VOLTROB 2024

PROJECT REPORT

PRESENTATION

TEAM NAME :SVPS

Team Members

1] Shushrutha N Gowda
2]Vismaya TV
3]Pushpa R
4] Sourav Kumar

Overview

Objective:

- Manages irrigation and pest control in agricultural settings.
- Uses real-time data to optimize water usage and minimize pesticide application.
- Ensures healthy crop growth and promotes environmental sustainability.

Stage 1: Irrigation and Pest Control System:(Resource Optimization and Pesticide Reduction)

PIR Sensor with Fresnel Lens: Detects pest motion; triggers water spray.

Soil Moisture Sensor: Monitors soil moisture levels; activates water pump for irrigation when moisture is low.

Buzzer: Activates when pests are detected to repel them away.

Water Pump and Motor Driver: Controls water and pesticide spray based on sensor inputs.

DHT11 Sensor: Measures temperature and humidity; provides environmental data to optimize irrigation.

Stage 2: **Solar Tracker:(Energy Efficiency and Environmental Sustainability)**

LDR (Light Dependent Resistor): Detects sunlight intensity; determines optimal solar panel angle.

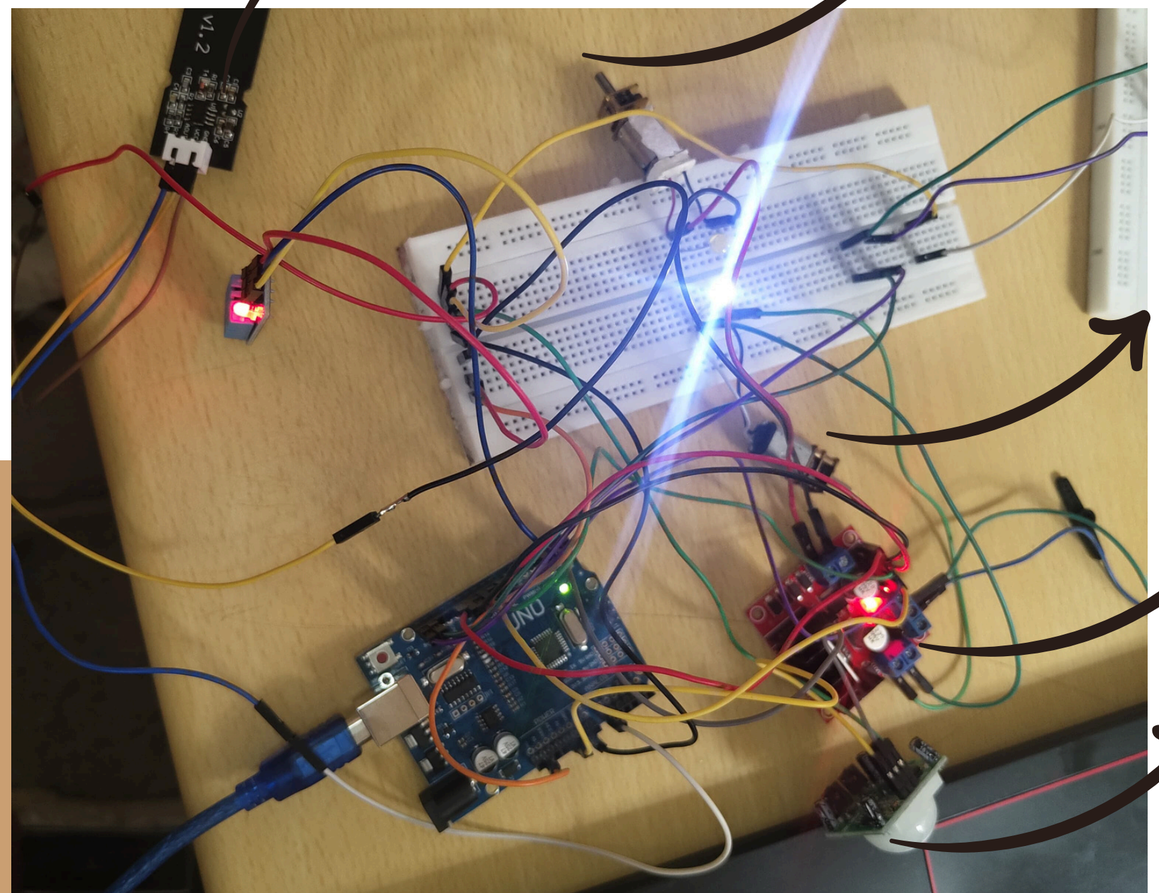
Servo Motor: Adjusts the solar panel's position for maximum sunlight exposure.

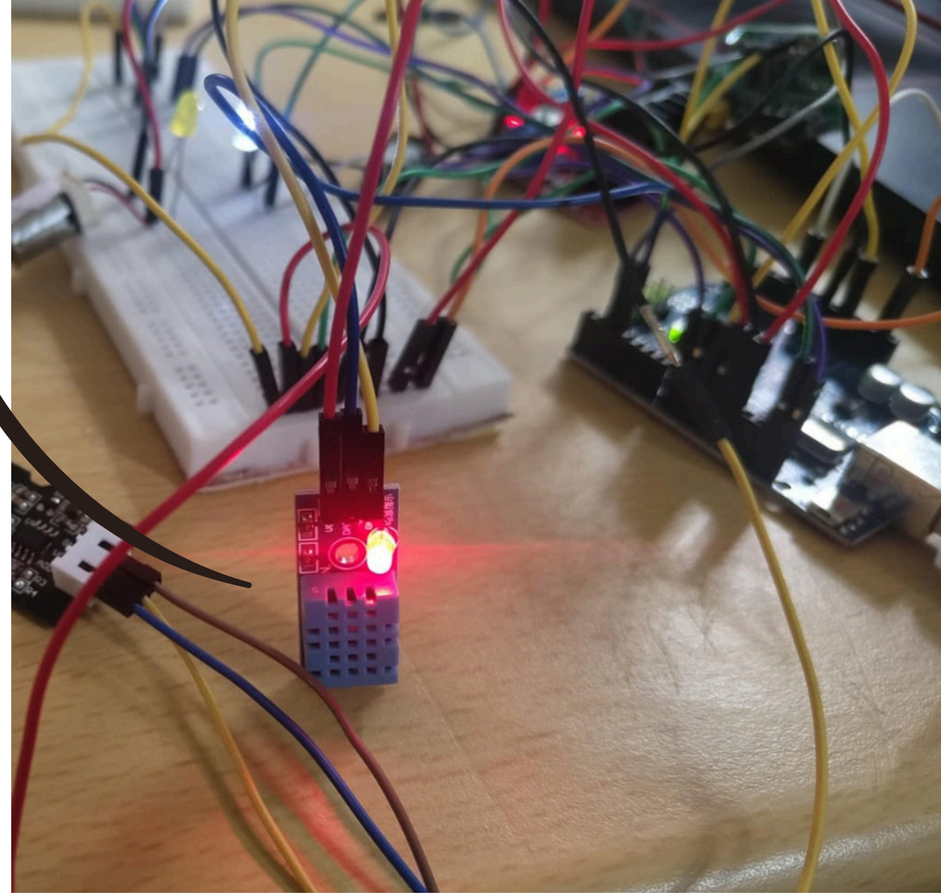
Stage 3: I2C COMMUNICATION:(System Synchronization and Coordination)

I2C Communication: Establishes communication between the two Arduinos to synchronize operations between the irrigation and pest control system, the DHT11 environmental monitoring, and the solar tracking system.

