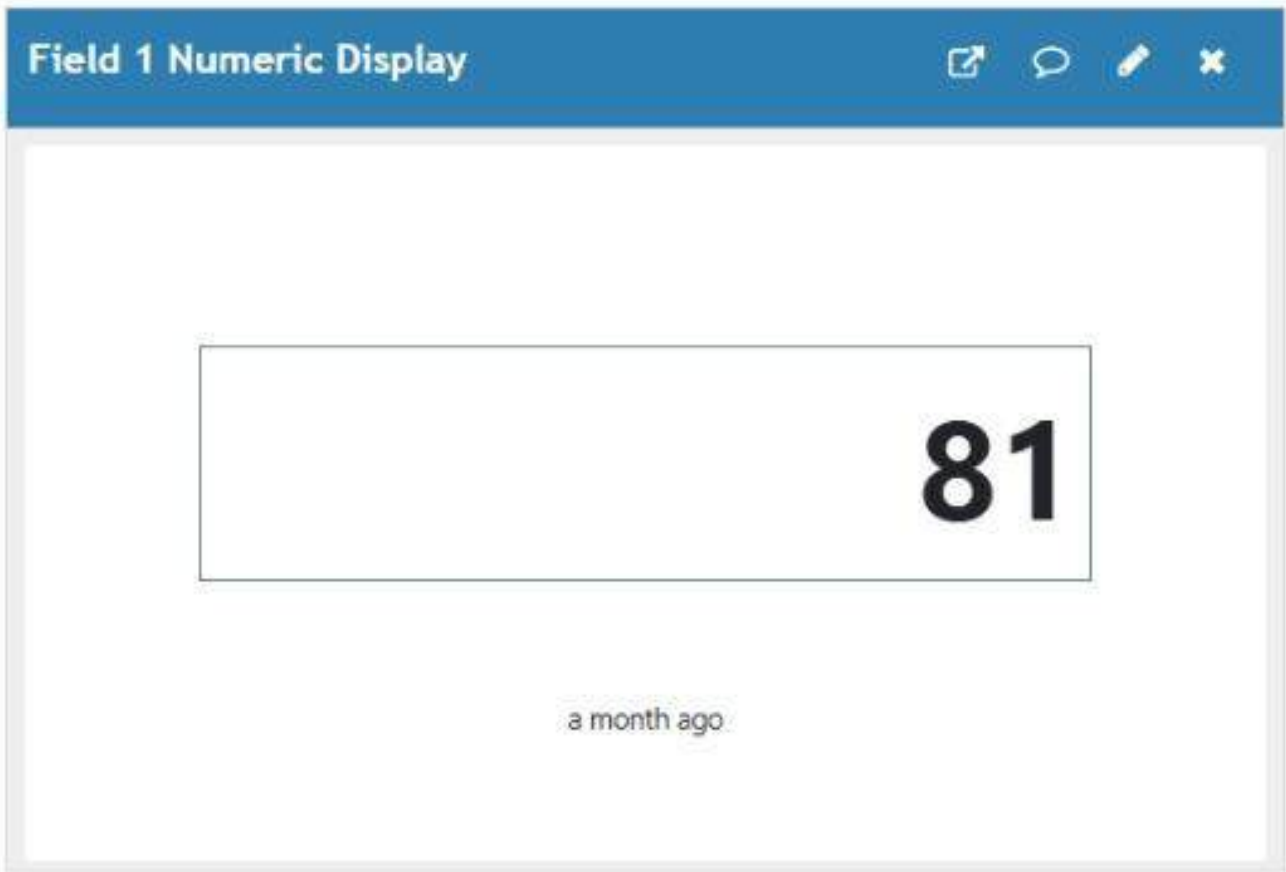

Output



Soil Moisture sensor

Channel Stats

Created: 26 days ago
Last entry: 26 days ago
Entries: 474



ThingSpeak dashboard created for visualization
Time-series graph shows LED state over time
Circular indicator displays current LED status

```
#include <WiFi.h>
#include "ThingSpeak.h" // always include thingspeak header file after other headers

// Update with your new WiFi credentials
char ssid[] = "mayank"; // your WiFi SSID
char pass[] = "mayank123"; // your WiFi password

WiFiClient client;

// Updated ThingSpeak details
unsigned long myChannelNumber = 3028215; // Your new channel ID
const char * myWriteAPIKey = "EGBHF8TTGWPCRV6"; // Replace with your new API key

// Analog pin connected to soil moisture sensor
#define SOIL_PIN 34

void setup()
{
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client); // Initialize ThingSpeak
}

void loop()
{
  // Read soil moisture value (0-4095)
  int soilRaw = analogRead(SOIL_PIN);

  // Convert to percentage (optional scaling)
  int moisturePercent = map(soilRaw, 4095, 0, 0, 100);

  // Reconnect to WiFi if disconnected
  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.print("Connecting to WiFi...");
    while (WiFi.status() != WL_CONNECTED)
    {
      WiFi.begin(ssid, pass);
      delay(5000);
      Serial.print(".");
    }
    Serial.println("Connected!");
  }

  // Update to ThingSpeak
  ThingSpeak.setField(1, moisturePercent);
  int status = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

  if (status == 200) {
    Serial.print("Soil Moisture: ");
    Serial.print(moisturePercent);
    Serial.println("% sent to ThingSpeak.");
  } else {
    Serial.print("Error sending data: ");
    Serial.println(status);
  }

  delay(1000); // Wait 15 seconds before sending next update
}
```

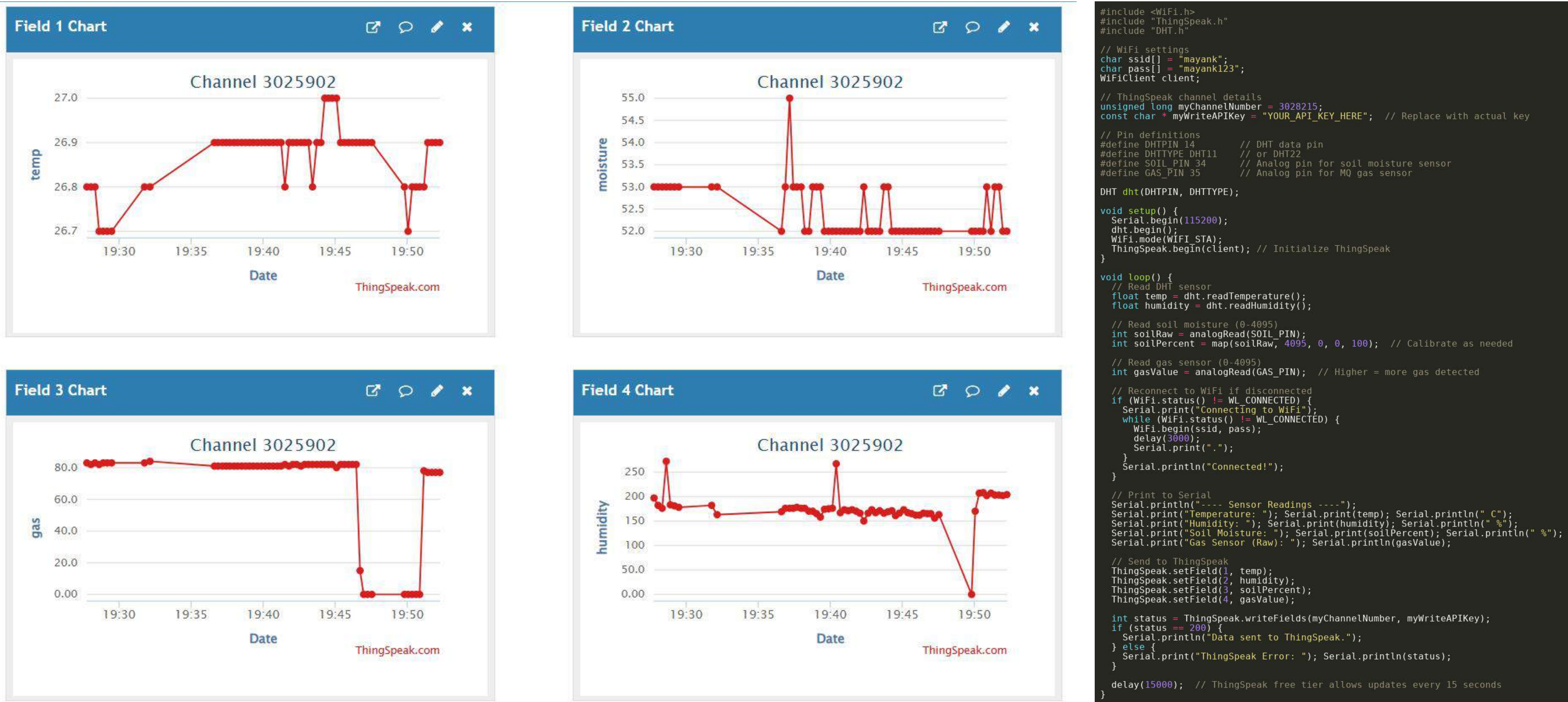

Output

Our Learnings:

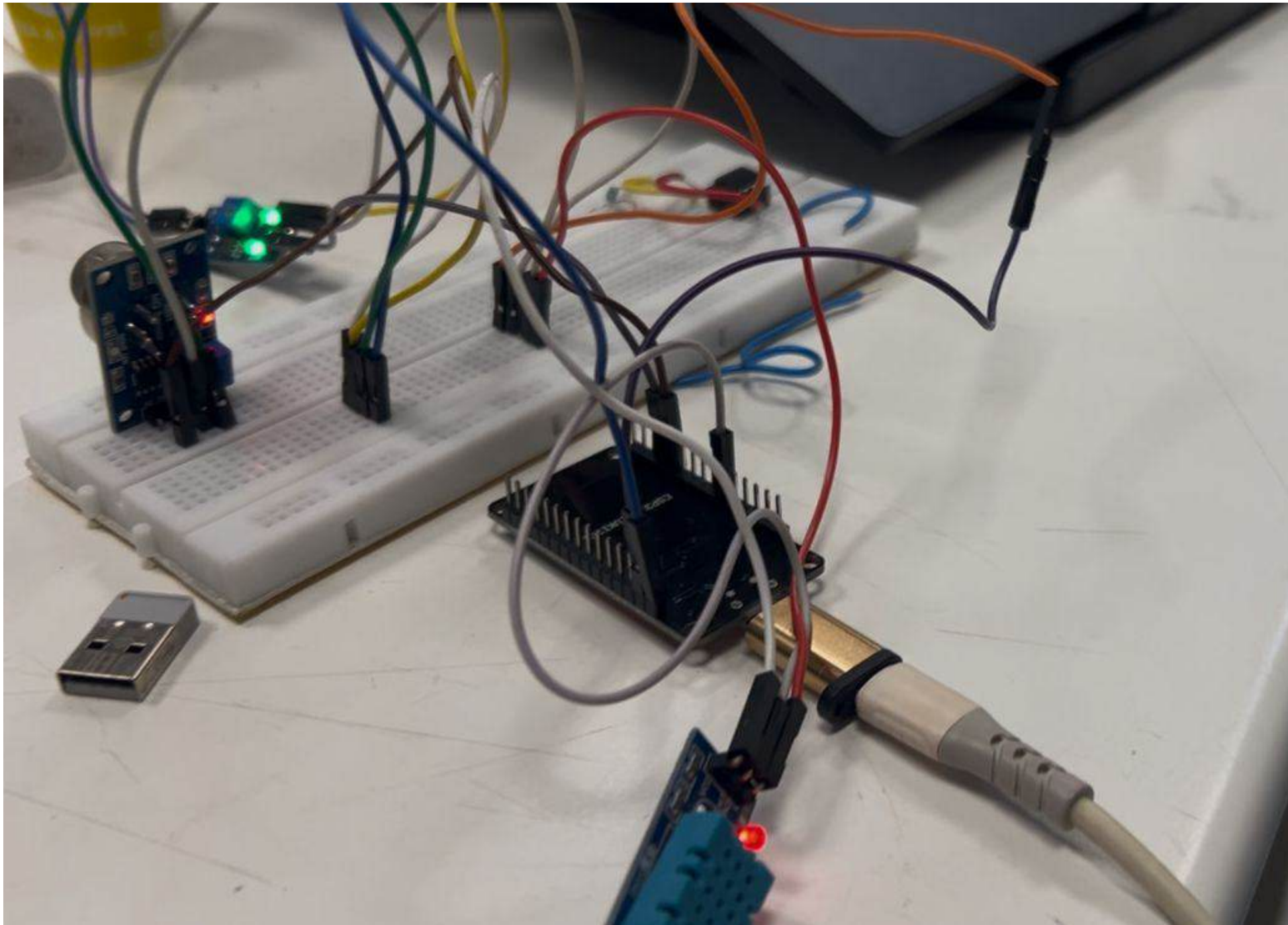
- Understood how a soil moisture sensor outputs analog values that can be mapped into percentages.
- Learned to connect ESP32 to WiFi and send sensor readings to ThingSpeak using an API key.
- Used ThingSpeak to visualize soil moisture data with:
 - Time-series graph (moisture trend over time).
 - Numeric display (real-time moisture value).
- Observed that stable WiFi and power are critical for continuous data logging.
- Gained experience in scaling raw analog values (0–4095) into a meaningful range (0–100%).
- Built a simple IoT application for agriculture, showing how soil conditions can be monitored remotely.



