Plant Ambassador (PA Robot)
(Grade:9 Level: Hard)

Overview

About the Project
Plant Ambassador is an interactive plant-care monitoring system designed to help users understand the needs of their plants in a simple and engaging way. The project uses a soil moisture sensor to monitor the water level in the soil and visually communicates the plant's condition through facial expressions. A happy face is displayed when the plant is healthy and receiving proper care, while a frowning face appears when the plant lacks sufficient water. By giving the plant a "face," the system acts as an ambassador for the plant, expressing its needs in a way that is easy for anyone to understand.

Problem Statement
Many plants suffer or die because people forget to water them or are unsure about when and how much water is needed. Beginners, in particular, find it difficult to interpret plant health signs such as dry soil or wilting leaves. Traditional plant care requires constant attention and experience, which not everyone has. As a result, plants are often neglected, leading to poor growth or death.

Proposed Solution
The Plant Ambassador project provides a simple and effective solution by continuously monitoring soil moisture using a sensor. When the soil moisture level is adequate, the system displays a happy face, indicating that the plant is healthy. When the soil becomes too dry, a frowning face is shown to alert the user that the plant needs water. This visual feedback makes plant care more intuitive, reduces guesswork, and encourages timely watering. The system helps users maintain healthier plants while making plant care more interactive and enj

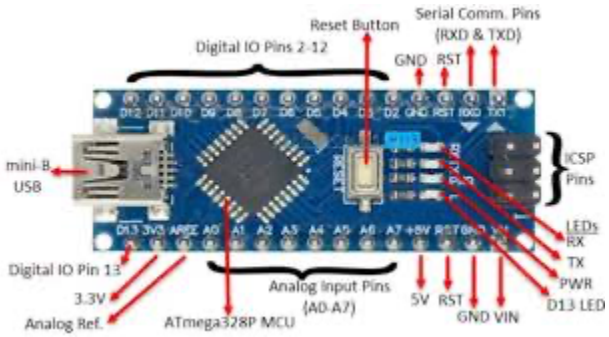
Prerequisite

To successfully design and implement the Plant Ambassador project, the following prerequisites are required. Basic understanding of Arduino programming**.** Knowledge of analog and digital input/output pins**.** Basic circuit design and wiring skills. Understanding of how soil moisture sensors work.Experience in working with arduino IDE for writing, compiling, and uploading the program to the Arduino Nano.

Components Required

Components	Quantity
Arduino Nano	1
Soil Moisture Sensor	1
OLED Display	1
Buzzer	1

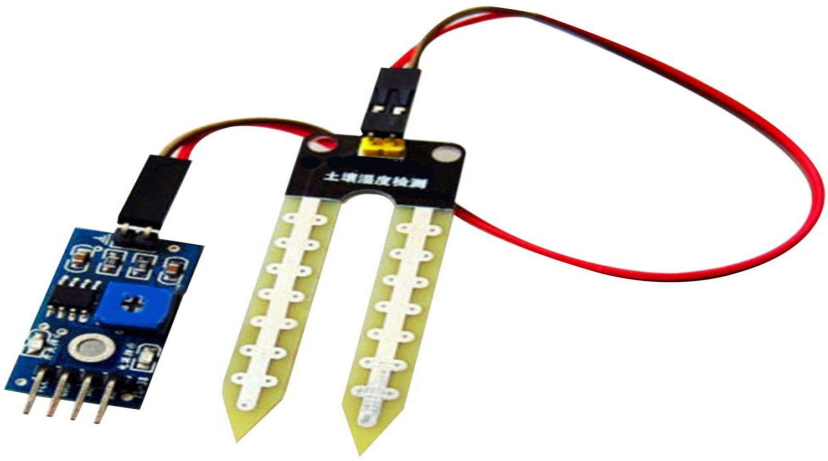
Arduino Nano:



The Arduino Nano is a small, compact microcontroller board based on the ATmega328P processor. It is used as the main control unit in the Plant Ambassador project. The Arduino Nano reads data from the soil moisture sensor, processes the input, and controls the output devices such as the facial expression display and the buzzer.