STAGE1:

Irrigation and Pest Control System

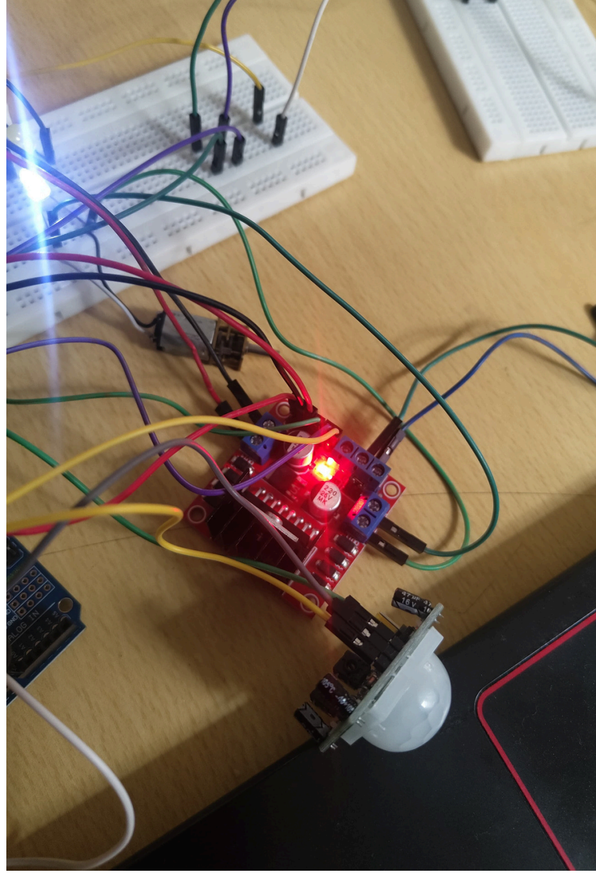




STAGE1

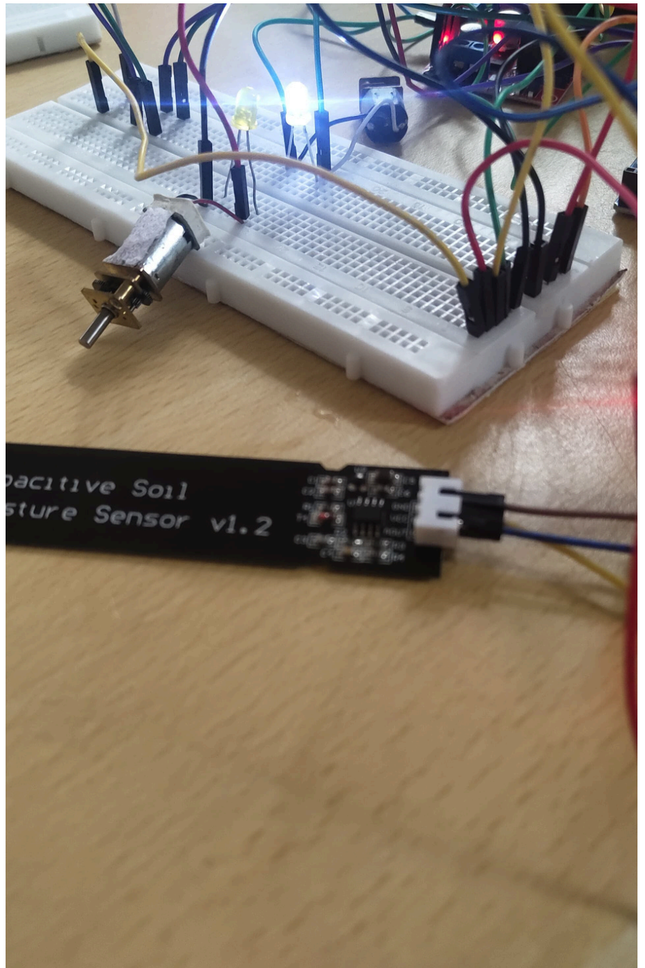
PIR Sensor with Fresnel Lens:

- **Overview:** The Passive Infrared (PIR) sensor is used to detect motion by measuring infrared radiation levels emitted by objects. The Fresnel lens is a special lens added to the sensor to improve its range and focus.
- **Function:** In this system, the PIR sensor is responsible for detecting the presence of pests. When a pest moves within the sensor's detection area, the PIR sensor identifies the change in infrared radiation and sends a signal to the Arduino. This signal triggers the water pump to spray pesticide, which helps in deterring pests from the area.



Soil Moisture Sensor:

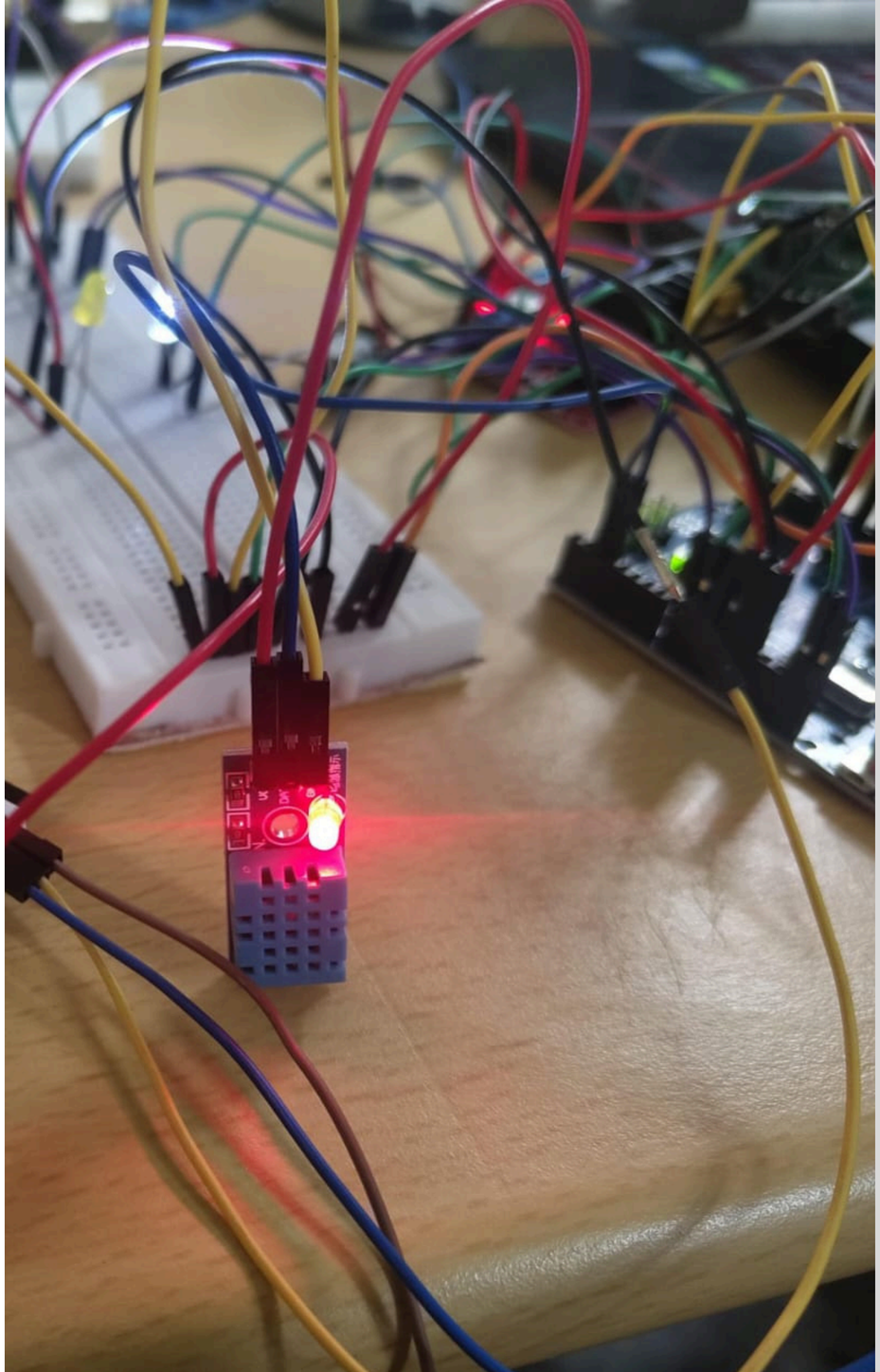
- **Overview:** The soil moisture sensor measures the water content in the soil. It typically provides an analog output, which can be read by the Arduino to determine the soil's moisture level.
- **Function:** The Arduino continuously monitors the moisture level in the soil through the sensor. If the moisture level falls below a predefined threshold, indicating that the soil is dry, the Arduino activates the water pump to irrigate the soil. This ensures that the plants receive adequate water, maintaining optimal soil conditions.