Soil Moisture+ DHT+ Gas sensor to ESP 32

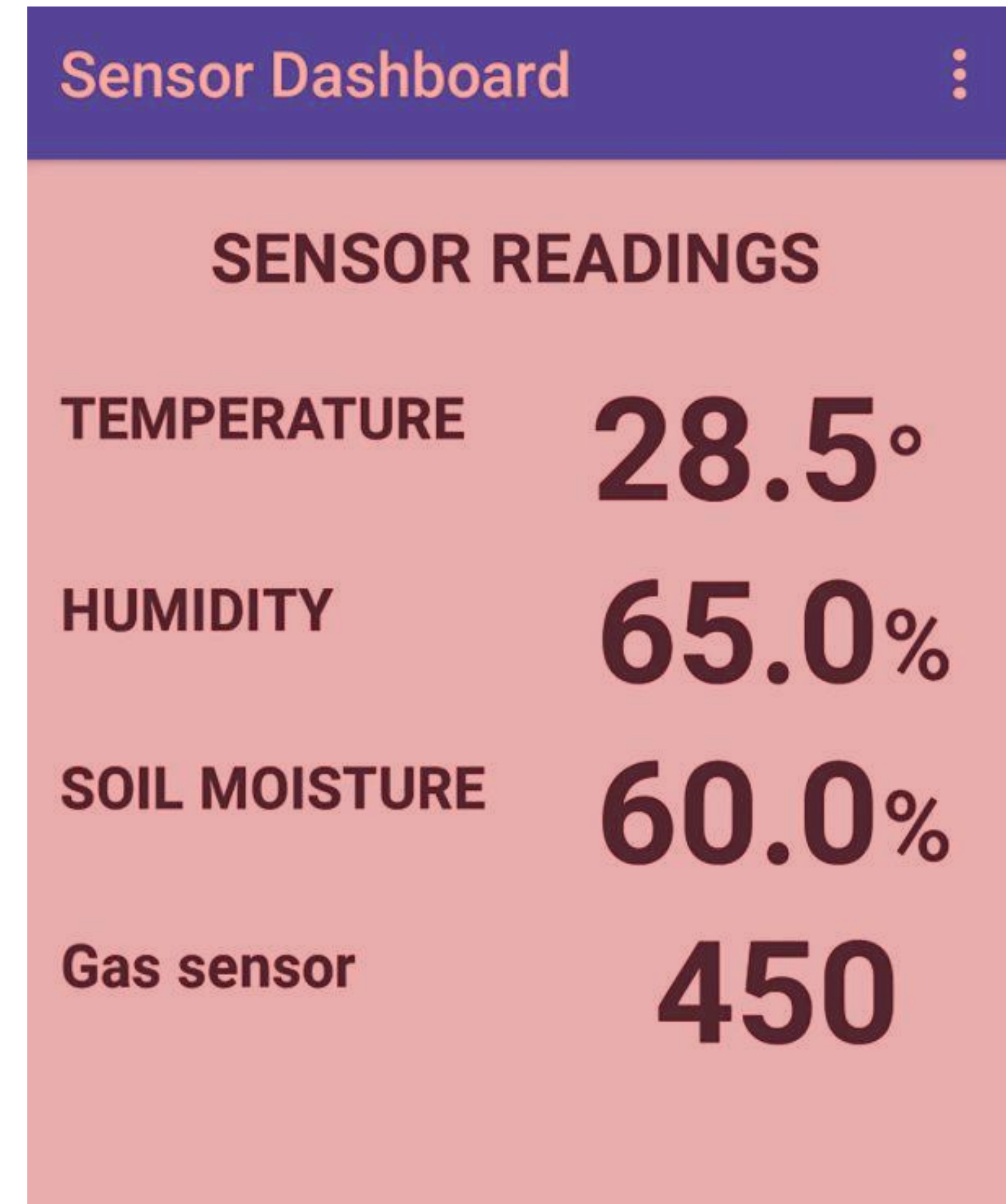


Output

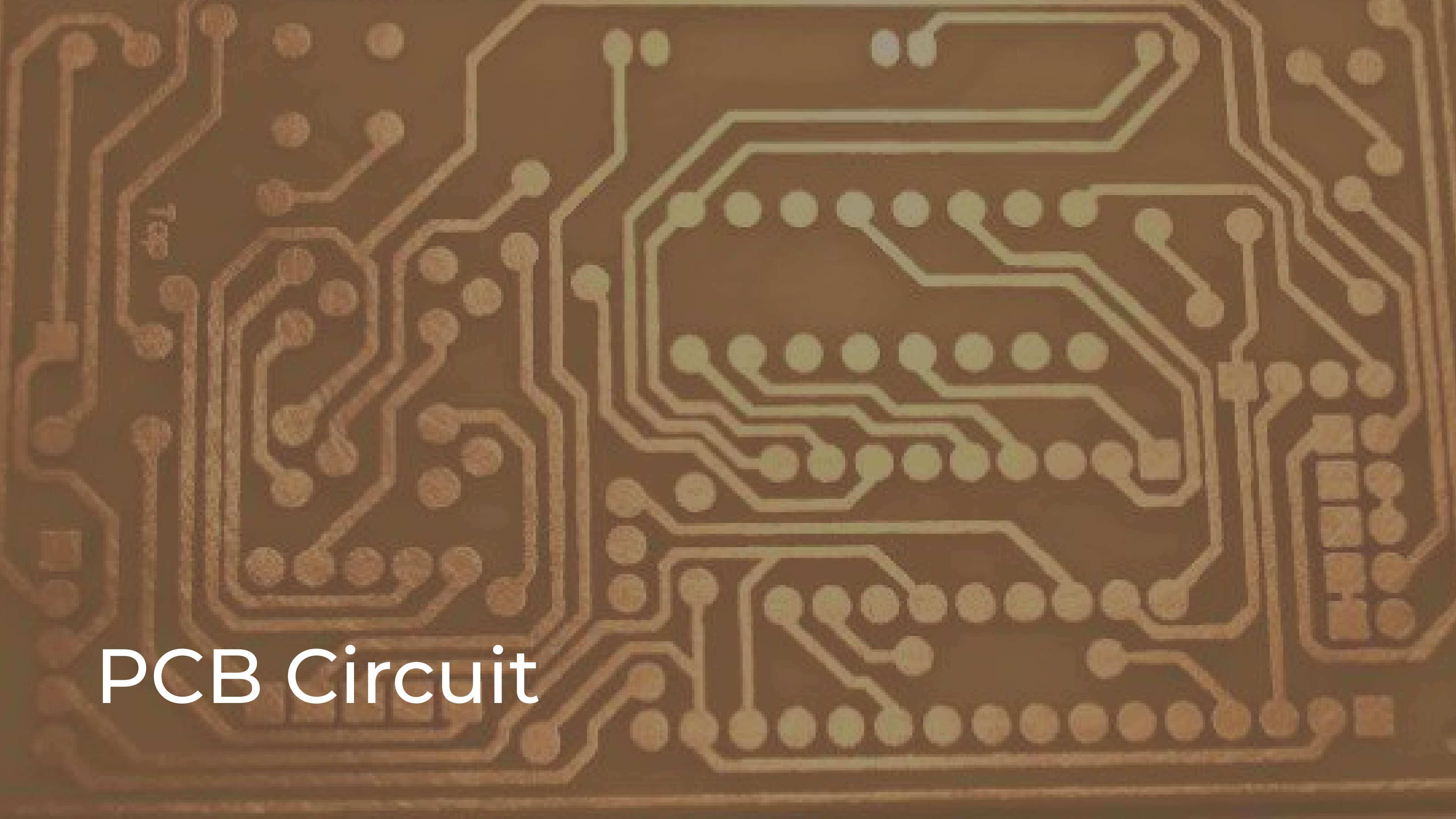


Our Learnings:

- Learned how to integrate multiple sensors (Soil Moisture, DHT, Gas) with ESP32.
- Understood sensor calibration and how raw values differ from usable data.
- Gained experience in Wi-Fi connectivity with ESP32 for IoT applications.
- Learned to use ThingSpeak for real-time data visualization and cloud storage.
- Observed how different environmental factors (temperature, humidity, soil moisture, gas levels) can be monitored together.
- Understood the importance of data intervals (15 seconds for free ThingSpeak).
- Learned how to format and upload data from ESP32 to a cloud dashboard.
- Identified the importance of data consistency and noise handling in sensor readings.

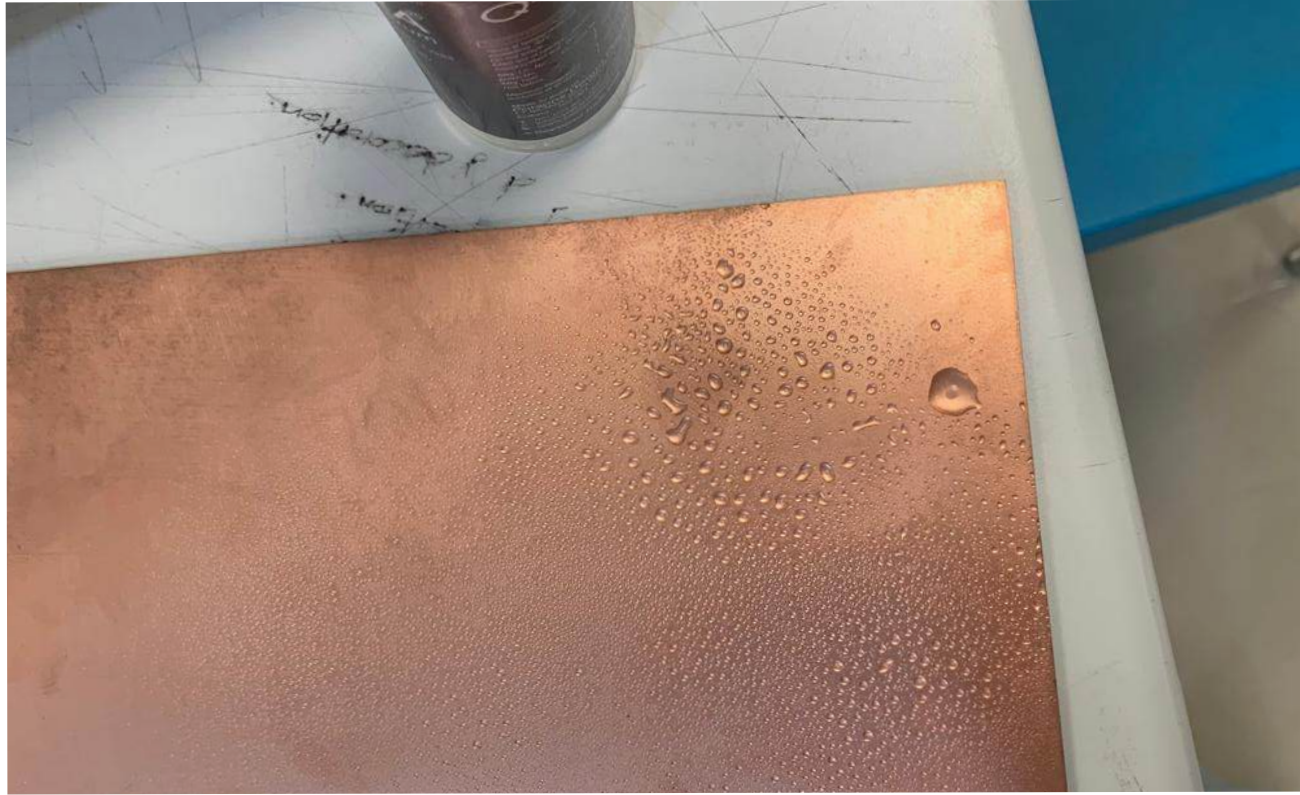


Readings on MIT App



PCB Circuit

Surface preparation



Scrub the copper cladding sheet with a steel rule to remove oxidation and ensure a clean surface.



Place the printed circuit sheet (toner side facing copper) onto the copper cladding.



Iron for 2 minutes in place (to make it stick) and then 5 minutes in circular motion to spread heat evenly.



Run under cold water and remove paper



Circuit is now on the copper board, in case of any broken lines drawing it in with a permanent marker and cut the section of copper with the circuit

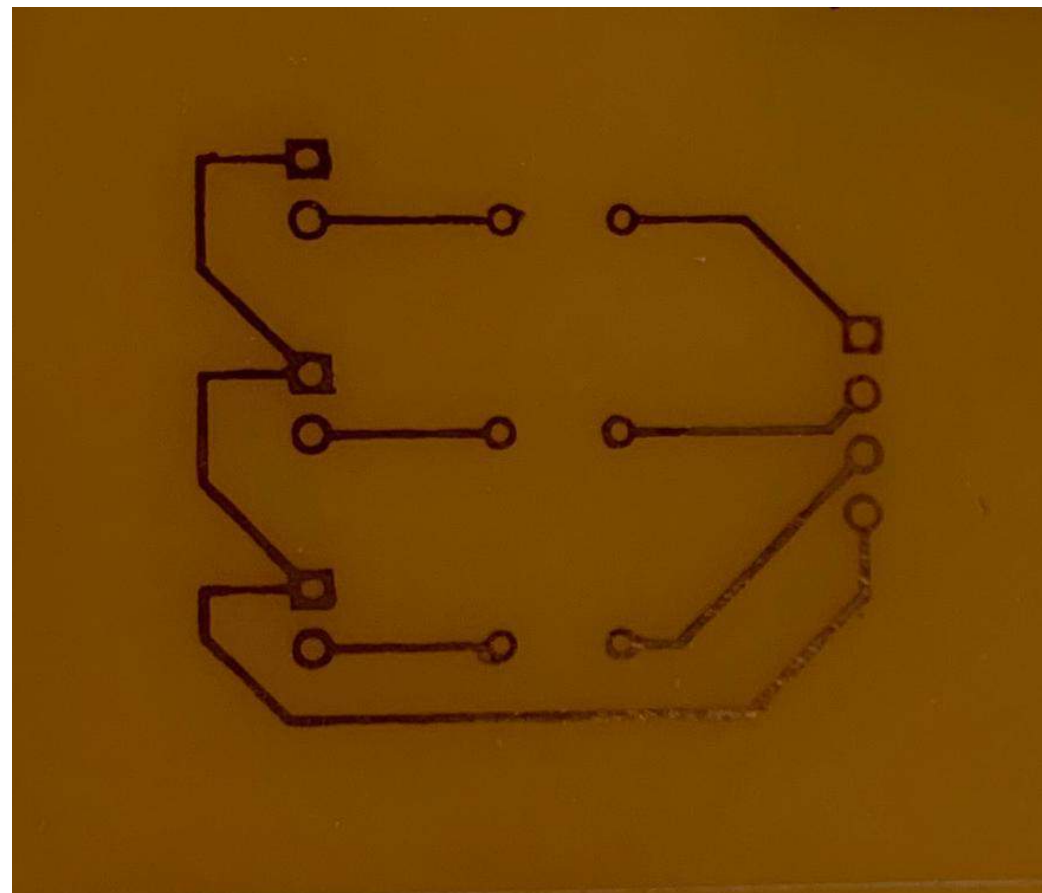
Solution preparation



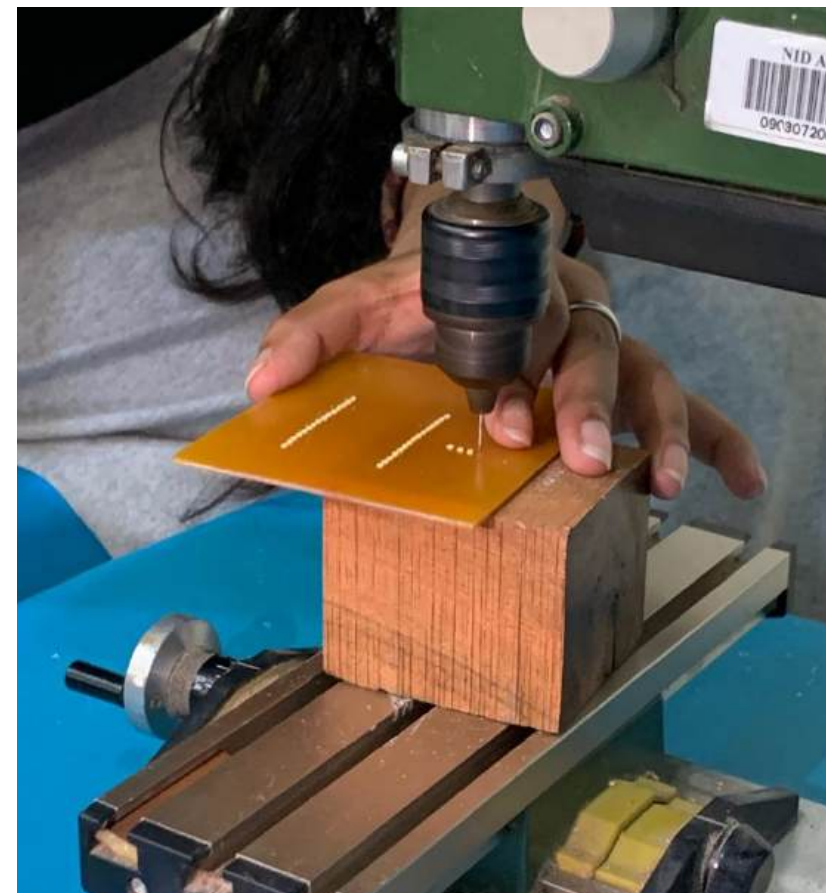
Ferric chloride anhydrous with water in 1:1 concentration and mix properly.



Submerge the copper-clad board in the solution, agitate for about 10 minutes until the copper fades to yellow, wash it and then use thinner to remove the black lines.

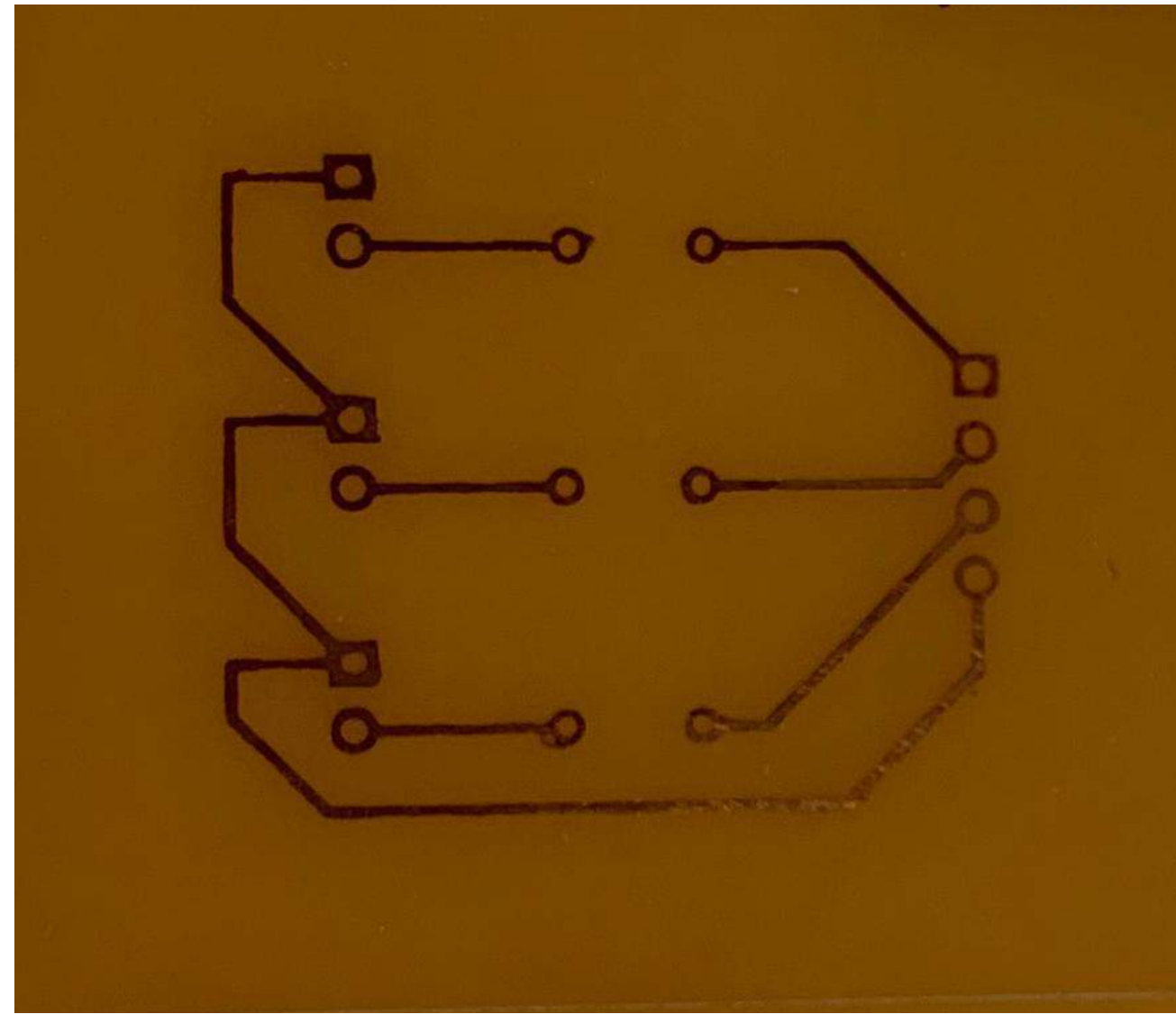


check the finished pcb connections with multi meter to ensure the pcb circuit is correct