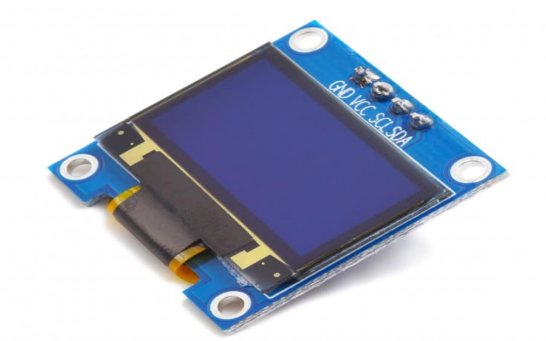
Due to its small size, low power consumption, and sufficient number of input/output pins, the Arduino Nano is well suited for embedded and portable applications. It can be easily programmed using the Arduino IDE and powered through a USB cable or an external power source.

Soil Moisture Sensor:



The soil moisture sensor is used to measure the water content present in the soil. In the Plant Ambassador project, this sensor continuously monitors the moisture level around the plant's roots and sends the data to the Arduino Nano.

OLED Display:



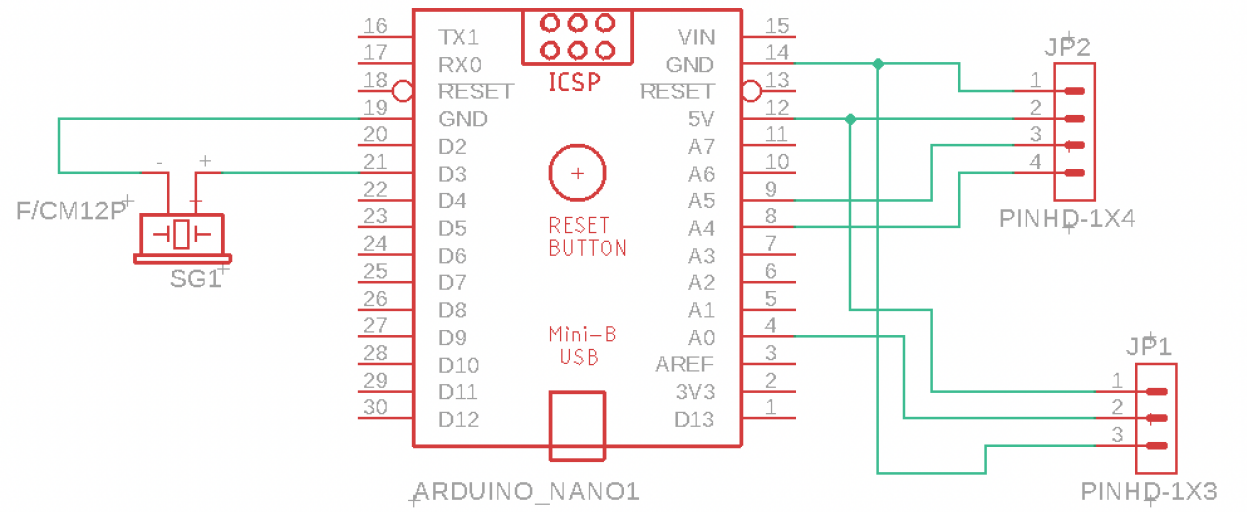
The **OLED (Organic Light Emitting Diode) display** is used to visually represent the condition of the plant in the **Plant Ambassador** project. It displays simple facial expressions, such as a **happy face** when the plant has sufficient water and a **frowning face** when the soil moisture level is low.

OLED displays provide high contrast, low power consumption, and clear visibility even in low light conditions. The display is controlled by the Arduino Nano, which updates facial expressions based on data received from the soil moisture sensor. This visual feedback makes it easy for users to quickly understand the plant's condition without needing technical knowledge.