STAGE1

DHT11 Sensor:

- **Overview:** The DHT11 sensor is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and outputs a digital signal on the data pin (no analog input pins needed).
- **Function:** In this system, the DHT11 sensor monitors the environmental conditions by measuring the temperature and humidity levels. This data can be used to make more informed decisions about irrigation, such as adjusting the watering schedule based on current weather conditions, thus optimizing water usage.



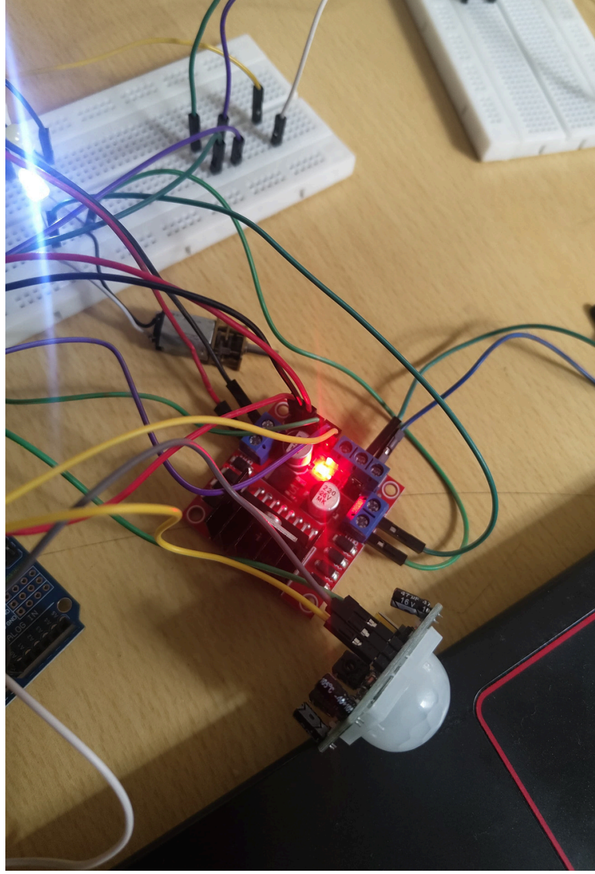
STAGE1

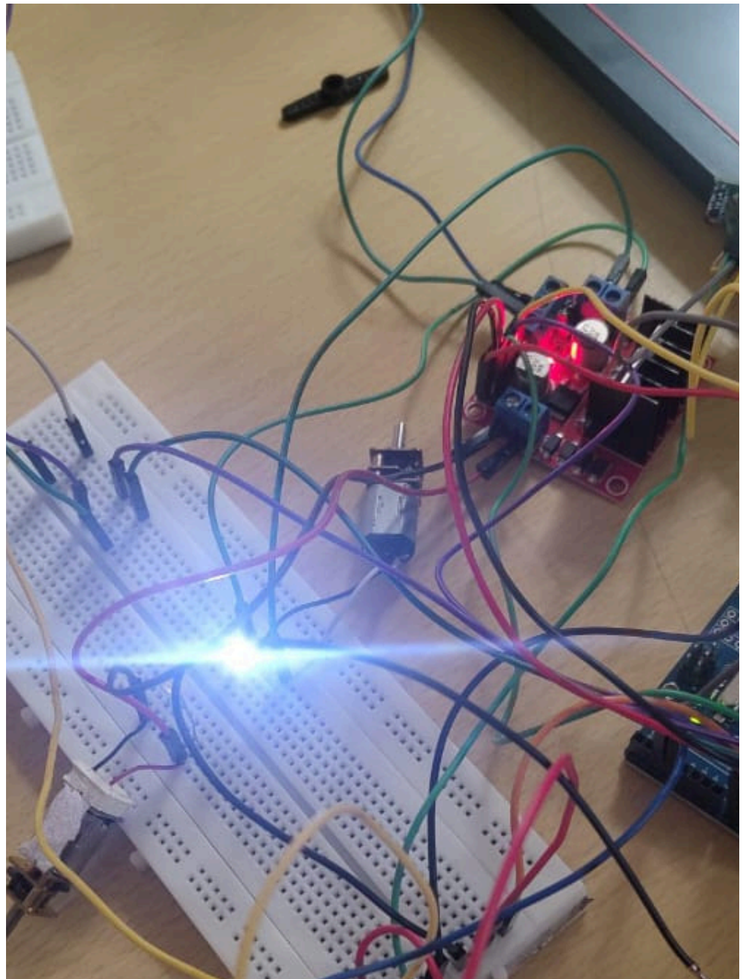
Motor Driver:

- **Overview:** The motor driver acts as an interface between the Arduino and the water pump. It allows the Arduino to control the water pump, which usually requires more current than the microcontroller can supply directly.
- **Function:** The motor driver receives signals from the Arduino and powers the water pump accordingly. This enables the Arduino to control the irrigation process efficiently and ensures the water pump operates when needed

Water Pump:

- **Overview:** The water pump is a device that moves water or pesticide from a source (such as a tank) to the target area (the field or garden). It is crucial for the irrigation process.
- **Function:** The pump is activated by the Arduino whenever the soil moisture sensor detects that the soil is dry or when the PIR sensor detects pests. The pump then sprays water/pesticide over the plants, either to irrigate them or to deter pests.