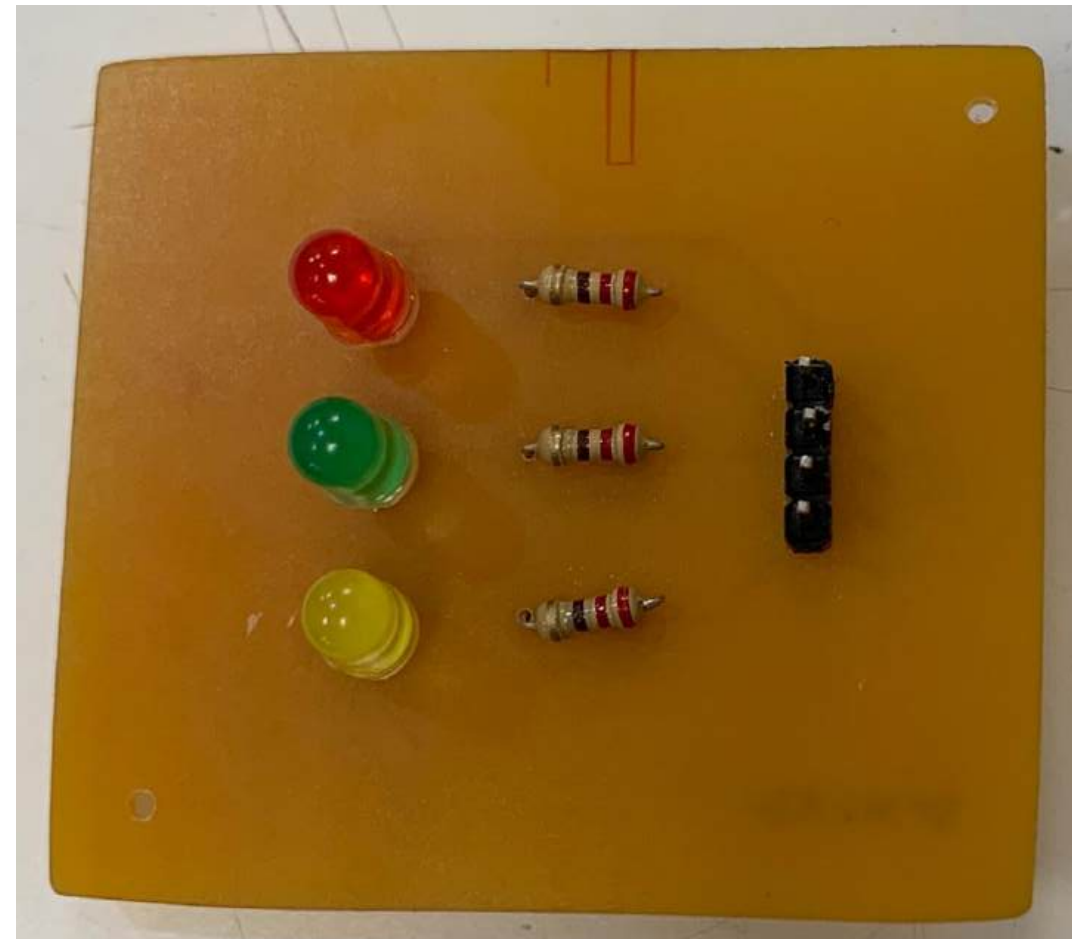
Drill holes with 0.5mm bit with a drill press to make place to hold components

3 LED on PCB board

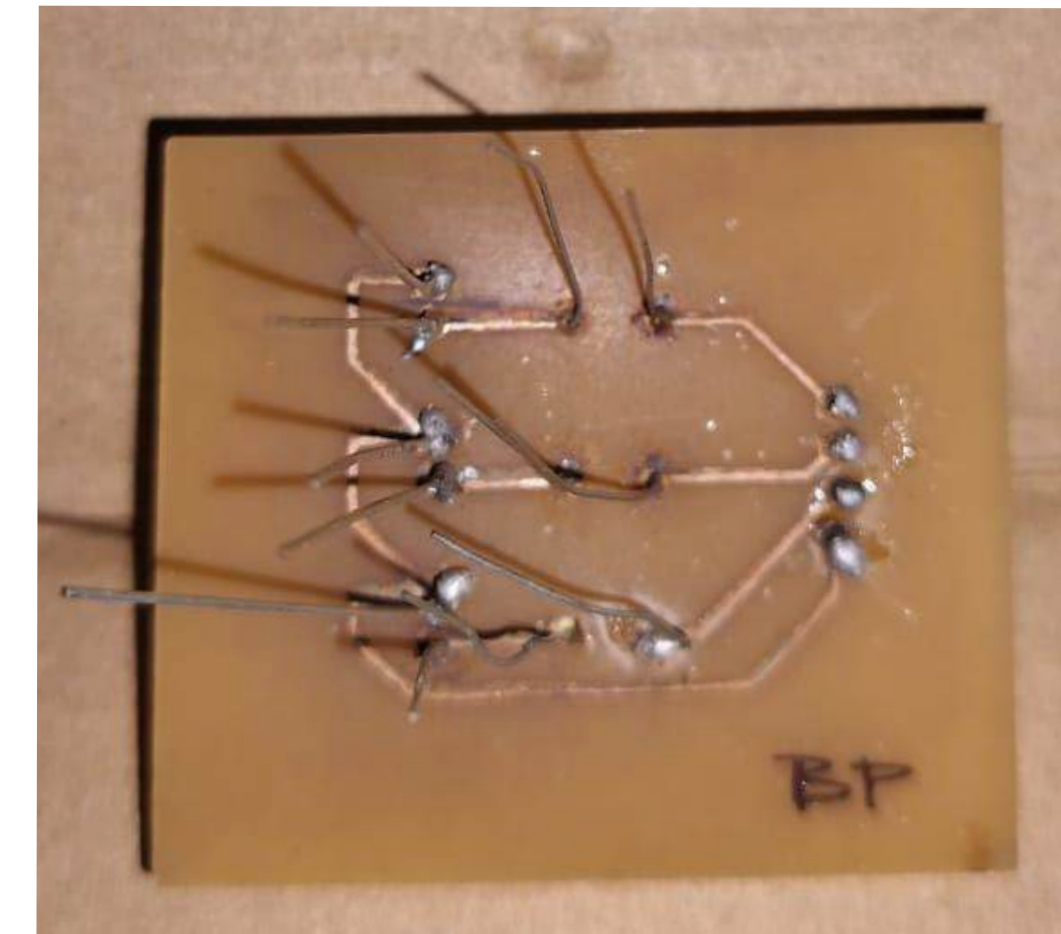


PCB board of 3 LEDs

Components:
3 RGB lights
3 Resistors (250 ohms)
4 pin male birchpin

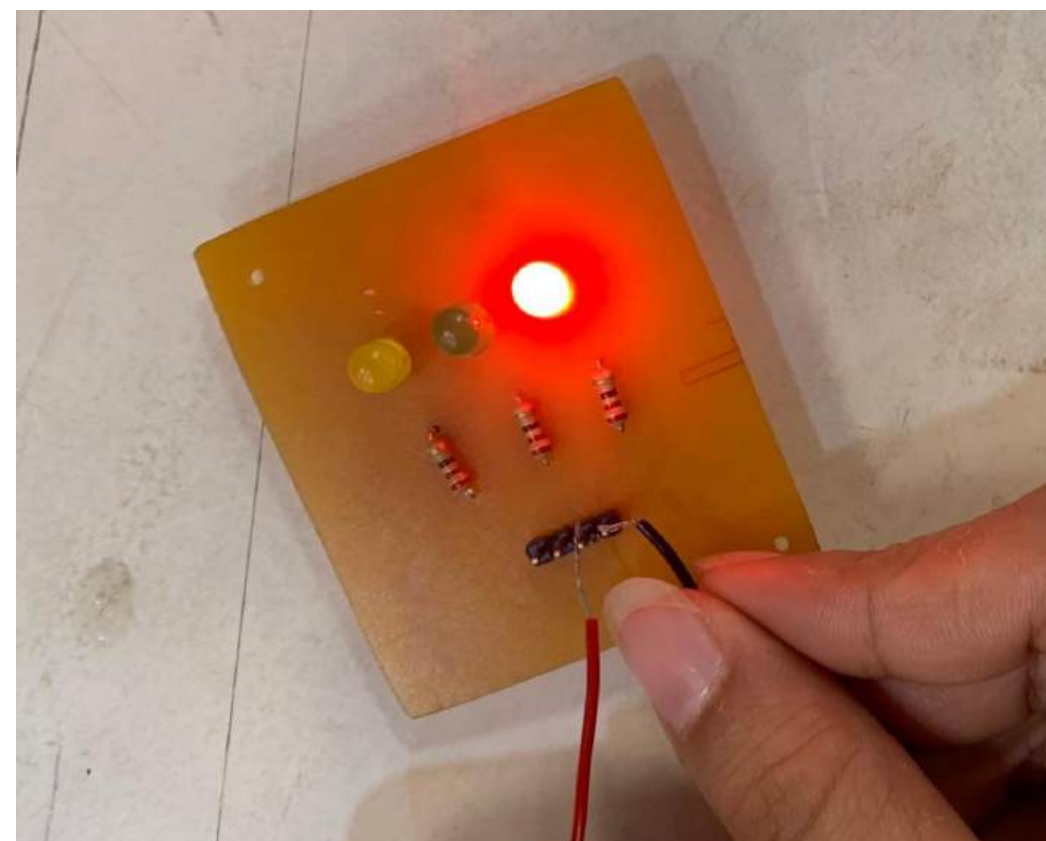


With components in place

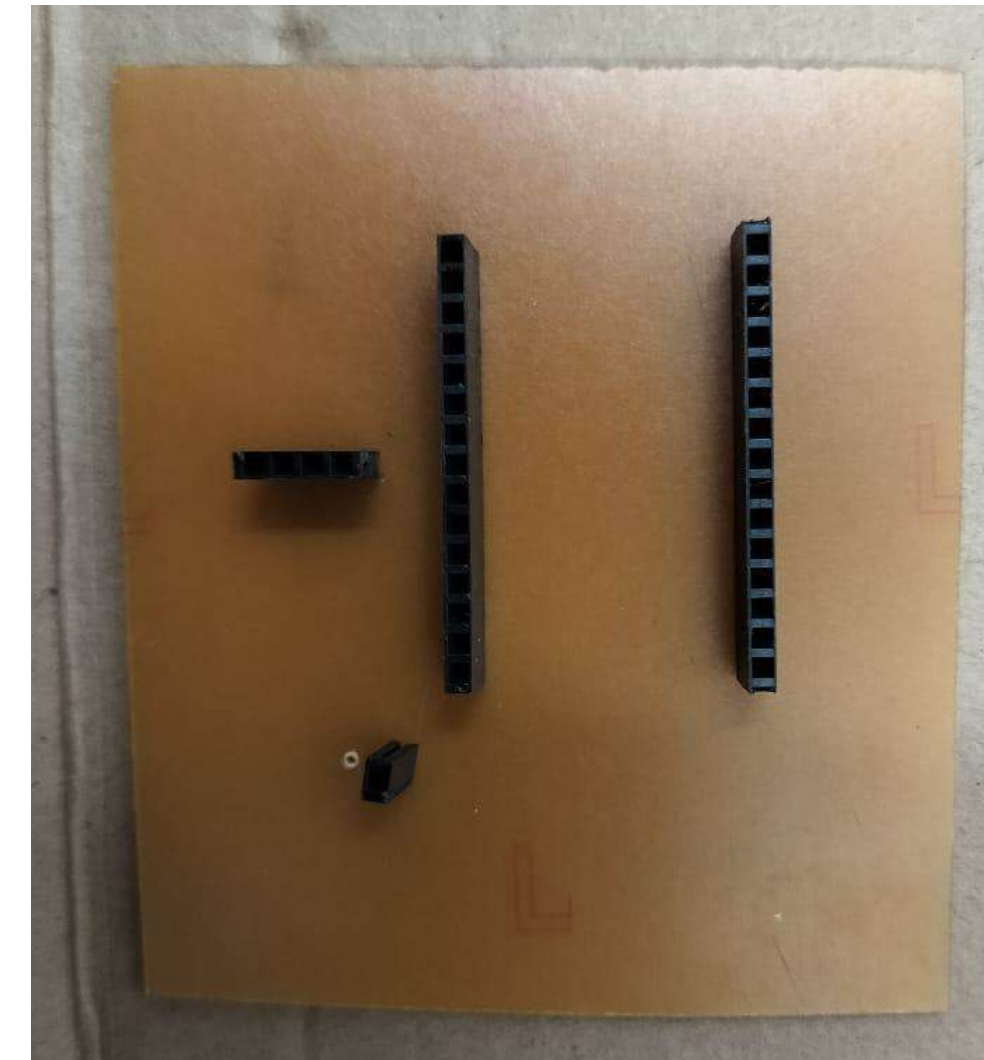
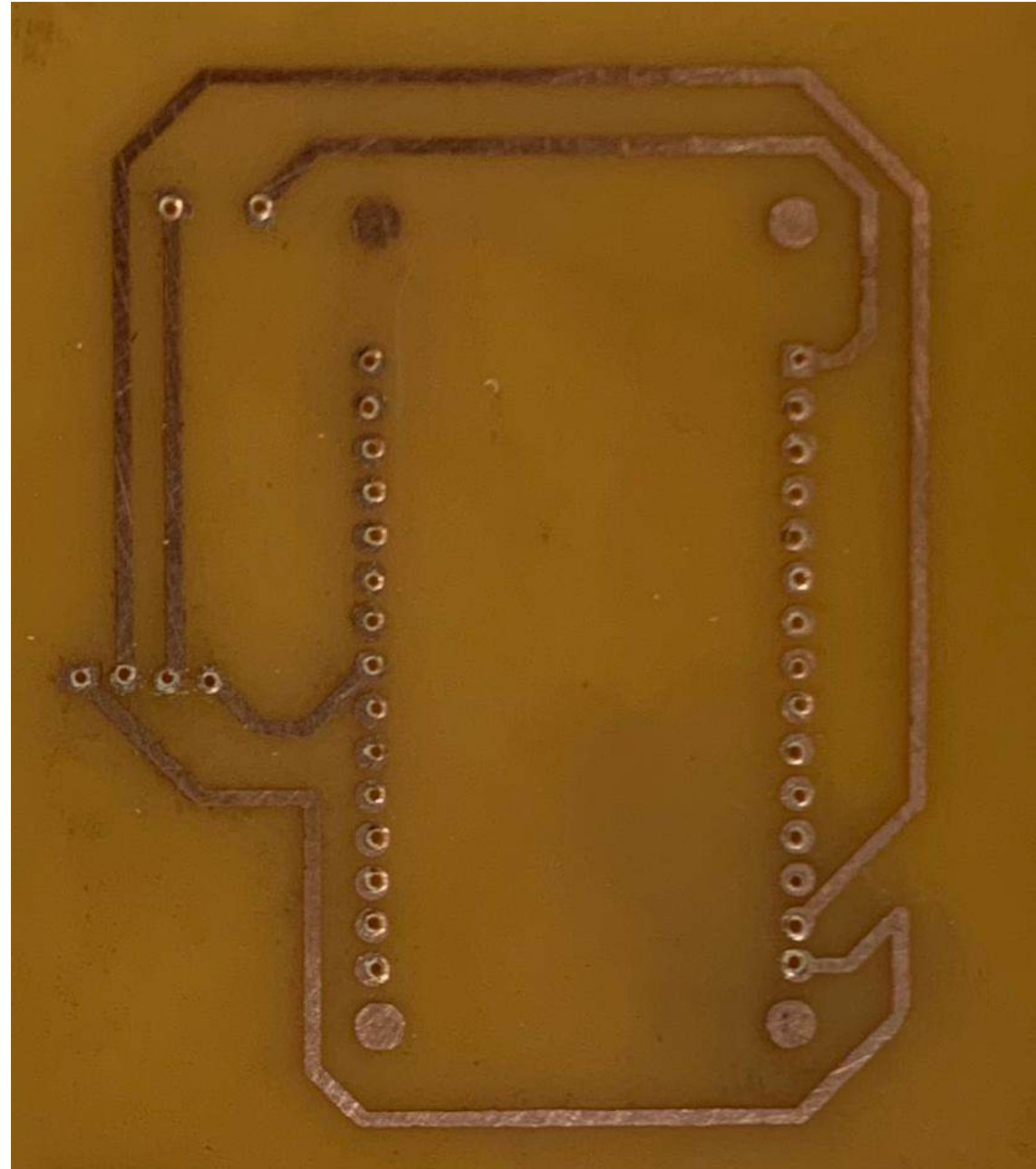