Buzzer:

The buzzer is an output device used to provide an audible alert in the Plant Ambassador project. It is activated by the Arduino Nano when the soil moisture sensor detects that the soil is too dry. When the buzzer sounds, it alerts the user that the plant needs water, even if the user is not looking at the display. This audio indication works along with the OLED display to ensure timely attention to the plant’s needs.

Circuit Diagram



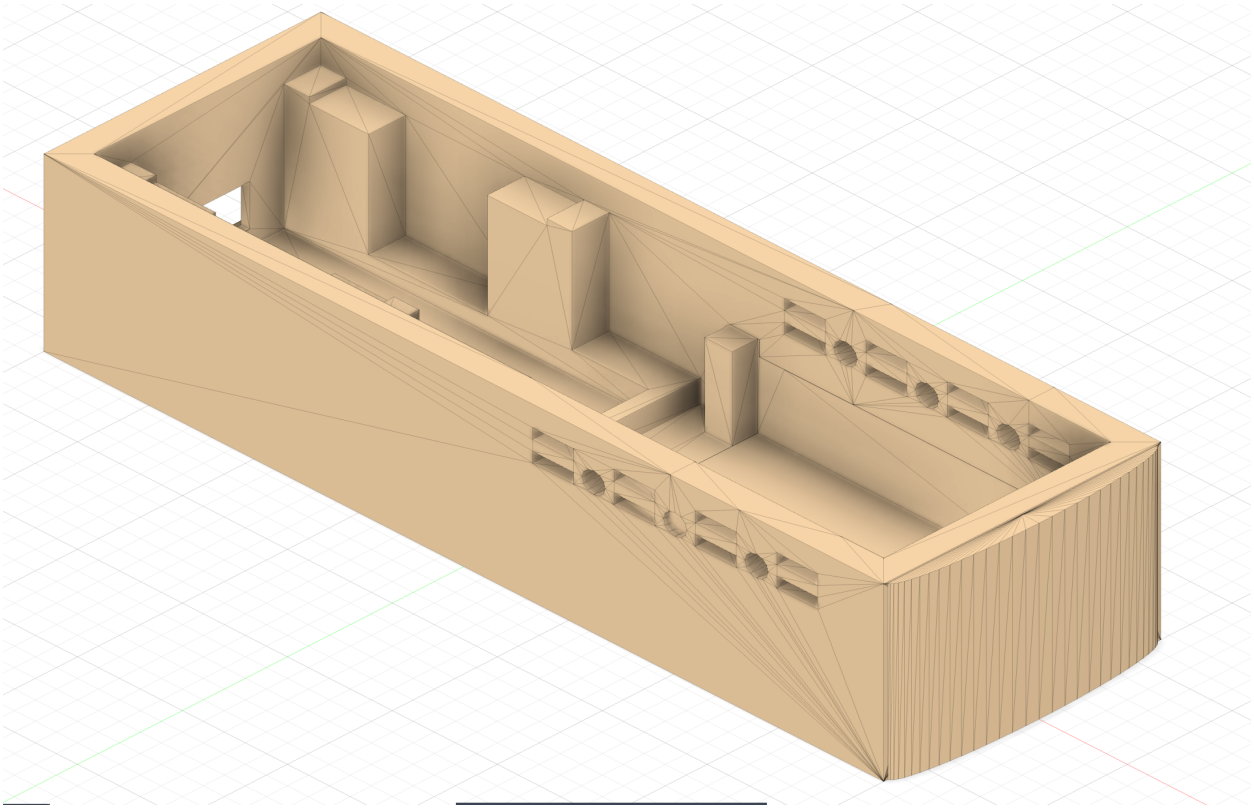
Port Mapping

Arduino Port	Device	Sensor/Actuator Pin
3	Buzzer	Positive pin
A4	OLED	SCL
A5	OLED	SDA
A0	Soil Moisture Sensor	Signal