Working(Code):

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

int pirPin = 2;  // Pin for PIR motion sensor
int irrigationMotorIn1 = 8;  // Motor driver input 1 for irrigation motor
int irrigationMotorIn2 = 9;  // Motor driver input 2 for irrigation motor
int irrigationMotorEn = 10;  // Motor driver enable pin for irrigation motor
int pestControlMotorIn1 = 11;  // Motor driver input 1 for pest control motor
int pestControlMotorIn2 = 12;  // Motor driver input 2 for pest control motor
int pestControlMotorEn = 13;  // Motor driver enable pin for pest control motor
int soilMoisturePin = A0;  // Pin for soil moisture sensor

int moistureThreshold = 600;  // Threshold for soil moisture level
const int slaveAddress = 8;  // I2C address of the slave

#define DHTPIN 3         // Pin connected to the DHT sensor
#define DHTTYPE DHT11     // Type of DHT sensor (DHT11)
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  pinMode(pirPin, INPUT);  // Set PIR pin as input
  pinMode(irrigationMotorIn1, OUTPUT);  // Set irrigation motor driver input 1 as output
  pinMode(irrigationMotorIn2, OUTPUT);  // Set irrigation motor driver input 2 as output
  pinMode(irrigationMotorEn, OUTPUT);  // Set irrigation motor driver enable as output
  pinMode(pestControlMotorIn1, OUTPUT);  // Set pest control motor driver input 1 as output
  pinMode(pestControlMotorIn2, OUTPUT);  // Set pest control motor driver input 2 as output
  pinMode(pestControlMotorEn, OUTPUT);  // Set pest control motor driver enable as output
```

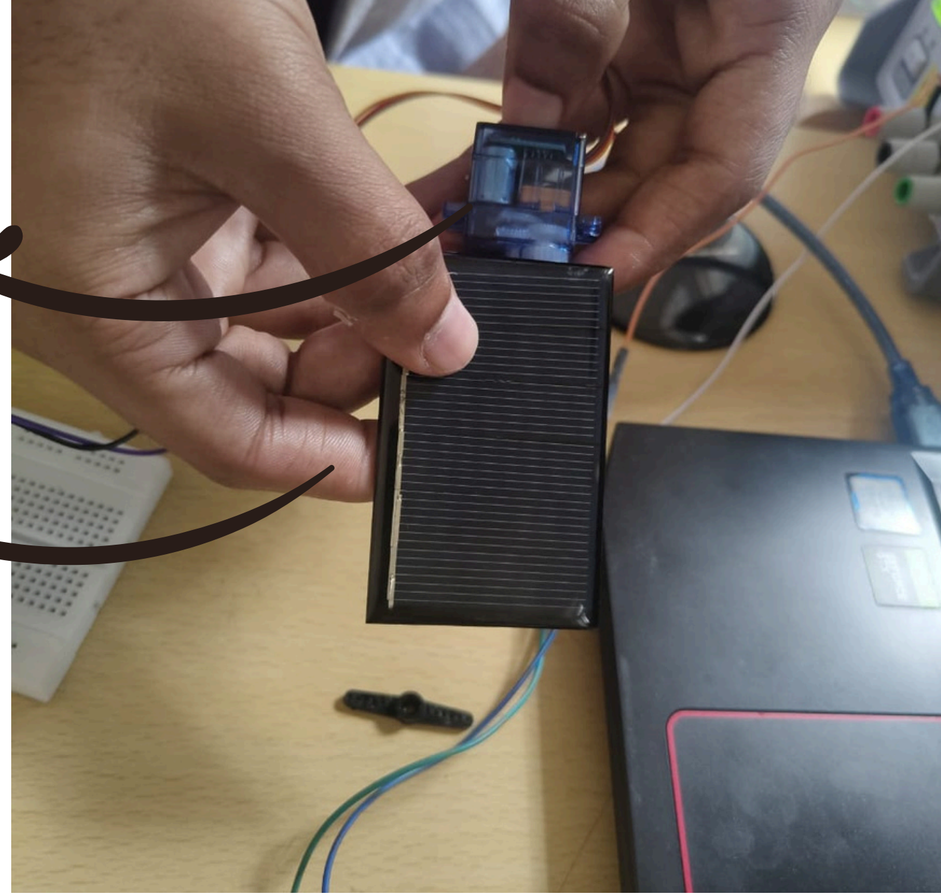
```
Serial.begin(9600);  // Initialize serial communication for debugging
  Wire.begin();        // Join the I2C bus as master

  dht.begin();         // Initialize DHT sensor
}

void loop() {
  int motionDetected = digitalRead(pirPin);  // Read motion detection from PIR sensor
  int soilMoistureValue = analogRead(soilMoisturePin);  // Read soil moisture level
  float temperature = dht.readTemperature(); // Read temperature from DHT sensor
  float humidity = dht.readHumidity();       // Read humidity from DHT sensor

  Serial.print("Soil Moisture: ");
  Serial.println(soilMoistureValue);
  Serial.print("Temperature: ");
  Serial.println(temperature);
  Serial.print("Humidity: ");
  Serial.println(humidity);

  // Adjust moisture threshold based on humidity
  int adjustedMoistureThreshold = moistureThreshold;