Backside of circuit after soldering components

Testing with power supply



1 RGB on the PCB board



Soldering the birchpins



Backside of circuit after soldering components



With components in place

PCB board of 1 RGB

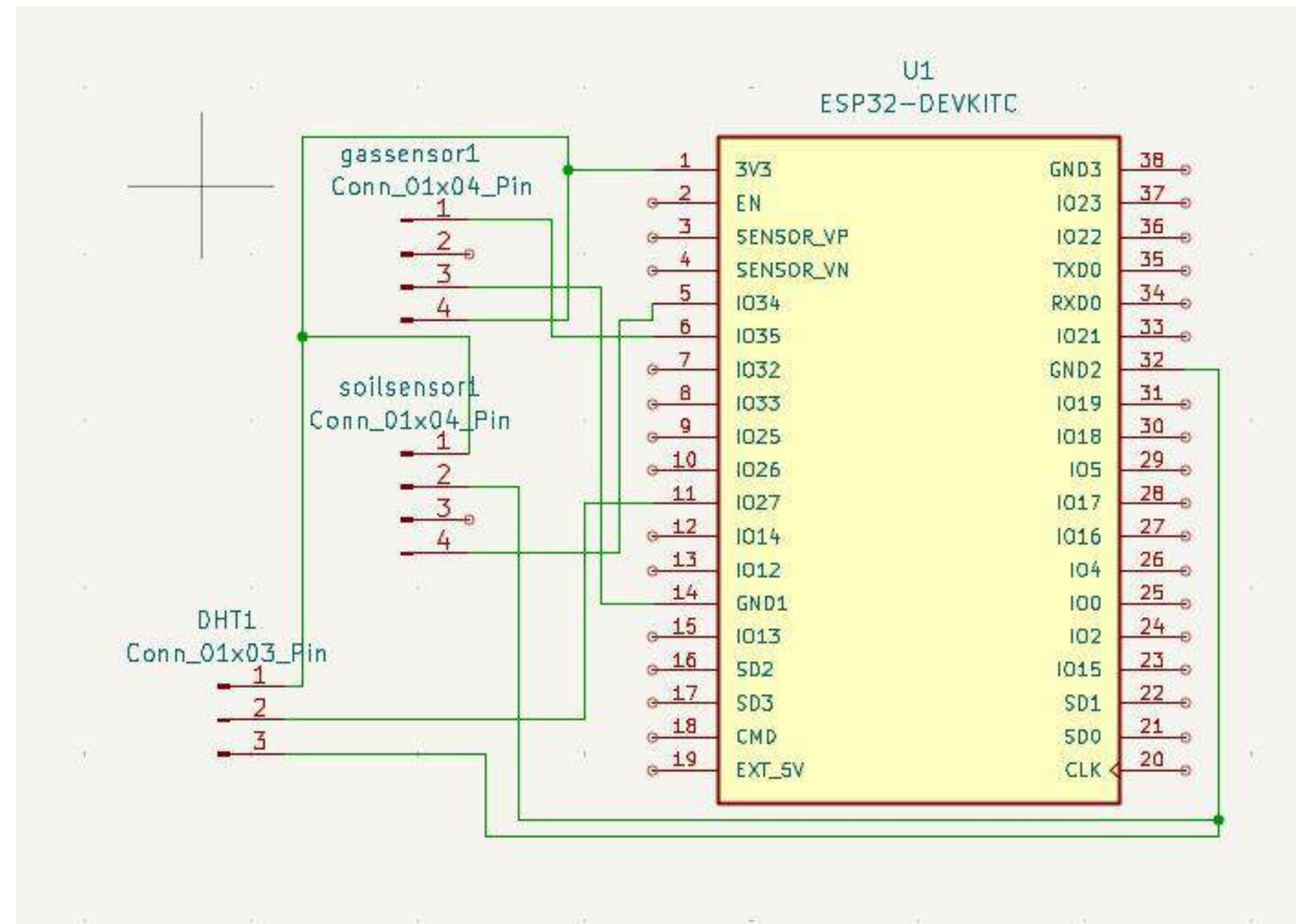
Components:

ESP32

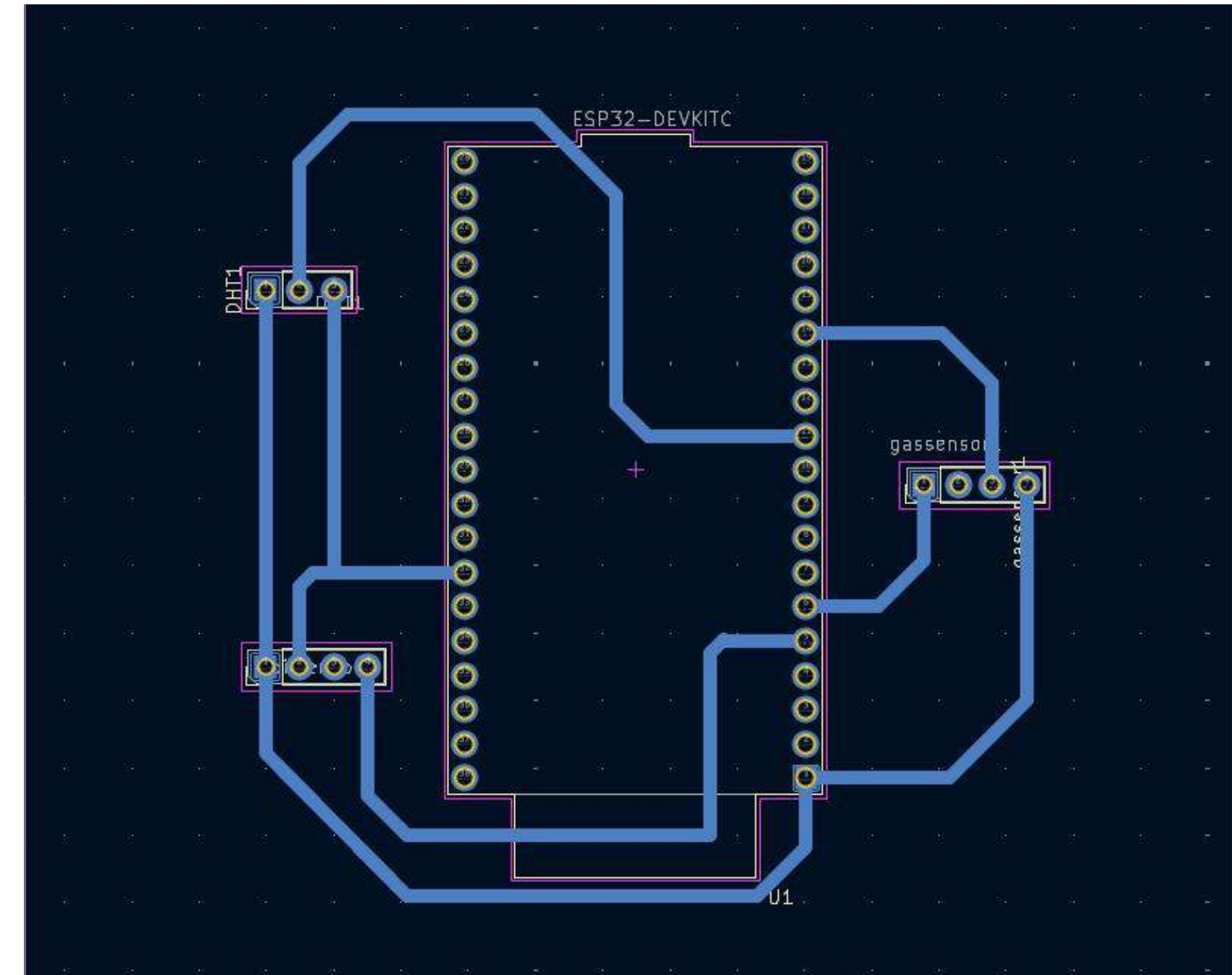
1 Resistor (250 ohm)

1 RGB LED

Kicad of Soil sensor , DHT-11 and gas sensor



Schematic: Logical wiring of ESP32 to DHT (temperature/humidity), soil sensor (moisture), and gas sensor. Each has power, ground, and signal lines mapped to GPIO pins.



PCB Layout: Physical board design with ESP32 in the center and connectors for the three sensors routed by copper tracks.

Final Project- Binny

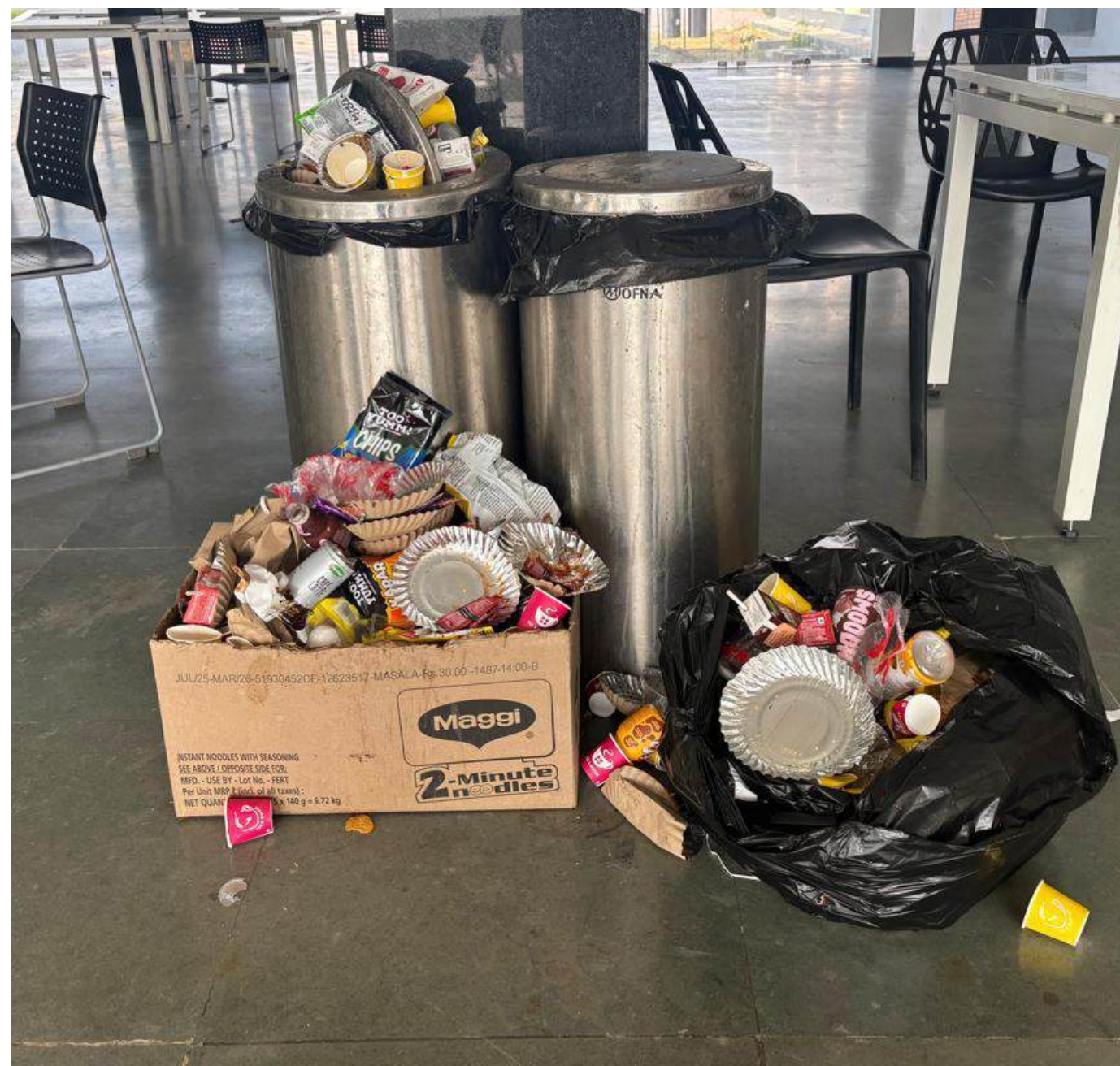
Smart robot dustbin

Why We Chose This Project:

In cafeterias and public dining areas, a common issue is the accumulation of trash left on tables because people often feel too lazy to get up and throw it in the dustbin. This not only creates hygiene problems but also ruins the overall dining experience.

Our idea is to **gamify the process of waste disposal**. By making the dustbin **interactive and mobile**, users can control it via their smartphones, instead of walking to the dustbin, the dustbin comes to you. This playful approach encourages better waste management habits.

Observations:



Despite having trashbins, trash is carelessly thrown on the tables of the floor in college cafeteria

Key Features that we wanted in the Project:

1. Automatic Opening and Closing of the Dustbin

- The dustbin lid will open and close automatically using a servo motor.
- An ultrasonic sensor added so the lid opens when trash is brought close to it, ensuring hands-free operation and improved hygiene.

2. Controlling of the Dustbin's wheels via App Commands

- The dustbin will be mounted on a set of wheels driven by DC motors.
- Users can control the movement of the dustbin (forward, backward, left, right) through a custom mobile application connected via Bluetooth.

Additional functions

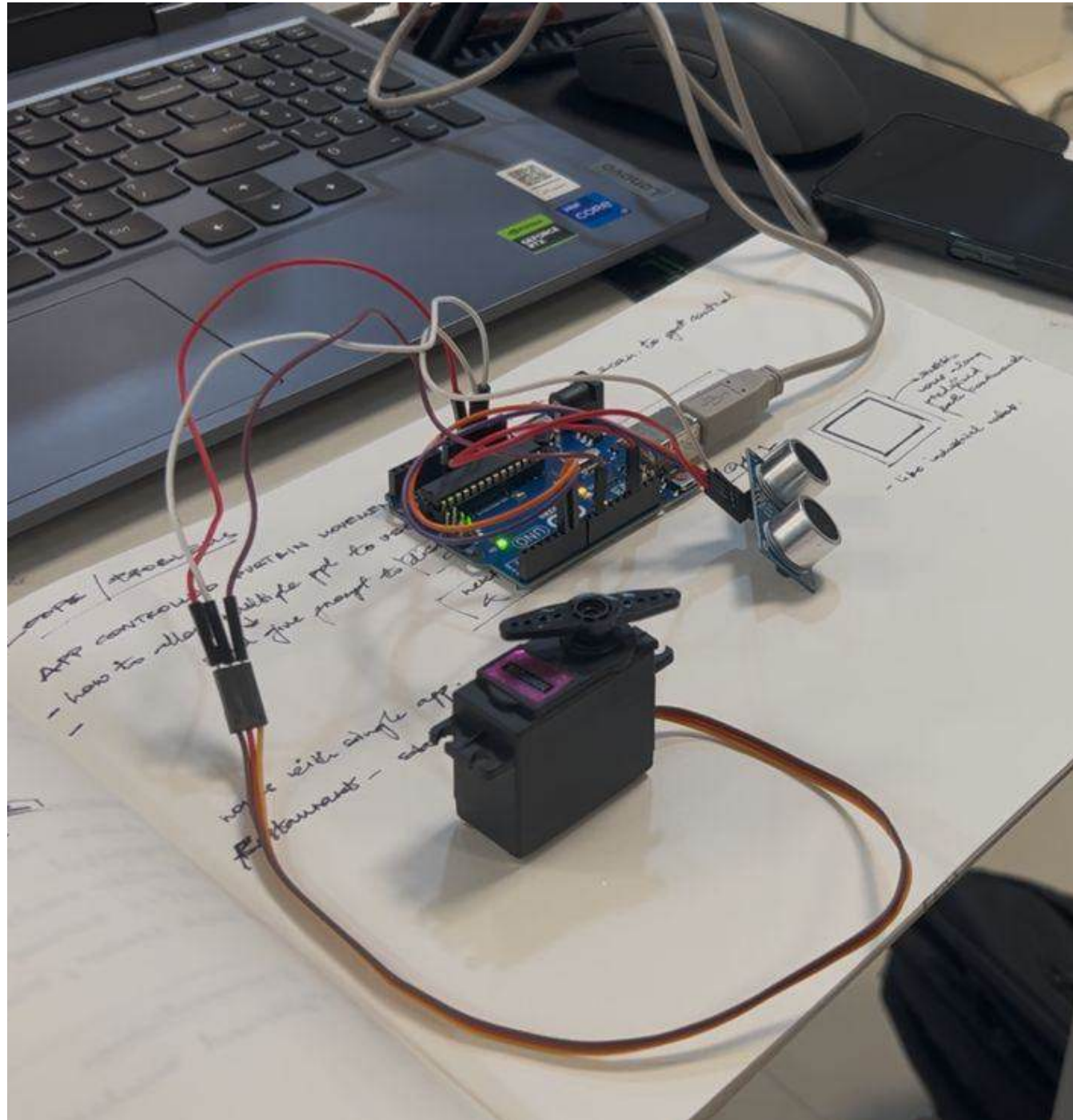
Since it will be in the cafeteria, we shud allow seamless connection to anyone. One at a time.