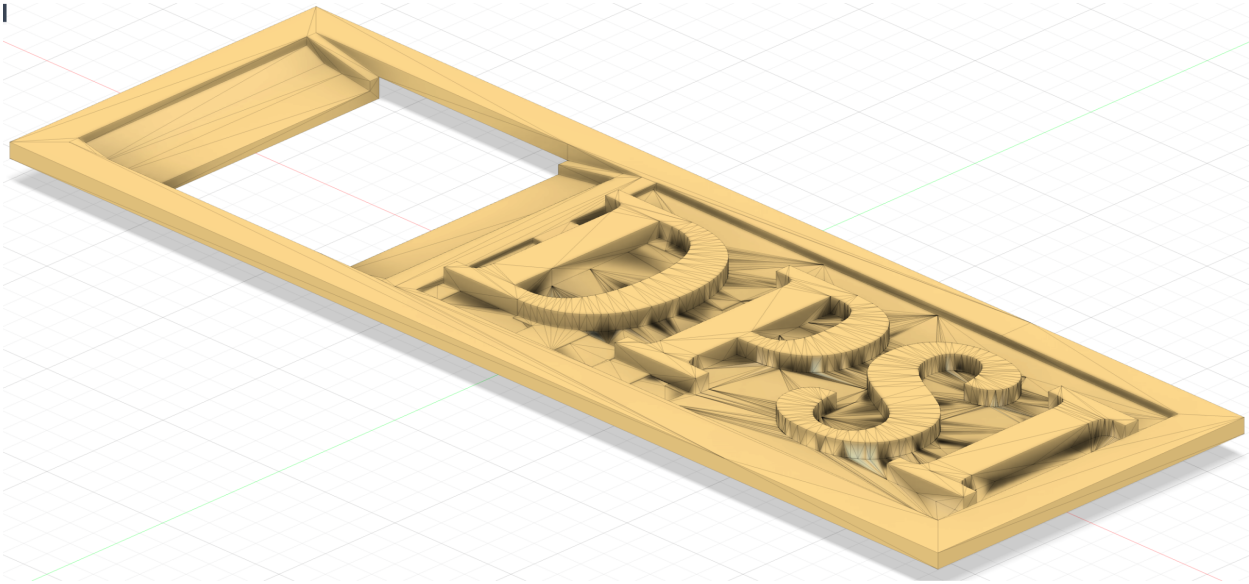
The circuit diagram consists of mainly four pins which are used for communicating with arduino and different sensors and actuators. The above mapping shows the pinout mapping and connection between arduino and different devices .