  if (humidity > 80) {
    adjustedMoistureThreshold -= 50;  // Reduce threshold to avoid overwatering in high humidity
  }
```

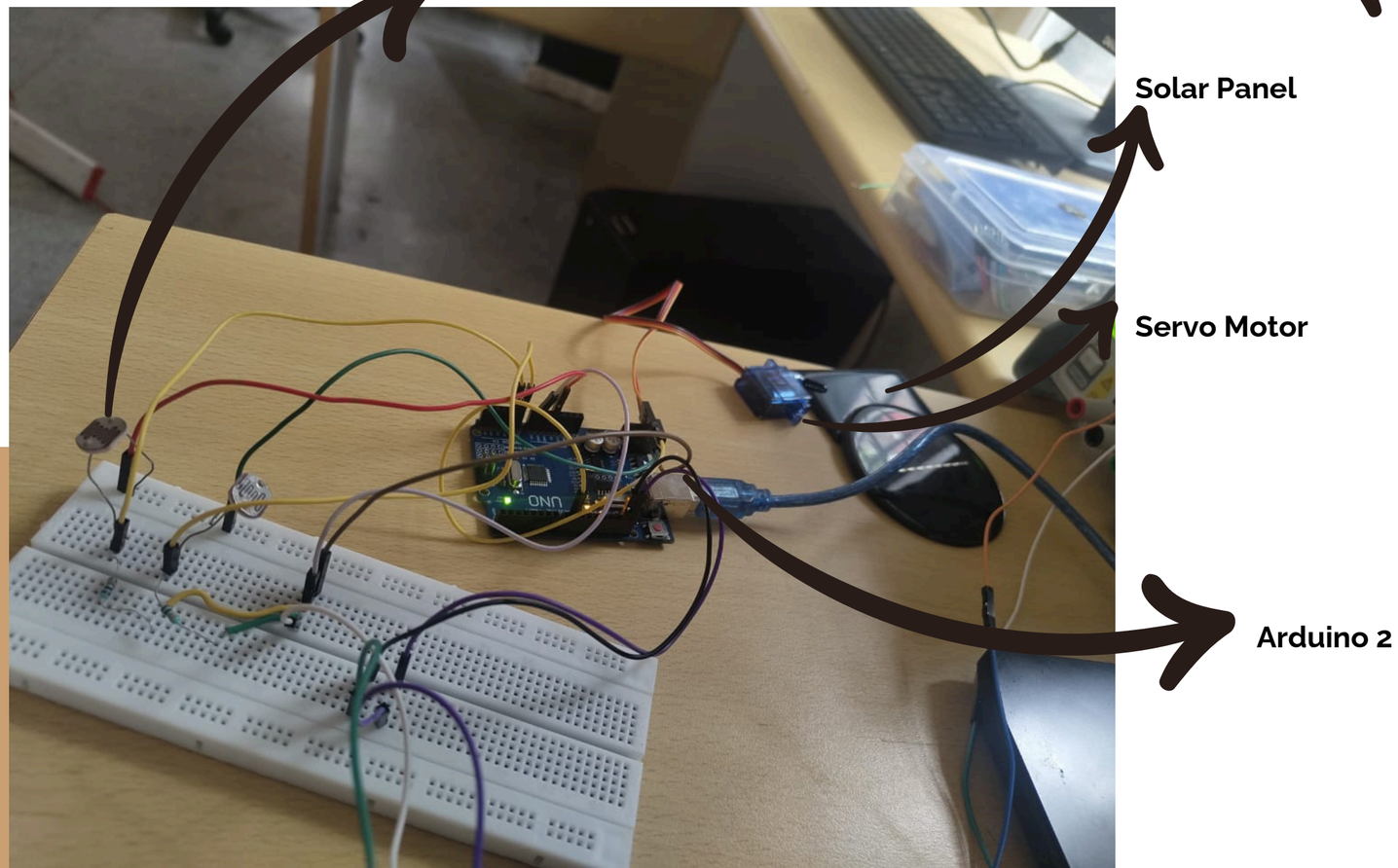
```
// Control irrigation motor based on soil moisture
  if (soilMoistureValue < adjustedMoistureThreshold) {
    Serial.println("Soil Moisture Low. Irrigating...");
    startIrrigationMotor();  // Start irrigation motor
    sendI2CSignal(1);  // Signal to slave Arduino: Irrigation ON
  } else {
    Serial.println("Soil Moisture Sufficient. Irrigation OFF");
    stopIrrigationMotor();  // Stop irrigation motor
    sendI2CSignal(0);  // Signal to slave Arduino: Irrigation OFF
  }

  // Control pest control motor based on motion detection
  if (motionDetected == HIGH) {
    Serial.println("Motion Detected! Activating Pest Control...");
    startPestControlMotor();  // Start pest control motor
    sendI2CSignal(2);  // Signal to slave Arduino: Pest Control ON
  } else {
    Serial.println("No Motion Detected. Pest Control OFF");
    stopPestControlMotor();  // Stop pest control motor
    sendI2CSignal(0);  // Signal to slave Arduino: Pest Control OFF
  }

  delay(500);  // Wait before next loop iteration
}
void startIrrigationMotor() {
  digitalWrite(irrigationMotorIn1, HIGH);  // Set irrigation motor direction
  digitalWrite(irrigationMotorIn2, LOW);
  analogWrite(irrigationMotorEn, 255);  // Set irrigation motor speed to full
}
```

```
void stopIrrigationMotor() {
  digitalWrite(irrigationMotorIn1, LOW);  // Stop irrigation motor
  digitalWrite(irrigationMotorIn2, LOW);
  analogWrite(irrigationMotorEn, 0);  // Set motor speed to zero
}

void startPestControlMotor() {
  digitalWrite(pestControlMotorIn1, HIGH);  // Set pest control motor direction
  digitalWrite(pestControlMotorIn2, LOW);
  analogWrite(pestControlMotorEn, 255);  // Set pest control motor speed to full
}

void stopPestControlMotor() {
  digitalWrite(pestControlMotorIn1, LOW);  // Stop pest control motor
  digitalWrite(pestControlMotorIn2, LOW);
  analogWrite(pestControlMotorEn, 0);  // Set motor speed to zero
}