Battery life

Weight of trash

Cleaning mechanism

PART 1- Opening and closing of dustbin lid

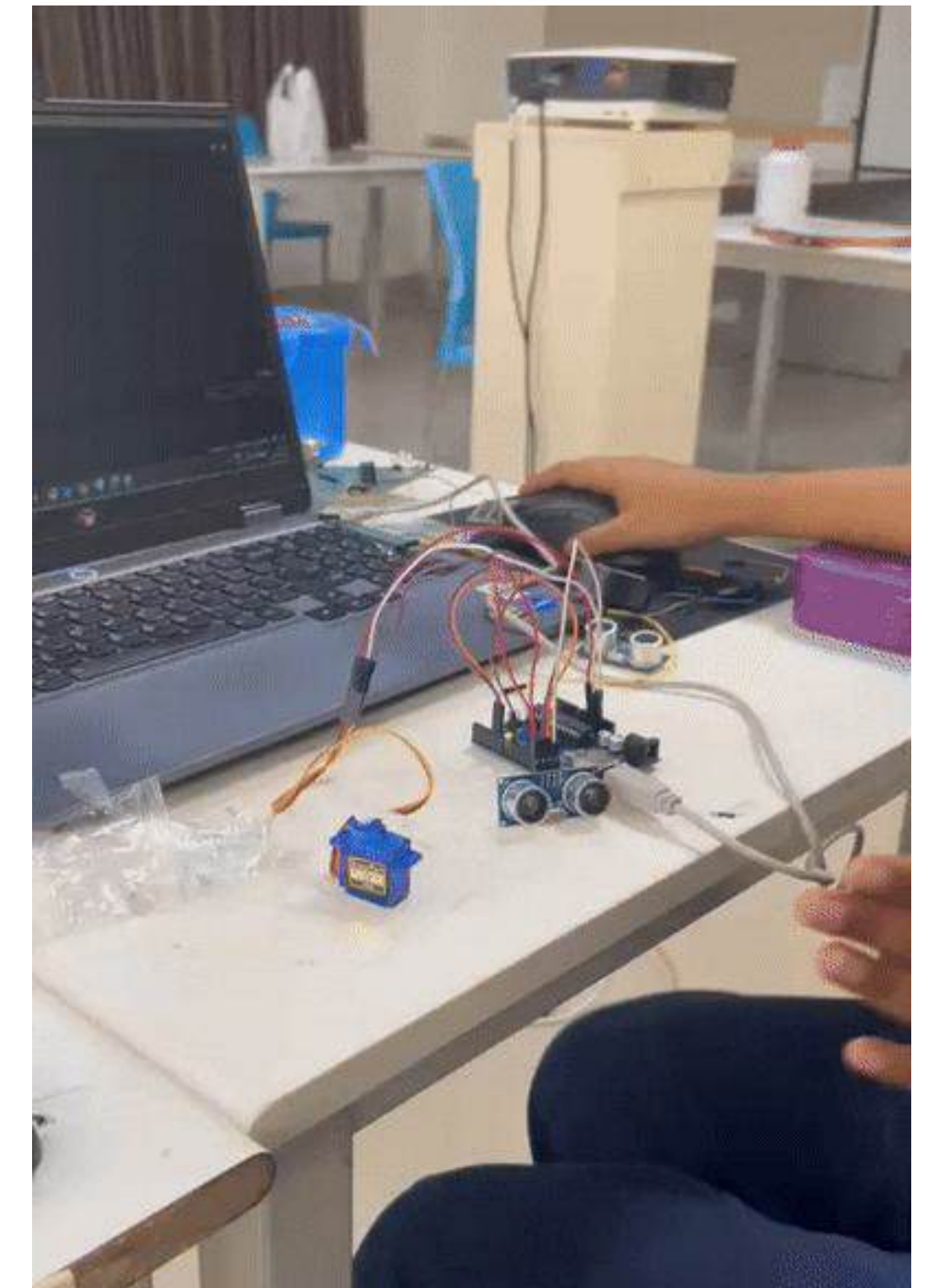


Components used-

- Arduino Uno
- Ultrasonic sensor
- MG996R Metal Gear Servo
- Jumper cables
- Dustbin

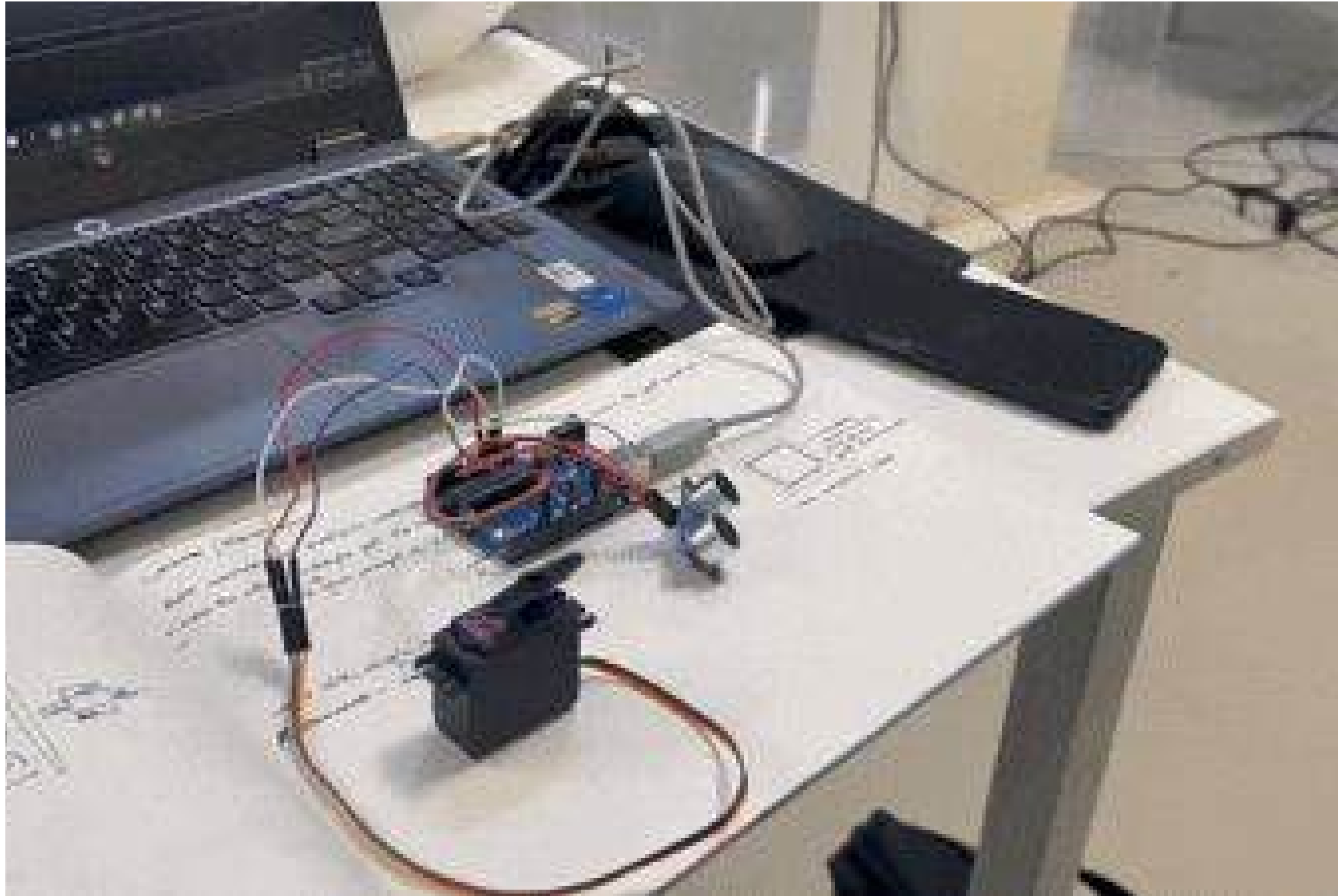
We started out by trying to run the sensors with the arduino code before attaching the components to the dustbin.

```
sketch_DUSTBINTOPFINAL14a.ino
1  #include <Servo.h>
2
3  Servo myservo;
4
5  int trigPin = 5;
6  int echoPin = 6;
7  long duration;
8  int distance;
9
10 void setup() {
11   Serial.begin(9600);
12   pinMode(trigPin, OUTPUT);
13   pinMode(echoPin, INPUT);
14   myservo.attach(9); // Servo on pin 9
15 }
16
17 void loop() {
18   // Ultrasonic reading
19   digitalWrite(trigPin, LOW);
20   delayMicroseconds(2);
21   digitalWrite(trigPin, HIGH);
22   delayMicroseconds(10);
23   digitalWrite(trigPin, LOW);
24
25   duration = pulseIn(echoPin, HIGH);
26   distance = duration * 0.034 / 2;
27   Serial.println(distance);
28   // Check if object is within 10 cm
29   if (distance <= 20 && distance > 0) {
30     myservo.write(180); // Open
31     delay(3000);        // Keep open for 2 seconds
32     myservo.write(0);   // Close
33     delay(500);         // Small delay to avoid flickering
34   }
35 }
```



1. The servomotor used was small and hence not strong enough to lift the lid of our dustbin
Fix- using a stronger servomotor(MG996R Metal Gear Servo)

Final connection plan for the lid



After ensuring the connection is working we attached a long piece of ABS plastic (5.28cm length) onto the servomotor to help it open the lid.



Made holes on the lid to fit the ultrasonic sensor.



Attached the sensors to the lid.

Testing and refining



Initially due to some issues in the code, it was giving reversed output (the bin closes when sensing something close and default stays in open state when nothing is sensed)

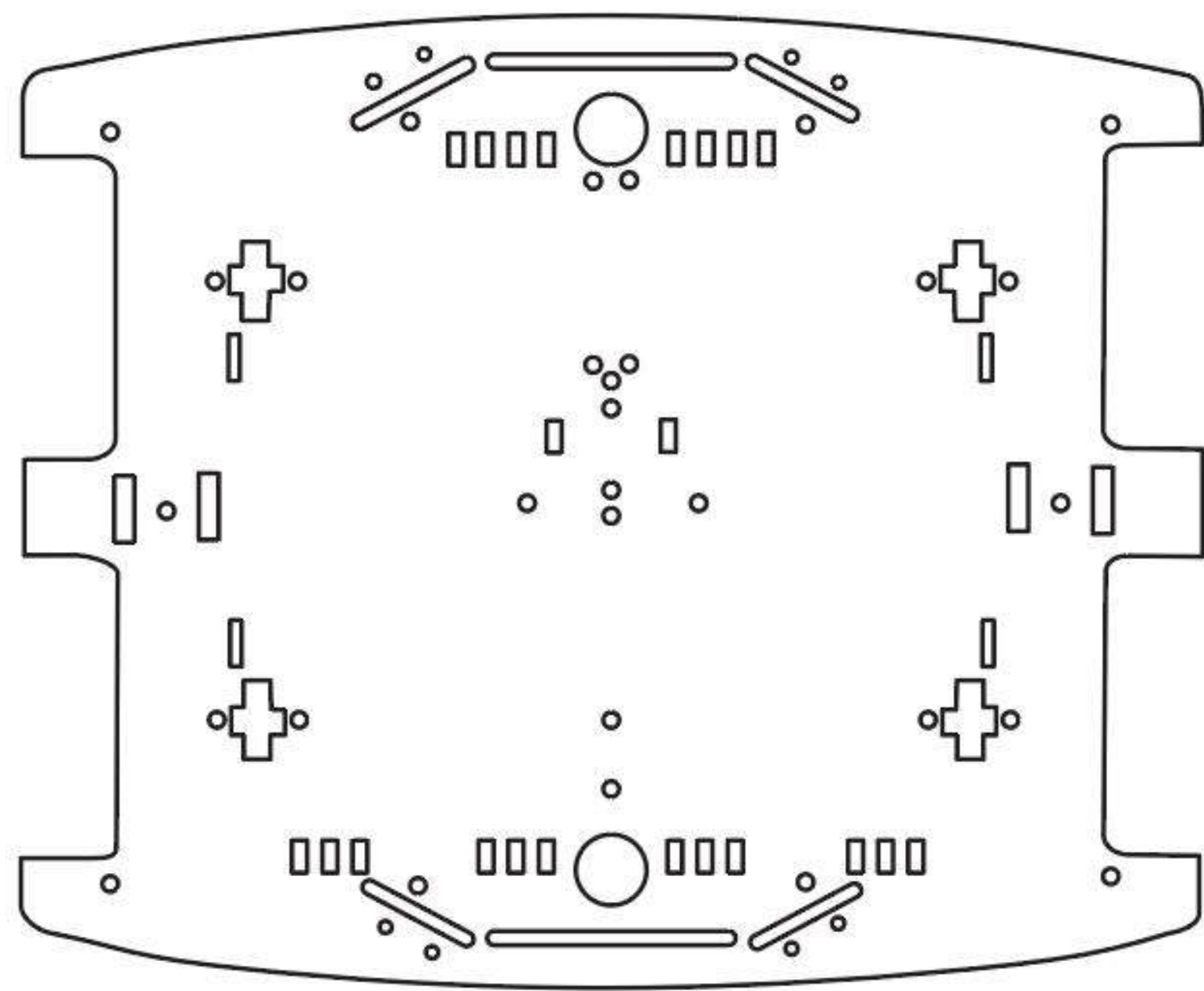


Troubleshoot- fixed the code , also added extra seconds to the amount of time it stays open after sensing something and increased the distance of sensing an object.