void sendI2CSignal(int signal) {
  Wire.beginTransmission(slaveAddress);
  Wire.write(signal);  // Send signal to slave Arduino (1 = Irrigation ON, 2 = Pest Control ON, 0 = OFF)
  Wire.endTransmission();
}
```

STAGE2:

Solar Tracking System

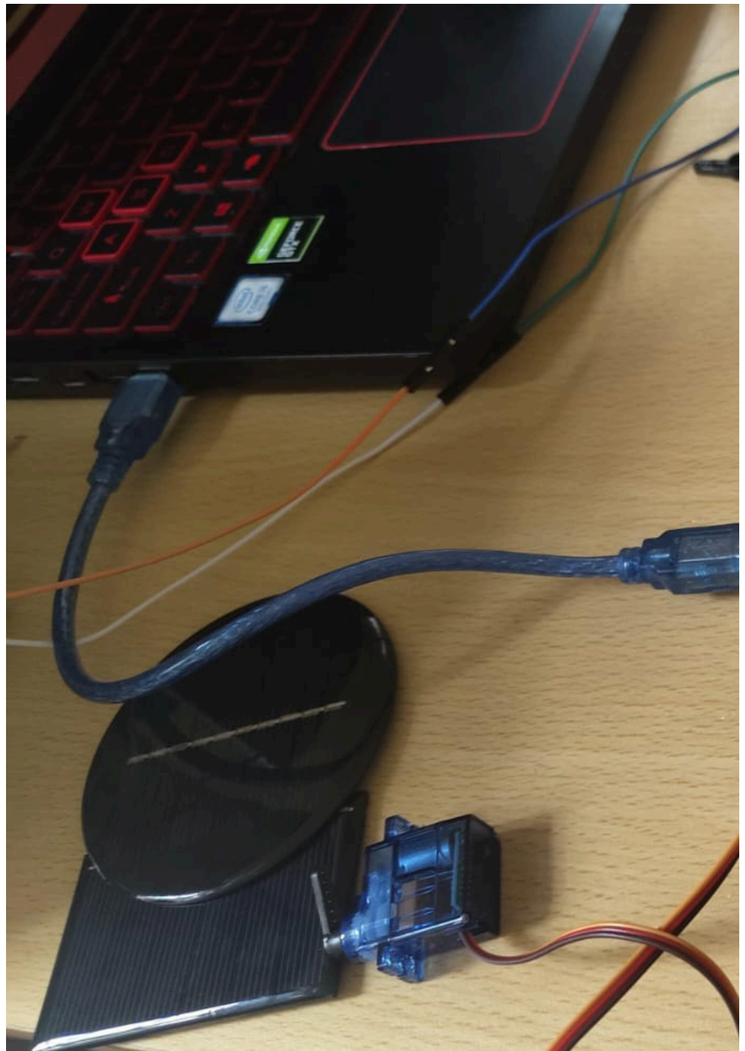


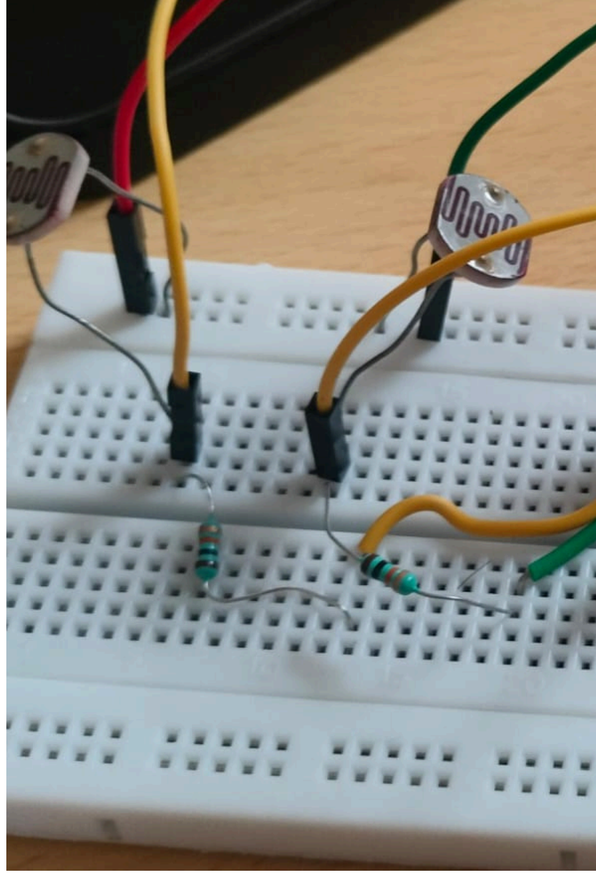


STAGE2

LDR (Light Dependent Resistor):

- **Overview:** The Light Dependent Resistor (LDR) changes its resistance based on the light intensity falling on it. Higher light intensity results in lower resistance.
- **Function:** In this system, multiple LDRs are strategically placed to detect sunlight intensity from different directions. The Arduino reads the resistance values from these LDRs to determine the direction of maximum sunlight. This data is used to adjust the position of the solar panel, ensuring it is always oriented towards the sun.





Servo Motor:

- **Overview:** A servo motor is an actuator that allows for precise control over angular position. It receives control signals from the Arduino to adjust its position accordingly.
- **Function:** The current servo motor moves the solar panel horizontally in a circular manner. This movement allows the panel to track the sun's position across the sky during the day, ensuring that it remains aligned with the direction of maximum sunlight. Although the panel isn't currently used for energy generation, this movement optimizes its future potential for energy capture.

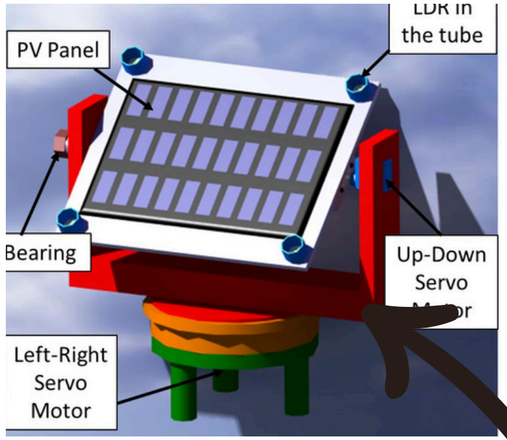
STAGE2

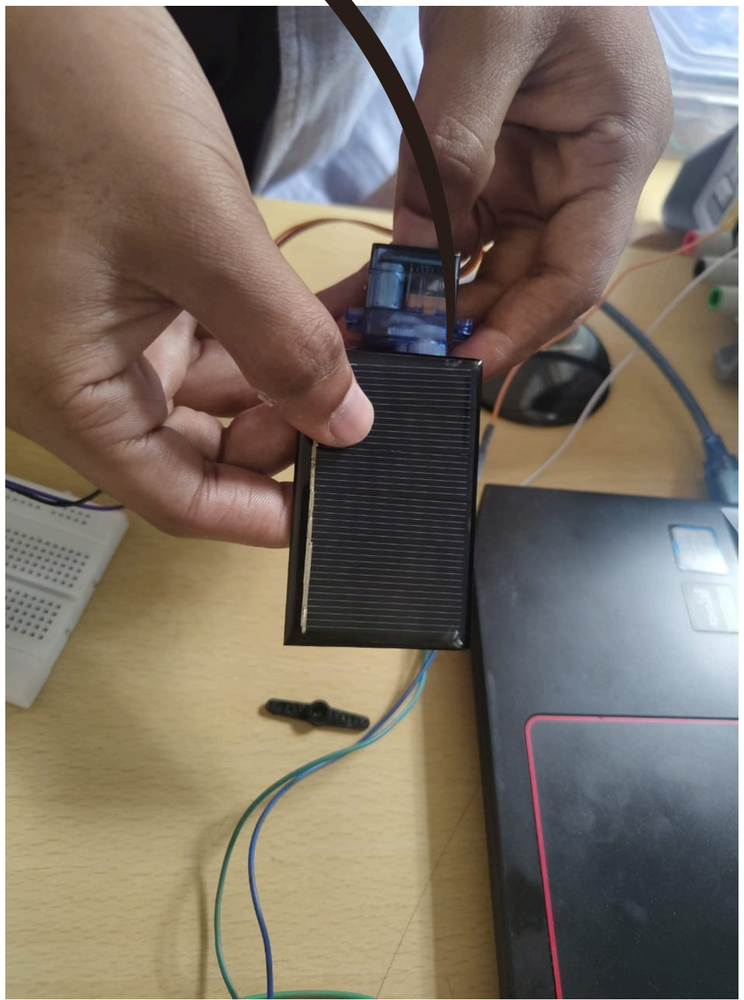
Buzzer:

- **Overview:** The buzzer is an electronic device that produces sound when an electrical signal is applied. It's commonly used for alerting or signaling purposes.
- **Function:** In this project, the buzzer is activated when the PIR sensor detects the presence of pests. The loud noise generated by the buzzer serves as an additional deterrent, scaring away pests that might harm the crops.

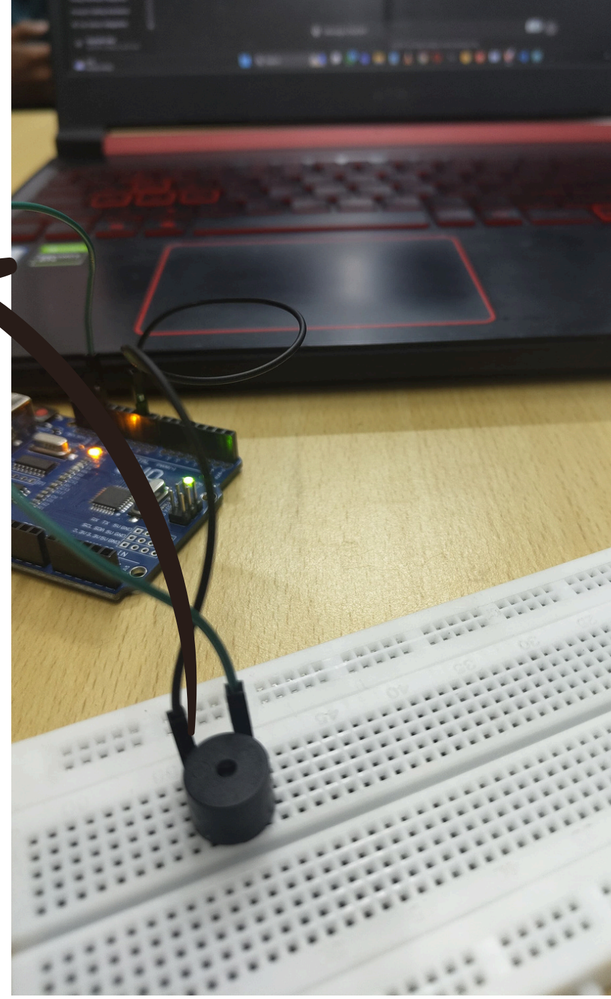
Solar Panel:

- **Overview:** The solar panel in this system is currently not used to derive energy but is included as a future implementation idea. It consists of photovoltaic cells that can generate electrical energy when exposed to sunlight.
- **Function:** Though the solar panel is not yet being used for energy generation, the system is designed to ensure that it remains oriented towards the sun throughout the day. This sets the foundation for future integration, where the panel can be used to generate and store energy.





Buzzer(40khz))



Working(Code):

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

Servo servoHorizontal;  // Horizontal servo

int ldrLeft = A0;  // Left LDR
int ldrRight = A1; // Right LDR

const int slaveAddress = 8;  // I2C address of this slave

int receivedData = 0;

// Pin for ultrasonic buzzer
int ultrasonicBuzzerPin = 7;  // Pin connected to the ultrasonic buzzer

void setup() {
  servoHorizontal.attach(9); // Attach the servo to pin 9
  servoHorizontal.write(90); // Start at the center position
  Serial.begin(9600);  // Initialize serial communication for debugging
  Wire.begin(slaveAddress);  // Join I2C bus as slave with address 8
  Wire.onReceive(receiveEvent);  // Register event function to handle incoming data

  pinMode(ultrasonicBuzzerPin, OUTPUT);  // Set buzzer pin as output
}
void loop() {
  if (receivedData == 1) {
    // Solar Tracking Activated
    int leftValue = analogRead(ldrLeft);   // Read the left LDR
    int rightValue = analogRead(ldrRight); // Read the right LDR
```

```
 // Compare the LDR values and adjust the servo position
    if (leftValue > rightValue + 10) {
      servoHorizontal.write(servoHorizontal.read() - 1); // Move servo to the left
    } else if (rightValue > leftValue + 10) {
      servoHorizontal.write(servoHorizontal.read() + 1); // Move servo to the right
    }

    delay(100);  // Small delay to stabilize the movement
  } else if (receivedData == 0) {
    // Motor is OFF, no tracking
    Serial.println("Solar Tracking Paused");
  }

  if (receivedData == 2) {
    // Activate ultrasonic buzzer to repel pests
    Serial.println("Activating Pest Control - Ultrasonic Buzzer");
    tone(ultrasonicBuzzerPin, 40000); // Emit a high-frequency sound (e.g., 40 kHz)
  } else if (receivedData != 2) {
    // Deactivate ultrasonic buzzer
    Serial.println("Deactivating Pest Control - Ultrasonic Buzzer");
    noTone(ultrasonicBuzzerPin); // Stop emitting sound
  }
}
void receiveEvent(int howMany) {
  if (Wire.available() > 0) {
    receivedData = Wire.read();  // Read the data from the master
    Serial.print("Received: ");  // Debug line
    Serial.println(receivedData); // Debug line
  }
}
```