Final working of Part 1



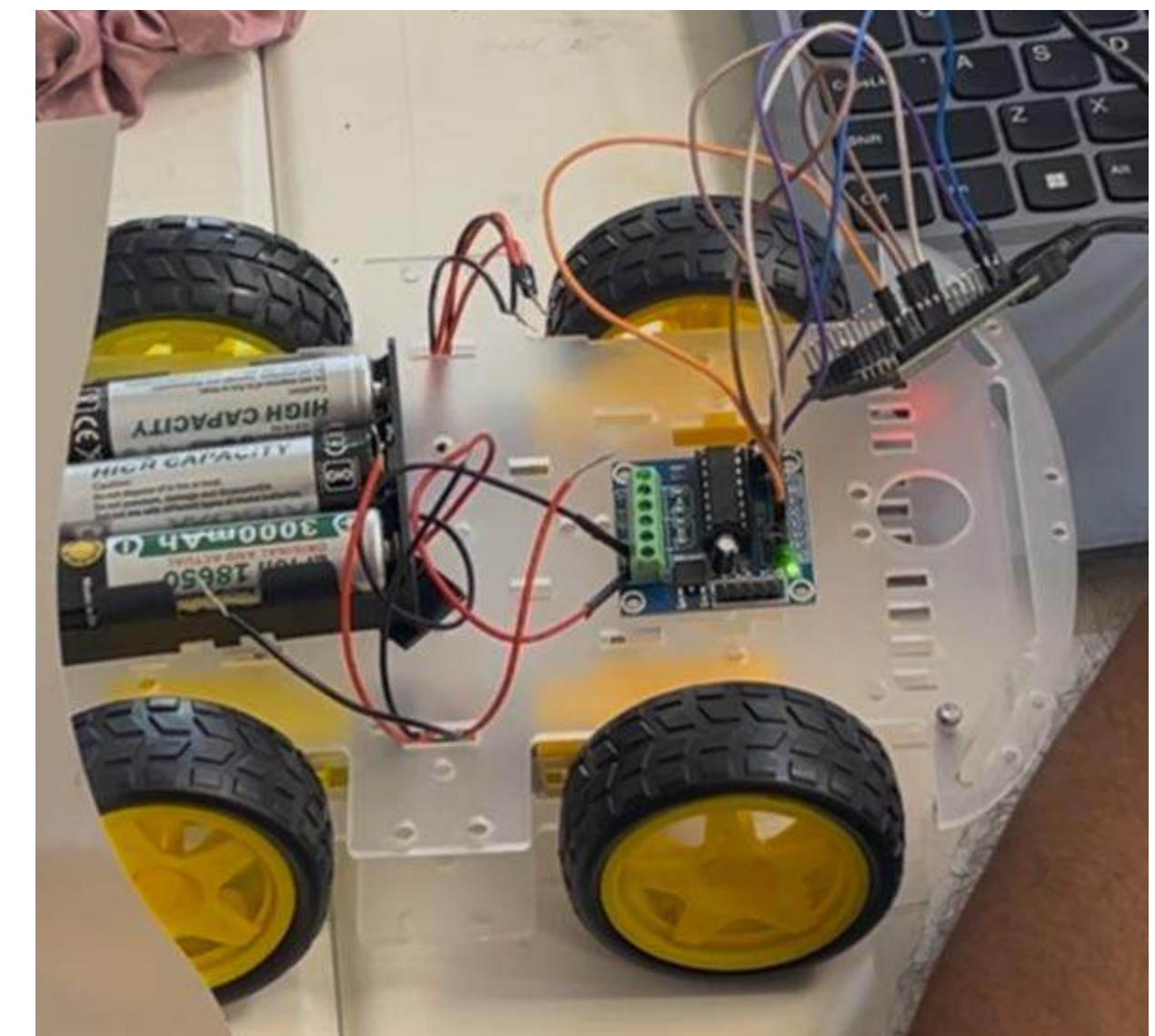
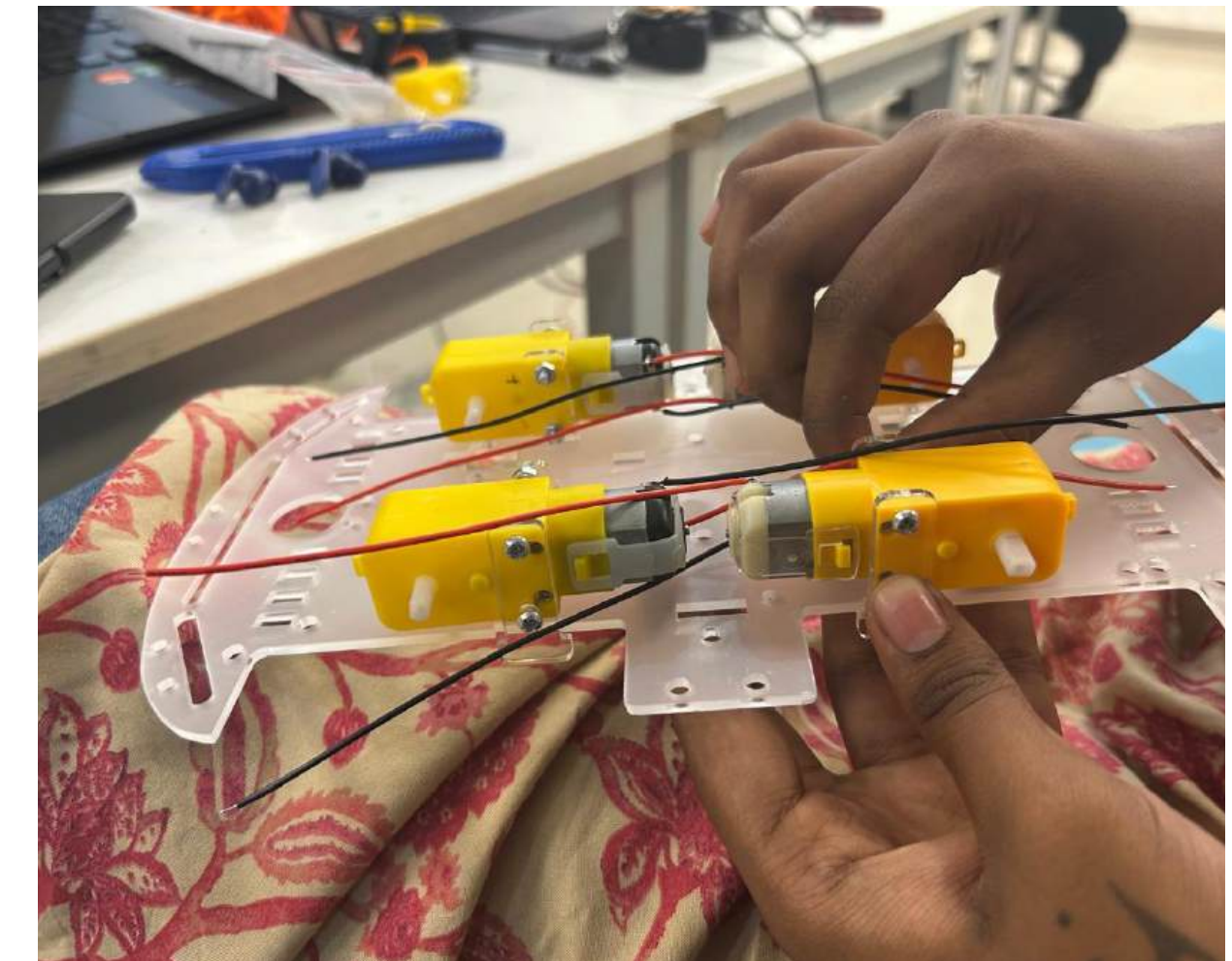
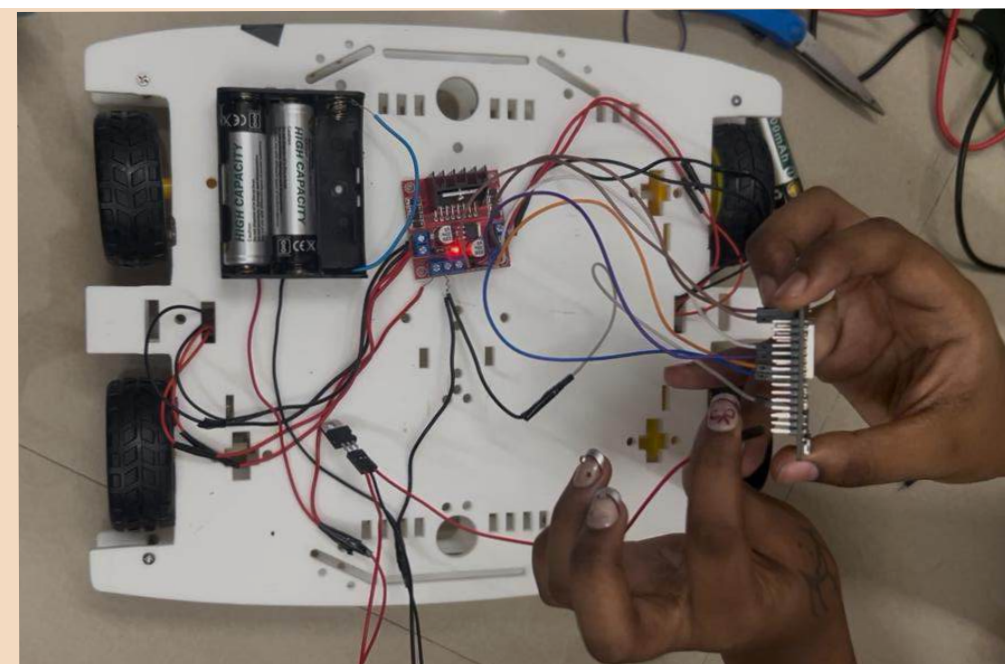
PART 2- Moving of the dustbin and app



We made a dxf file in illustrator for the custom chassis for placing our dustbin on. This was laser cut on ABS plastic sheet of 2mm thickness.

Components used-

- ESP 32
- L298N Motor Driver/L293D
- Chassis of robotic car
- 3 Battery pack (12V/9V)
- 7805ic



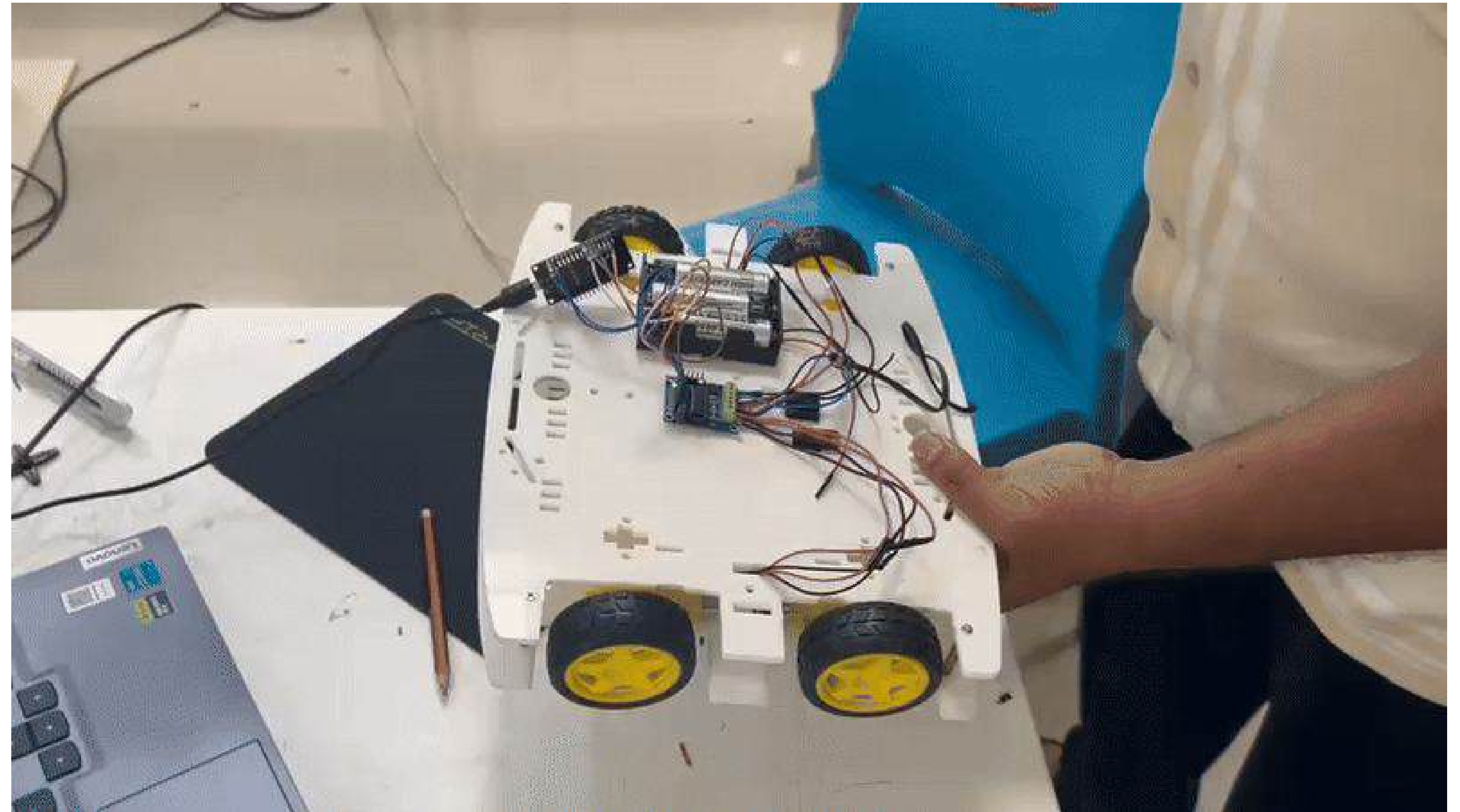
Assembling the robot car

Using L293D Motor Driver

- Tried running the motors with the L293D.
- Motors only **worked for a short while** and then **stopped**.
- Based on an online reference, decided to **switch to the L298N for better performance**.

Why it may have failed (L293D limitations):

- Can only supply about **600 mA continuous current per channel** (peaks up to ~1A). Many DC motors need more, especially under load.
- Higher voltage drop** across the internal transistors, so **motors get less usable voltage**.
- Less efficient for driving heavier** or higher-current **motors**, which causes stalling or shutting down after a short run.



Using L298n Motor Driver

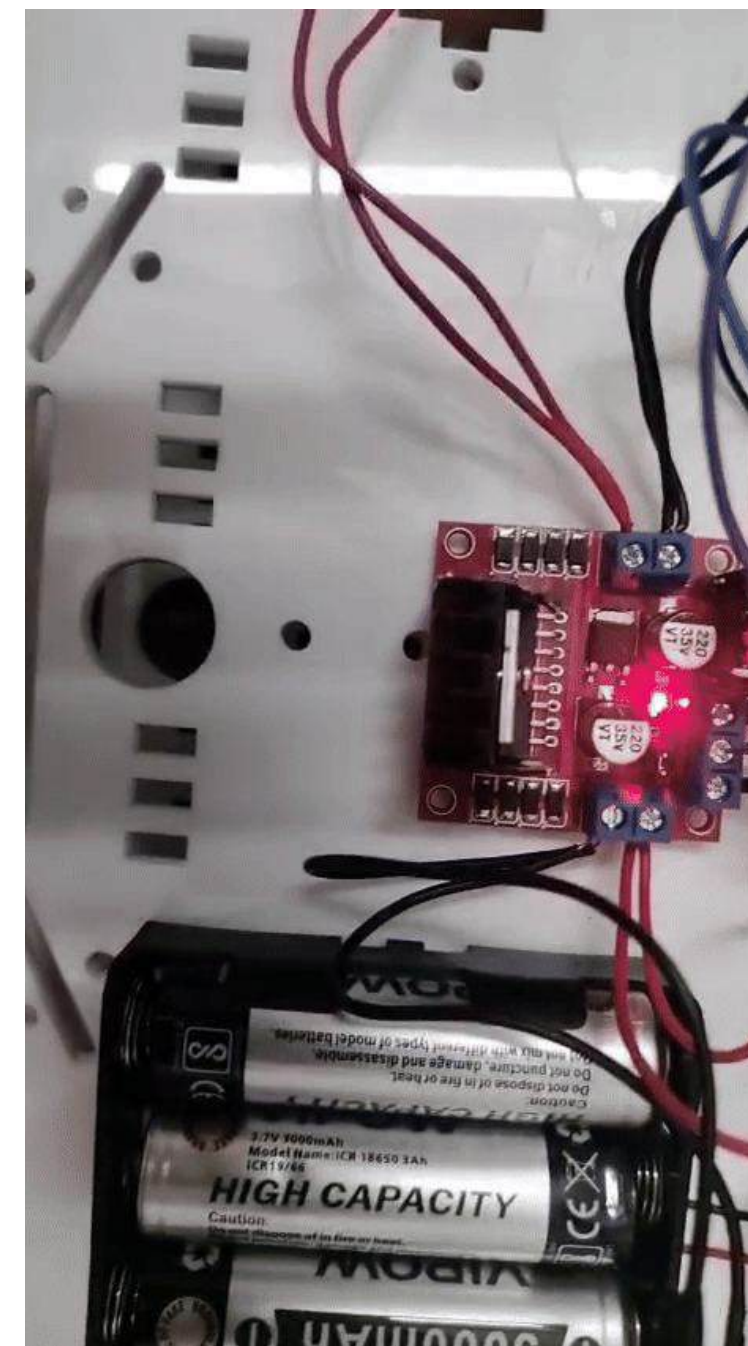
Problems and fixes tried

1. Connected ESP32 GND to the GND of the driver → indicator light dimmed.
2. Added a **7805 IC to regulate** voltage/current → **IC overheated**.
3. **Removed one cell** from the 12V (3-pack) battery, so it became **8V** approx., reconnected for continuity, and attached VIN to the driver's third 12V pin → the **7805 IC overheated again but slowly**
4. **Removed the battery connection** and powered ESP32 directly via laptop USB → **ESP32 started to overheat**.
 - Tested with a multimeter → **ESP32 had fused**.
5. When connecting the **vin pin to third 7805 ic pin at a particular angle**, only **one side of the car is moving**.

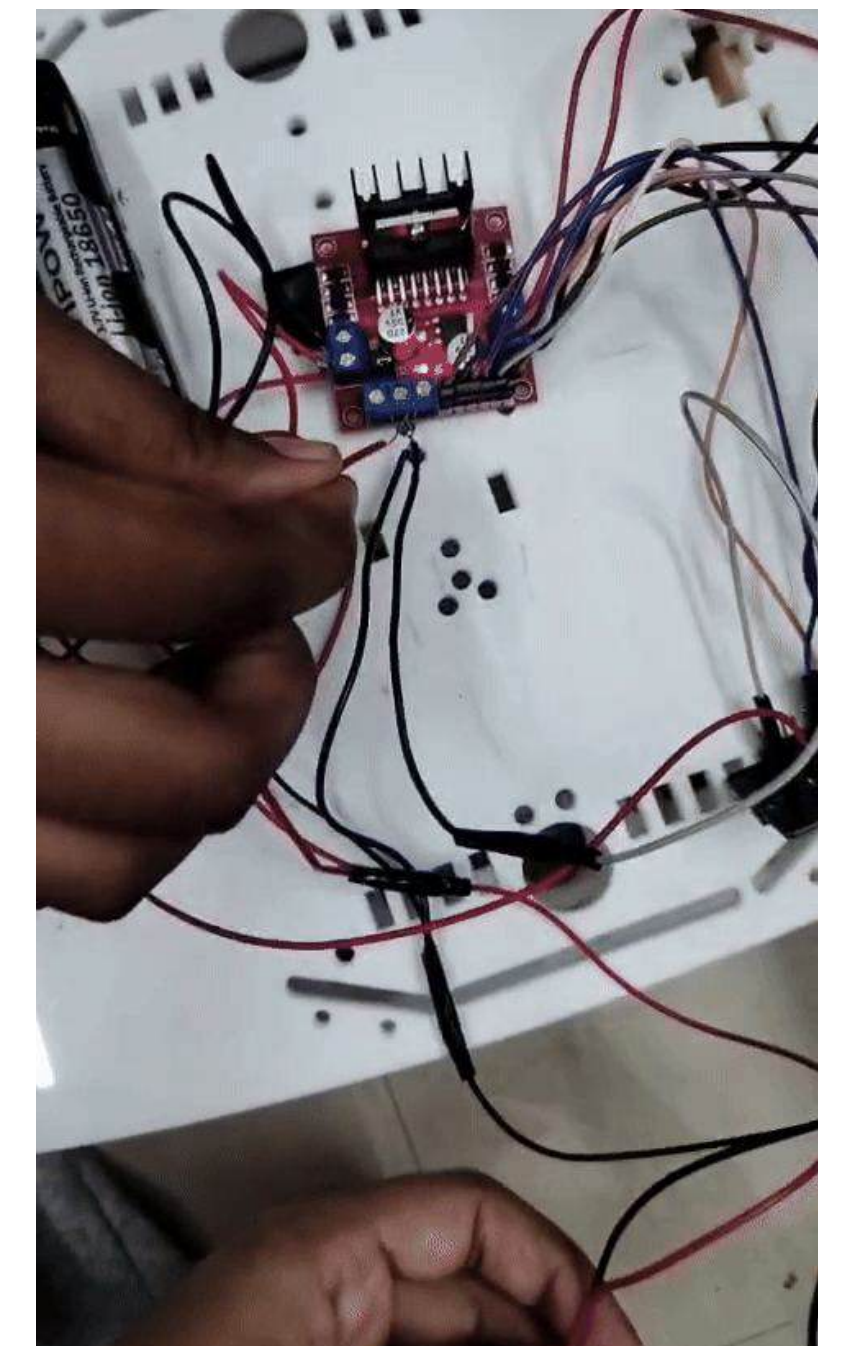
Why L298N was considered better:

- Handles up to **2A per channel, more than L293D**.
- Uses Darlington transistor arrays, giving it **more robustness for medium motors**.
- Has separate **motor supply (12V)** and **logic supply (5V)** pins, which helps with **stability**.
- Widely used with ESP32 robot projects, so easier to find references and tutorials.

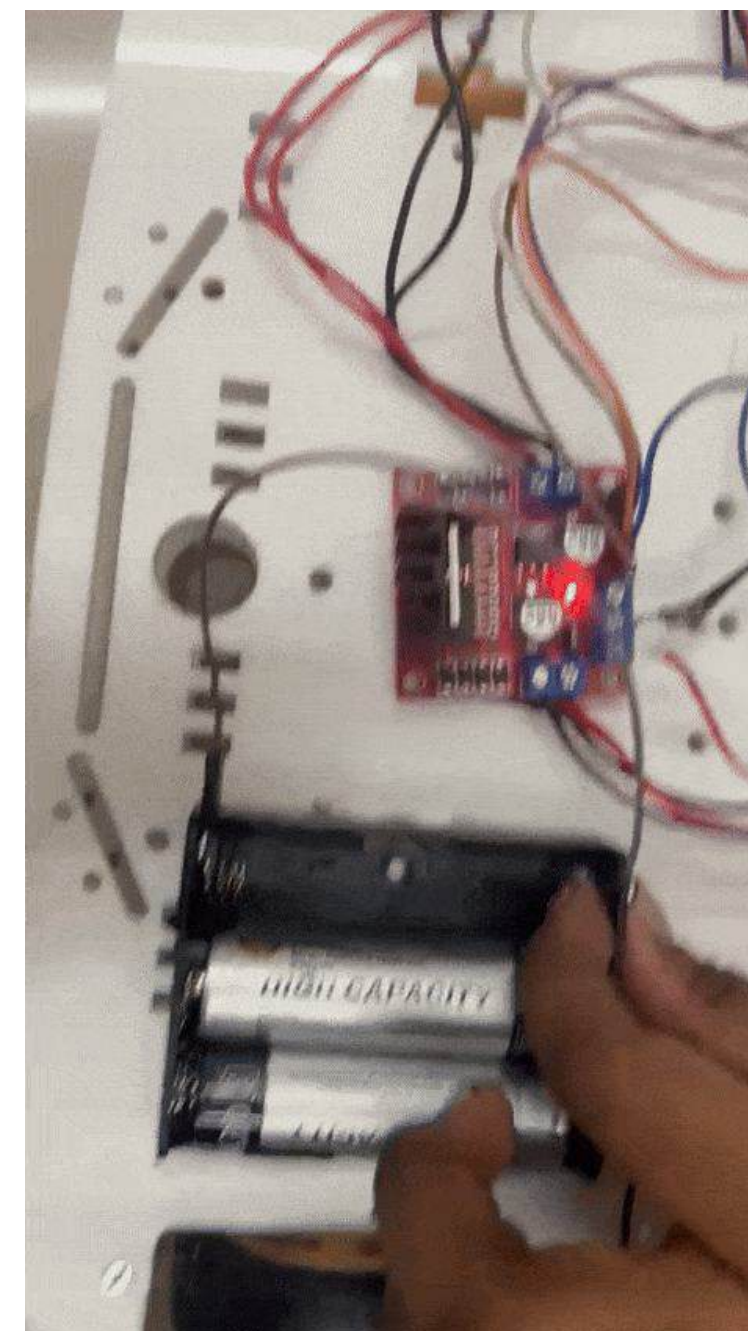
1



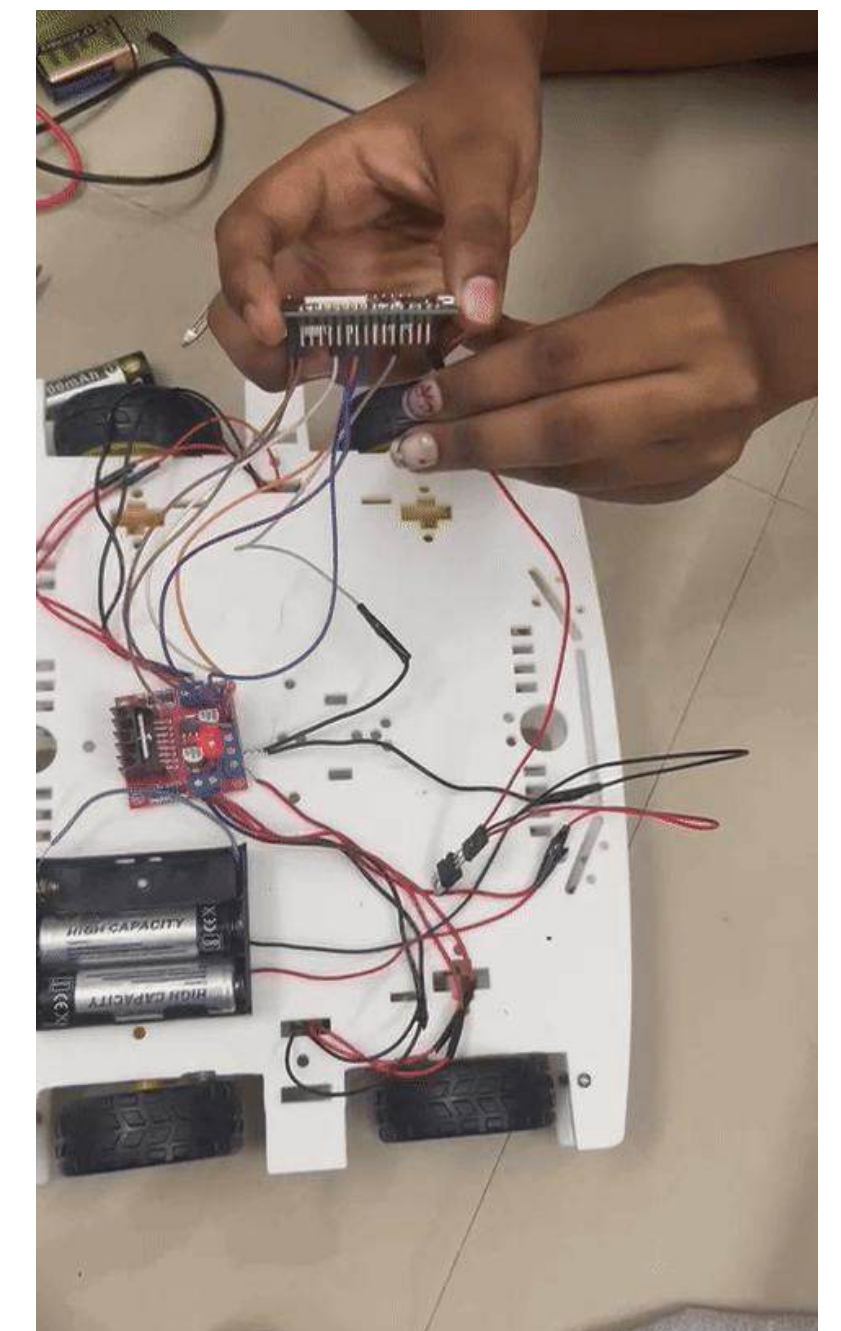
2



3



5



Code for running the motor with esp 32

ESP32 is set up as a Bluetooth device named My_BT_Car.

It listens for single-character commands sent from a phone over Bluetooth.

Commands:

F → Move forward

B → Move backward

L → Turn left (pivot)

R → Turn right (pivot)

S/others → Stop

Two DC motors are controlled via L298N/L293D motor driver.

Motor direction is set with digital pins; speed is controlled using ESP32's hardware PWM channels.

```
#include "BluetoothSerial.h"
BluetoothSerial SerialBT;

// --- Motor pins for ESP32 DevKit V1 ---
int enableRightMotor = 22; // ENA
int rightMotorPin1 = 16; // IN1
int rightMotorPin2 = 17; // IN2

int enableLeftMotor = 23; // ENB
int leftMotorPin1 = 18; // IN3
int leftMotorPin2 = 19; // IN4

#define MAX_MOTOR_SPEED 255
const int PWMFreq = 1000, PWMResolution = 8;
const int rightMotorPWMSpeedChannel = 4, leftMotorPWMSpeedChannel = 5;

void rotateMotor(int rs, int ls) {
  // Right motor direction
  digitalWrite(rightMotorPin1, rs > 0 ? HIGH : LOW);
  digitalWrite(rightMotorPin2, rs < 0 ? HIGH : LOW);
  // Left motor direction
  digitalWrite(leftMotorPin1, ls > 0 ? HIGH : LOW);
  digitalWrite(leftMotorPin2, ls < 0 ? HIGH : LOW);
  // Speed (PWM)
  ledcWrite(rightMotorPWMSpeedChannel, abs(rs));
  ledcWrite(leftMotorPWMSpeedChannel, abs(ls));
}

void setUpPinModes() {
  pinMode(enableRightMotor, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);
  pinMode(enableLeftMotor, OUTPUT);
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);

  ledcSetup(rightMotorPWMSpeedChannel, PWMFreq, PWMResolution);
  ledcSetup(leftMotorPWMSpeedChannel, PWMFreq, PWMResolution);
  ledcAttachPin(enableRightMotor, rightMotorPWMSpeedChannel);
  ledcAttachPin(enableLeftMotor, leftMotorPWMSpeedChannel);
  rotateMotor(0, 0);
}

void setup() {
  Serial.begin(115200);
  setUpPinModes();
  SerialBT.begin("My_BT_Car"); // Pair to this name
  Serial.println("Send F/B/L/R/S over Bluetooth");
}

void loop() {
  if (!SerialBT.available()) return;
  char c = SerialBT.read();
  switch (c) {
    case 'F': rotateMotor(MAX_MOTOR_SPEED, MAX_MOTOR_SPEED); break;
    case 'B': rotateMotor(-MAX_MOTOR_SPEED, -MAX_MOTOR_SPEED); break;
    case 'L': rotateMotor(-MAX_MOTOR_SPEED, MAX_MOTOR_SPEED); break;
    case 'R': rotateMotor(MAX_MOTOR_SPEED, -MAX_MOTOR_SPEED); break;
    default: rotateMotor(0, 0); break; // 'S' or anything else
  }
}
```


Code for running the motor with esp 32 on dabble app

The second code is designed to work with an external controller or custom app that sends structured data (like joystick values, button states, and speed) in a comma-separated format.

The ESP32 simply reads and executes these instructions without deciding the direction itself.

Connections:

```
int enableRightMotor = 22; // ENA
int rightMotorPin1  = 16; // IN1
int rightMotorPin2  = 17; // IN2
```

```
int enableLeftMotor  = 23; // ENB
int leftMotorPin1    = 18; // IN3
int leftMotorPin2    = 19; // IN4
```

Unlike the first code where the ESP32 interprets simple one-letter commands (F, B, L, R, S) and decides movement,