STAGE3: I2C Communication Setup

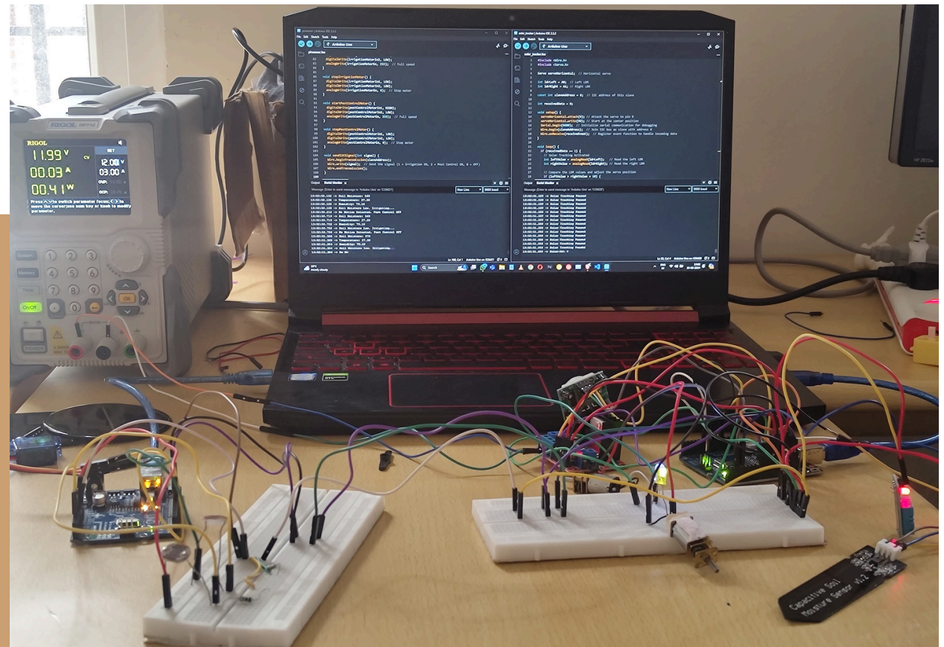
Objective: Establish communication between the two Arduinos to synchronize operations across the irrigation and pest control system, the DHT11 environmental monitoring, and the solar tracking system.

Overview

In Stage 3, we set up I2C communication to enable efficient data exchange and coordination between the two Arduinos used in the project. This setup allows the master Arduino to manage the irrigation and pest control systems, environmental monitoring via the DHT11 sensor, and the solar tracking system, while the slave Arduino assists in executing commands and providing feedback.

Components Involved

1. Master Arduino:
   - Manages sensors (PIR sensor, soil moisture sensor, DHT11) and actuators (irrigation motor, pest control motor).
   - Sends control signals to the slave Arduino to activate or deactivate components based on real-time data.
2. Slave Arduino:
   - Receives control signals from the master Arduino.
   - Executes the commands by controlling motors and other actuators.



Functionality

- Synchronization: Ensures that the master and slave Arduinos work together to maintain optimal operation of the irrigation system, pest control mechanisms, and solar tracking system.
- Data Exchange: Facilitates real-time communication between the Arduinos to synchronize actions based on sensor inputs and system status.
- Control Signals: Allows the master Arduino to send commands to the slave Arduino, indicating whether to activate or deactivate specific system components.

Benefits

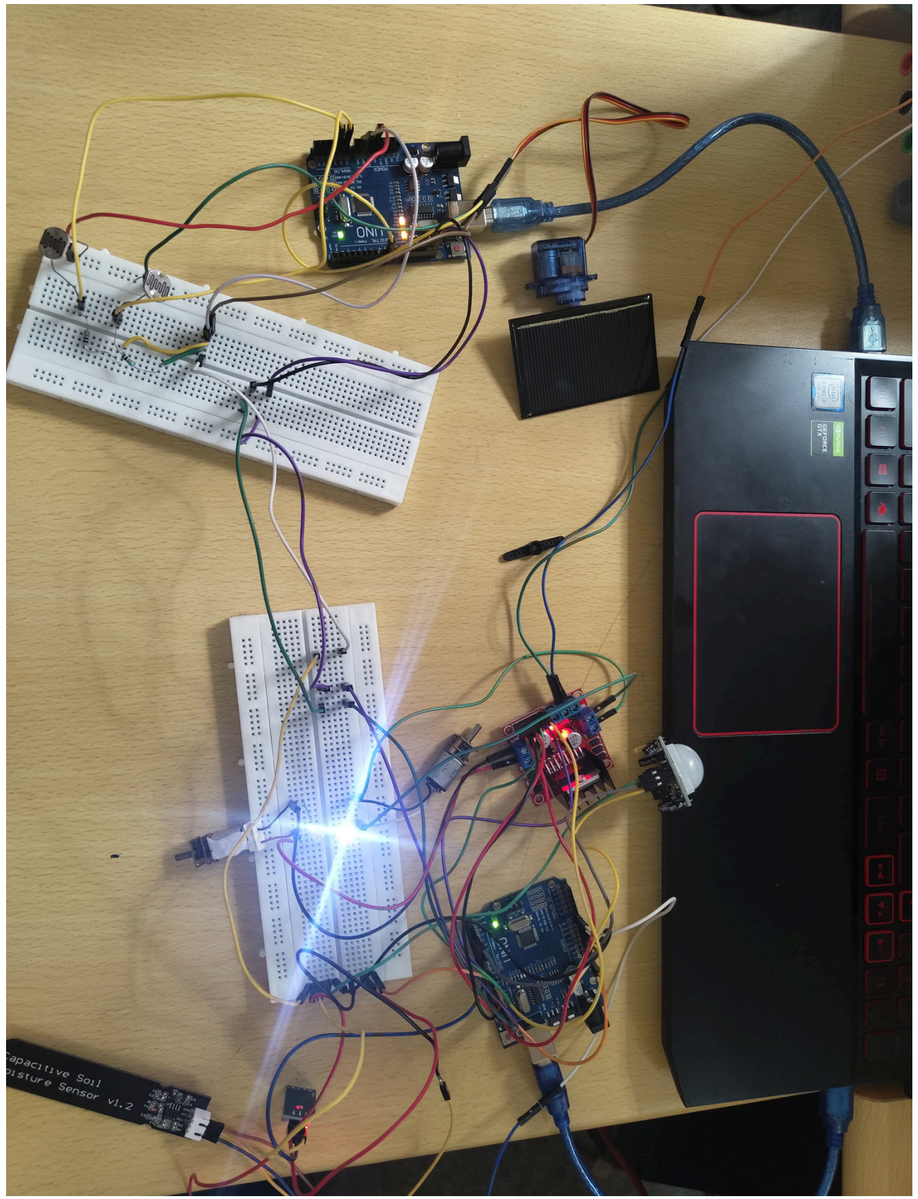
- Coordinated Operation: I2C communication enables precise control and coordination of different system components, enhancing overall efficiency and performance.
- Simplified Wiring: Reduces the complexity of connections between the Arduinos, making the system easier to manage and expand.
- Real-Time Response: Provides the ability to respond to changing conditions and sensor data promptly, ensuring the system operates effective

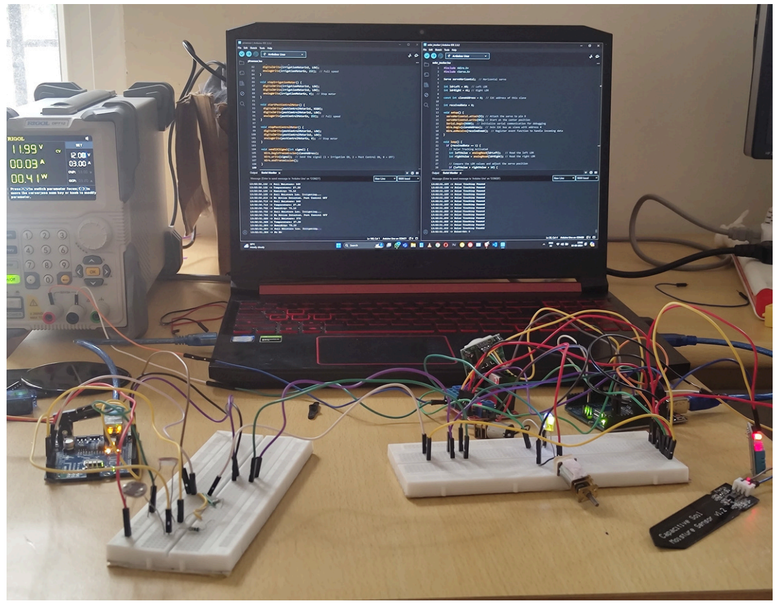
PROBLEMS FACED:

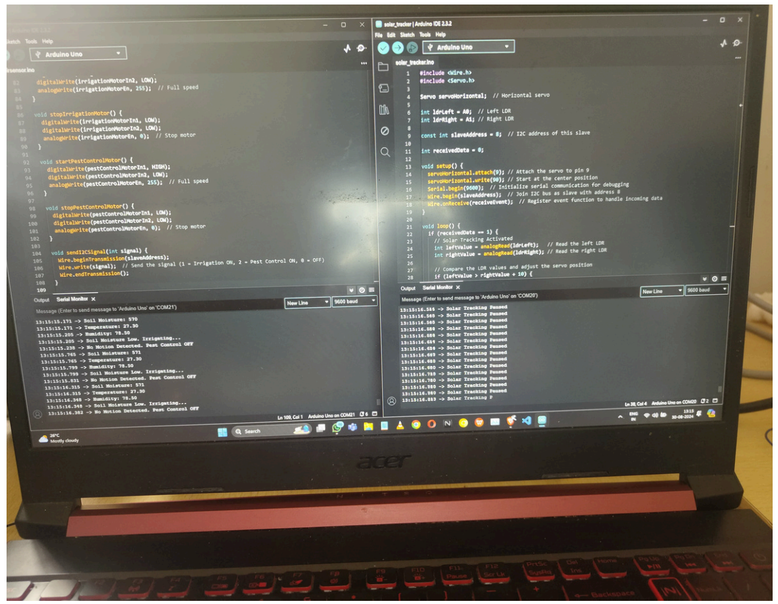
- **Limited Lab Access**: Due to the upcoming **internals**, the lab was opened for limited hours, restricting our access to the components and equipment needed for the project. **We could not work on the project at home as the components had to remain in the lab**.

- **Component Shortages**: We encountered several shortages of key components, including the **buzzer, solar panel mount, LDRs, water pumps(replaced with motor 1), and pesticide dispenser(replaced with motor 2)**. These components were either in use for other ongoing projects or unavailable, leading to delays and modifications in the project plan.

- **Frequent Dismantling:** Every time the lab was about to close, we had to **dismantle our circuit to return borrowed components.** This added extra work and reduced the efficiency of our progress.







THANK YOU