The second code depends on an external controller to determine direction and speed, making it more app-driven.

```
#include <BluetoothSerial.h>

// Create a Bluetooth Serial object
BluetoothSerial SerialBT;

// Motor driver pins (ESP32 DevKit V1 GPIOs)
#define IN1 13 // Motor A
#define IN2 12 // Motor A
#define IN3 14 // Motor B
#define IN4 27 // Motor B
#define ENA 5 // Speed control (PWM) -> L298N ENA/ENB
#define RELAY 4 // Relay output

// PWM setup
const int freq = 30000;
const int pwmChannel = 2;
const int resolution = 8;

// Variables to hold received values
int x = 0, y = 0, swi = 0;
int front = 0, back = 0, left = 0, right = 0, speedVal = 0;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32_Car"); // Bluetooth device name

  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(RELAY, OUTPUT);

  // Setup PWM for motor speed
  ledcSetup(pwmChannel, freq, resolution);
  ledcAttachPin(ENA, pwmChannel);

  Serial.println("ESP32 DevKit V1 Car Ready. Connect via Bluetooth to 'ESP32_Car'");
}

void loop() {
  if (SerialBT.available()) {
    String input = SerialBT.readStringUntil('\n'); // receive data until newline

    // Expected format: x,y,swi,front,back,left,right,speed
    // Example: "0,0,1,0,1,0,0,180"
    sscanf(input.c_str(), "%d,%d,%d,%d,%d,%d,%d,%d",
           &x, &y, &swi, &front, &back, &left, &right, &speedVal);

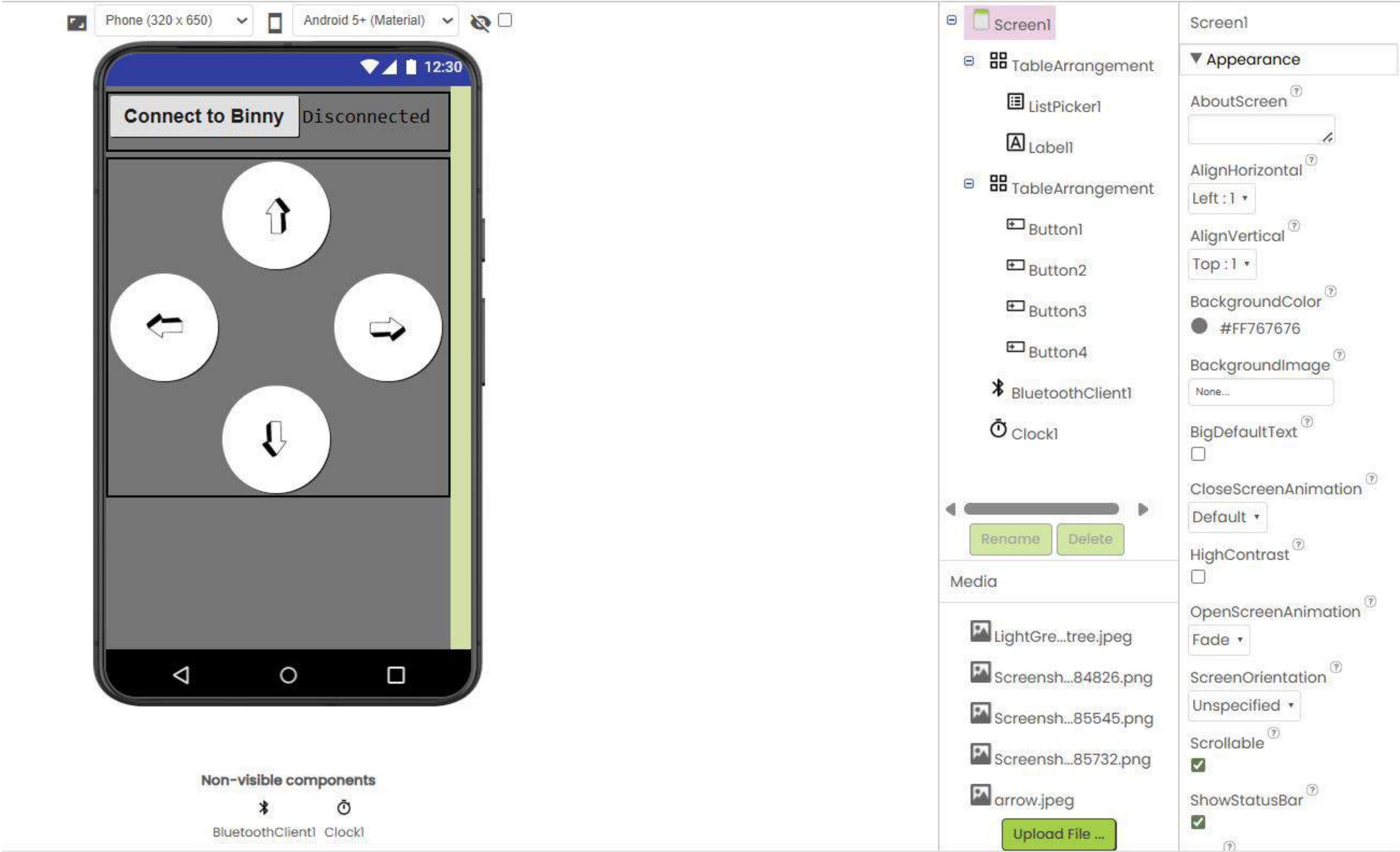
    // Motor speed
    ledcWrite(pwmChannel, speedVal);

    if (back == 1) { // Move backward
      digitalWrite(IN1, HIGH);
      digitalWrite(IN2, LOW);
      digitalWrite(IN3, HIGH);
      digitalWrite(IN4, LOW);
    }
    else if (front == 1) { // Move forward
      digitalWrite(IN1, LOW);
      digitalWrite(IN2, HIGH);
      digitalWrite(IN3, LOW);
      digitalWrite(IN4, HIGH);
    }
    else if (right == 1) { // Turn right
      digitalWrite(IN1, LOW);
      digitalWrite(IN2, HIGH);
      digitalWrite(IN3, HIGH);
      digitalWrite(IN4, LOW);
    }
    else if (left == 1) { // Turn left
      digitalWrite(IN1, HIGH);
      digitalWrite(IN2, LOW);
      digitalWrite(IN3, LOW);
      digitalWrite(IN4, HIGH);
    }
    else { // Stop
      digitalWrite(IN1, LOW);
      digitalWrite(IN2, LOW);
      digitalWrite(IN3, LOW);
      digitalWrite(IN4, LOW);
    }

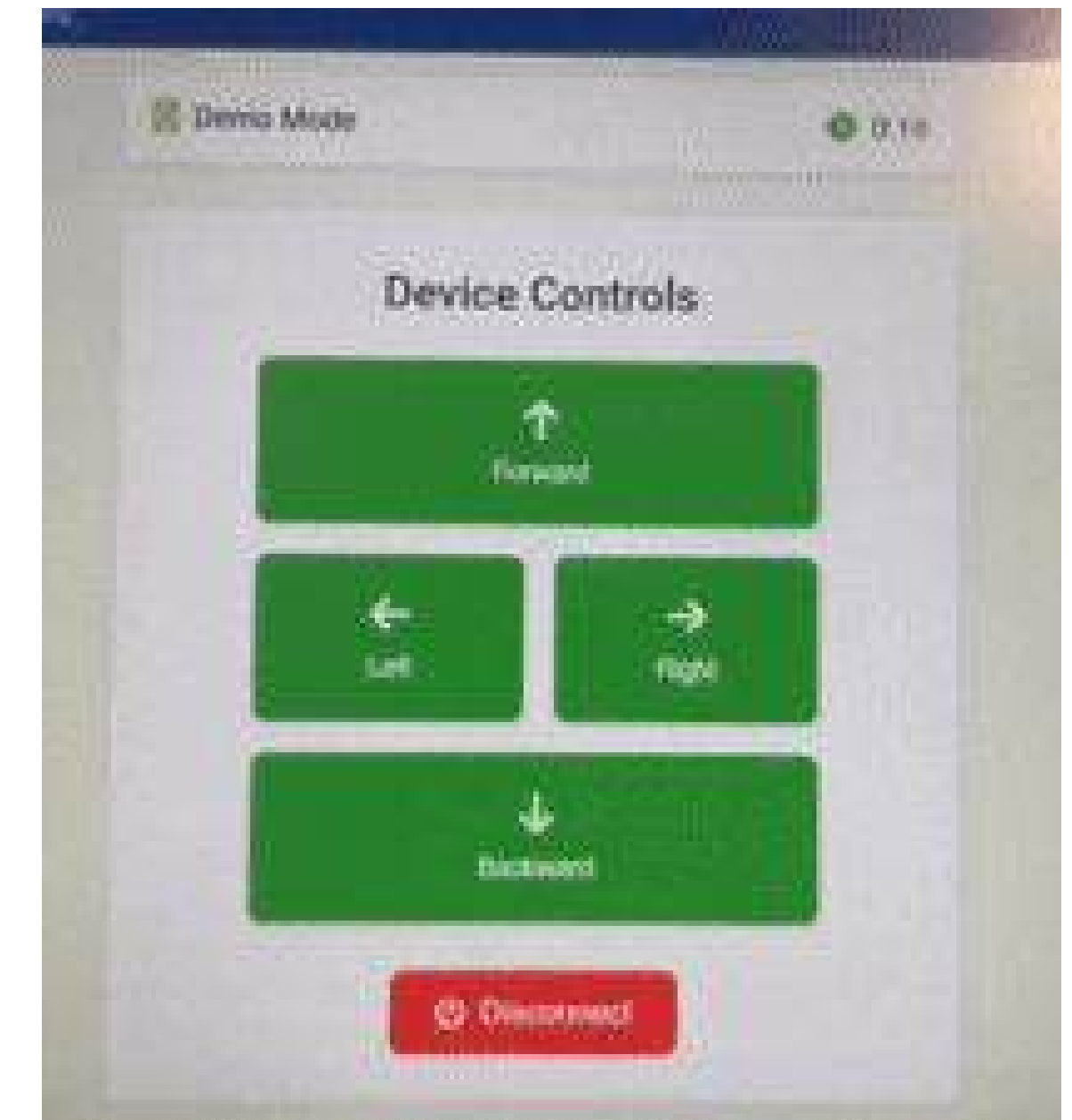
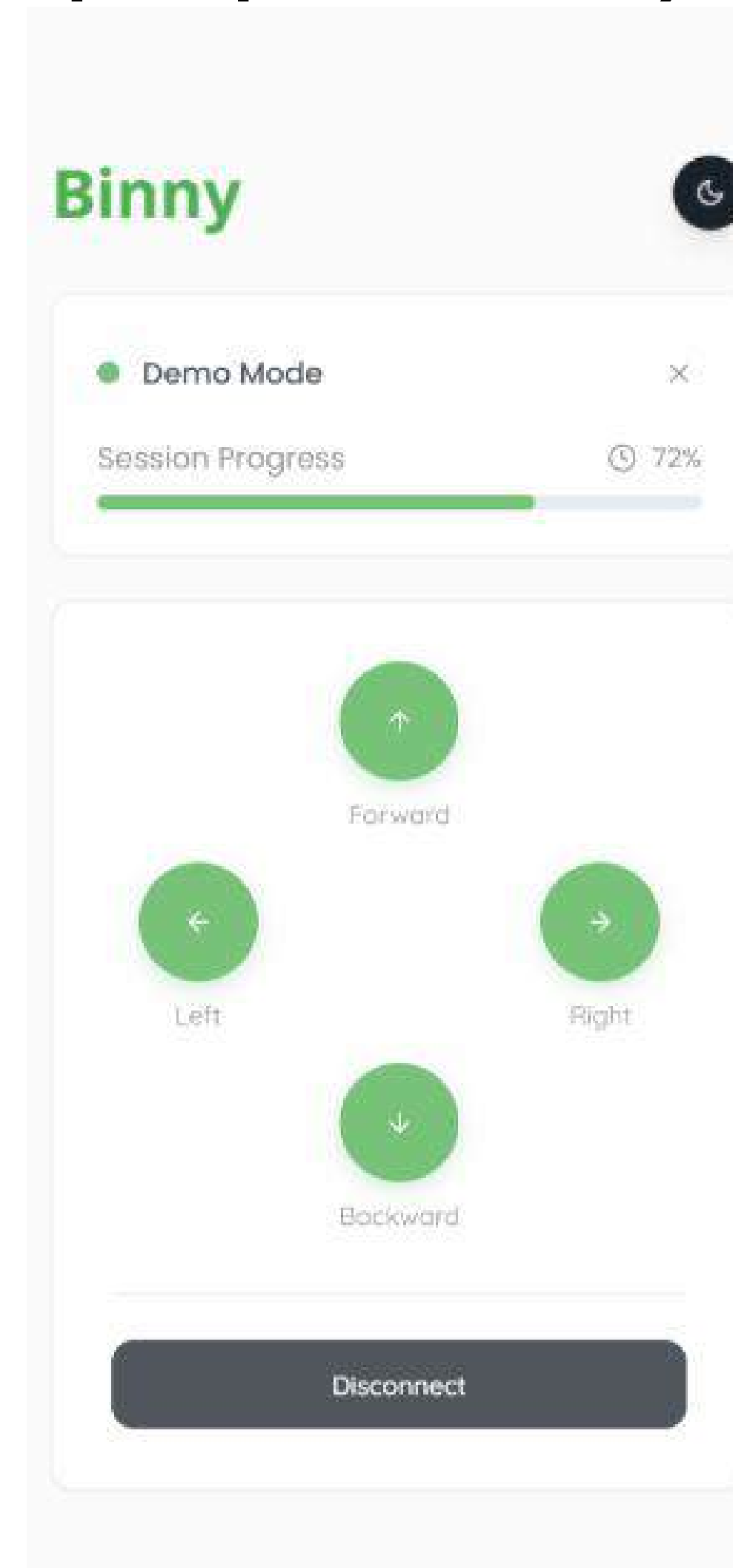
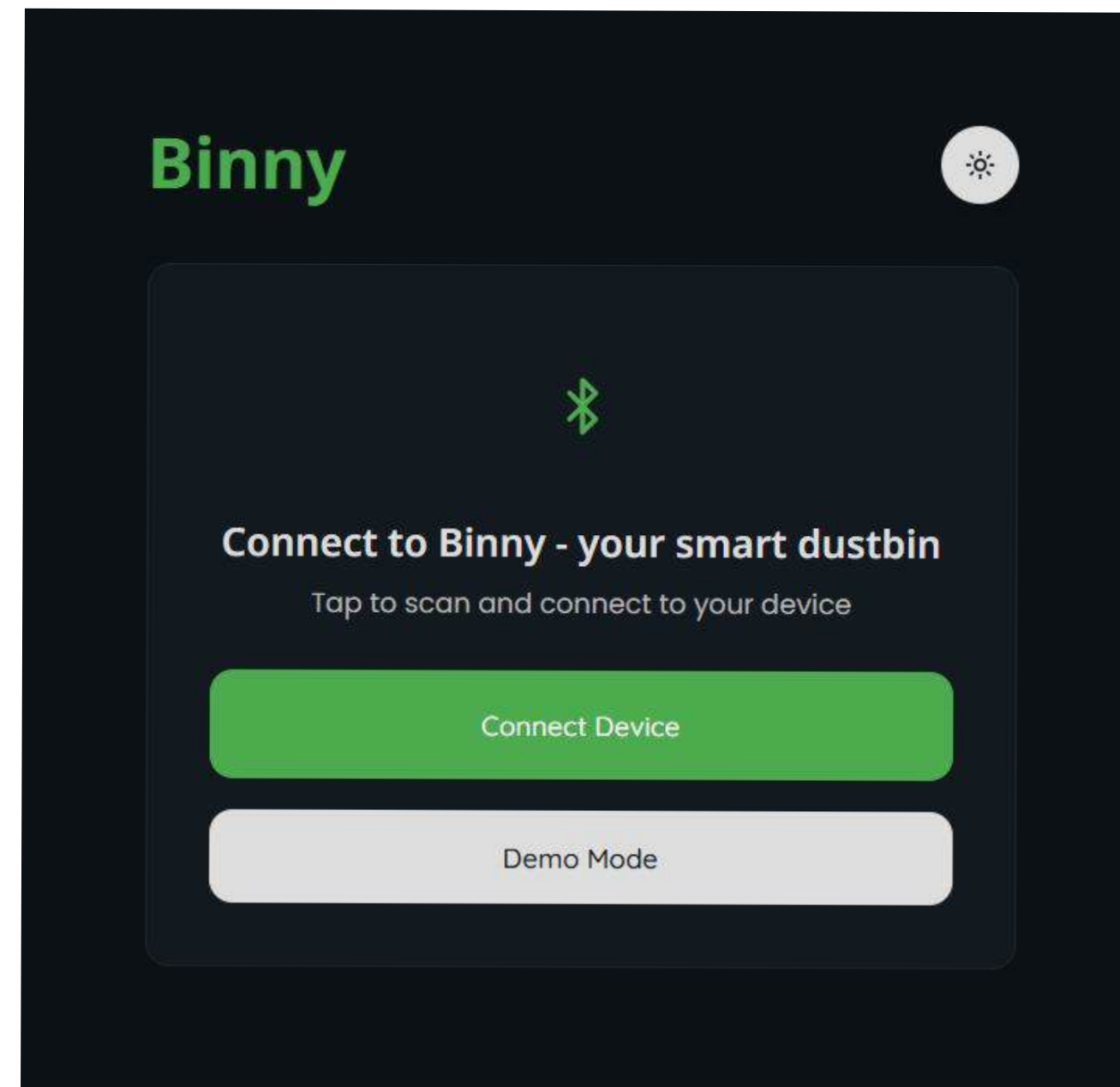
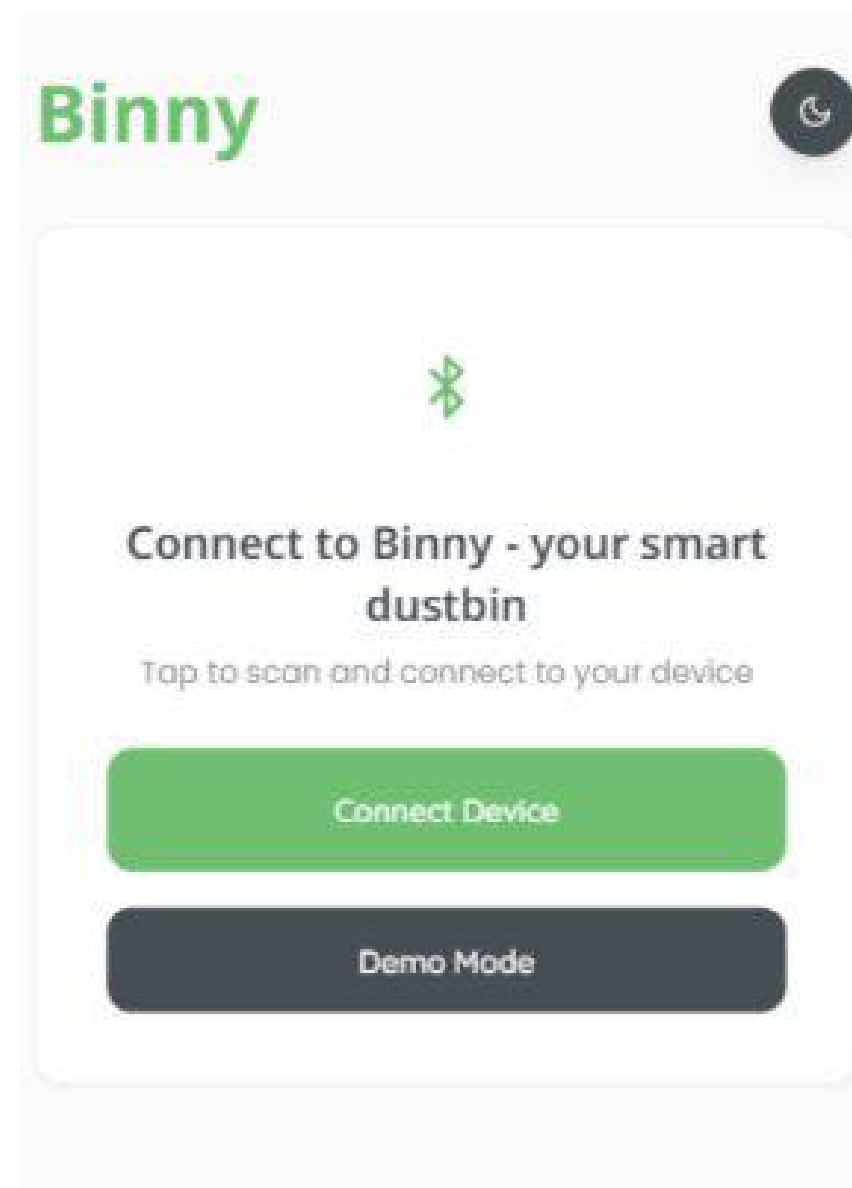
    // Relay control
    if (swi == 1) {
      digitalWrite(RELAY, HIGH);
    } else {
      digitalWrite(RELAY, LOW);
    }

    // Debug print
    Serial.print("X: "); Serial.print(x);
    Serial.print(" Y: "); Serial.print(y);
    Serial.print(" Speed: "); Serial.println(speedVal);
  }
}
```


App for the robot made on mit (for one person)



Website for the robot (can allow multiple people to use)



1. The home screen allows the user to **connect the dustbin to their phone via Bluetooth**.
2. After connecting, the user is taken to the **control page**, which provides **directional buttons and a 1-minute session timer** (Arbitrary time to call dustbin to self).
3. Once the **timer ends**, the **app automatically disconnects**, making the device **available for the next user** through a QR code.
4. A dark mode toggle is provided for user preference.
5. The current Demo Mode is only for testing purposes and will not appear in the final version.
6. **Proper backend code integration** is required to ensure smooth hardware control, session handling, and safety.

FACTORS TO CONSIDER FOR REAL LIFE IMPLEMENTATION

Phase 1: Prototype Overview

Aim: building a working prototype.

- The dustbin lid opens automatically.
- The bin can be driven like a robot using a smartphone app (MIT App Inventor).

This phase proved that both the mechanical movement and the basic user control are possible.

Phase 2: Multi user interaction

Aim: Smooth cafeteria use with multiple people.

- QR Codes on tables → instant access to the control app.
- Control Sessions:
 - One user at a time.
 - Auto-disconnect after 1–2 minutes or 20–30s inactivity.
 - Prompt to release bin early when done.
- Busy Message: If another user tries, show “I’m hard at work right now! Please try again shortly.”
- Gamification: Fun messages like “Bin on the way!” or “Mission complete — thanks for helping!”

Phase 3: Working Practical model

- Practical Needs:
- Rechargeable Li-ion battery with USB-C/docking station.
- Motors strong enough to move with 3–5 kg trash.
- Easy inner bin removal for staff cleaning.