CAD Design

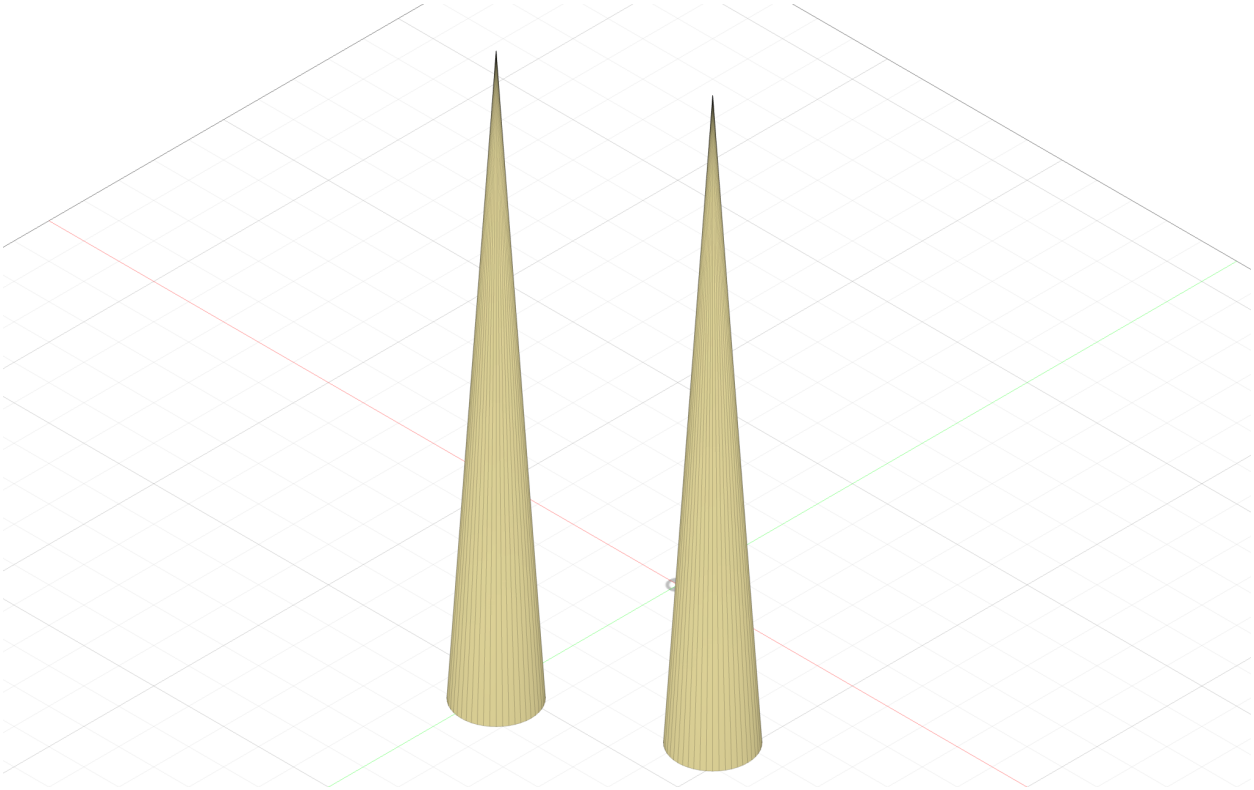
Bottom Case:



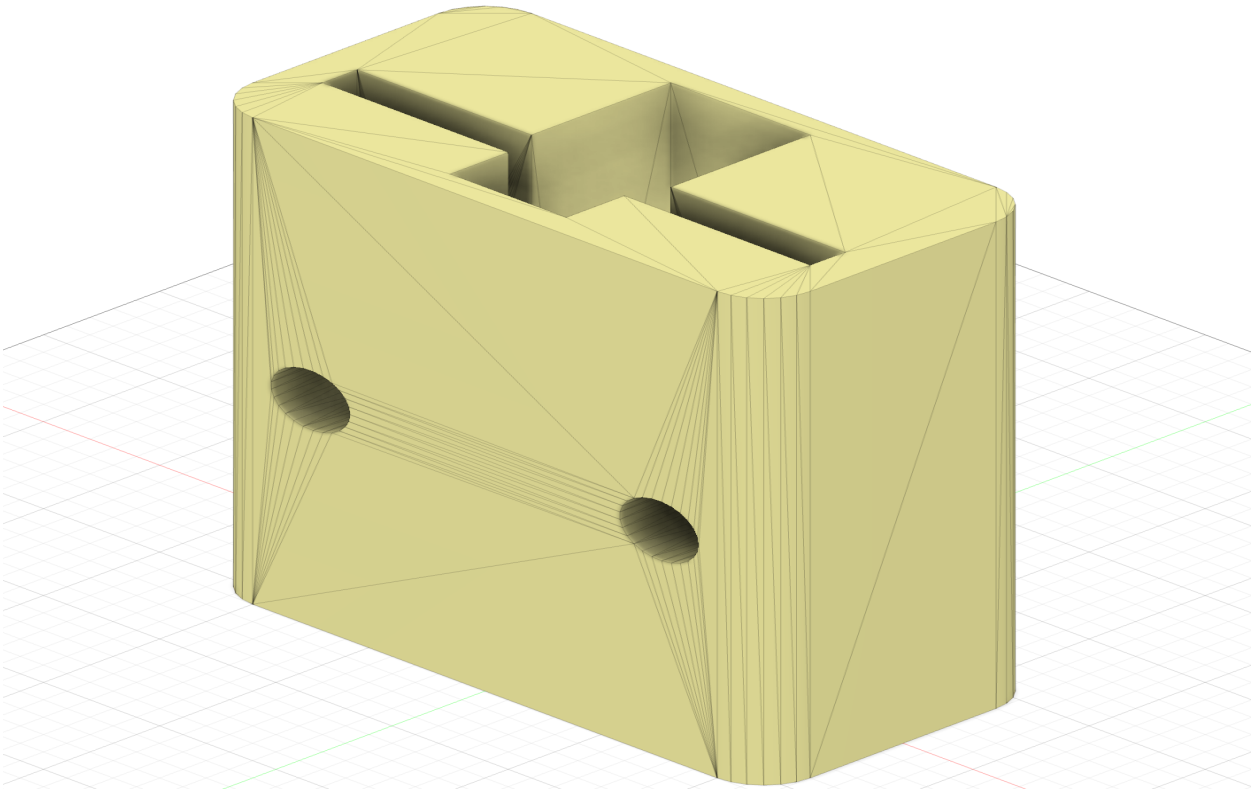
Top Case:



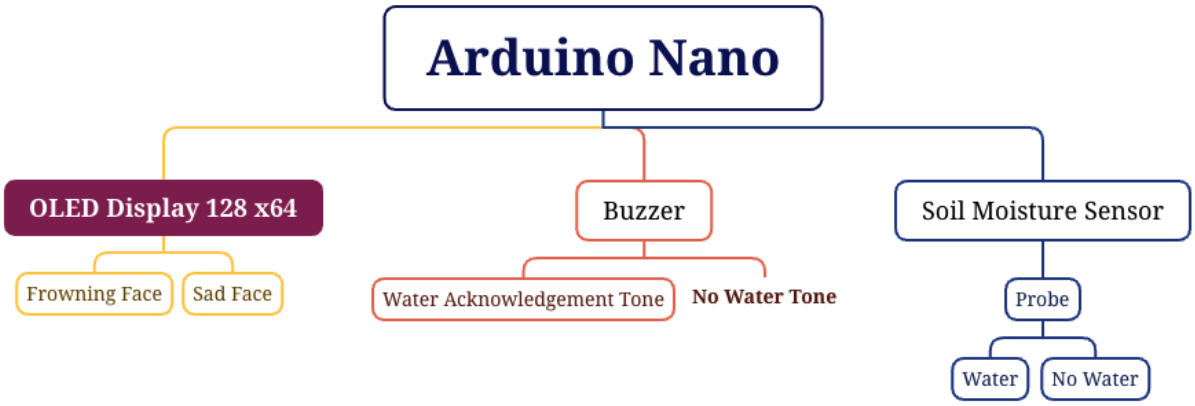
Legs:



Probes:



Block Diagram



The block diagram illustrates the overall working of the Plant Ambassador system and how different components interact with the Arduino Nano, which acts as the central controller.

Arduino Nano (Central Controller)

The Arduino Nano is the core of the system. It receives input data from the soil moisture sensor, processes this information, and controls the output devices such as the OLED display and the buzzer based on the moisture condition of the soil.

Soil Moisture Sensor

The soil moisture sensor is connected to the Arduino Nano and continuously monitors the moisture content in the soil through its probe. When **water is present** in the soil, the sensor sends a corresponding signal to the Arduino Nano. When **no water (dry soil)** is detected, a different signal value is sent.This data helps the Arduino Nano determine whether the plant is adequately watered or not.

OLED Display (128 × 64)

The OLED display provides visual feedback about the plant's condition. Based on the sensor input processed by the Arduino Nano a happy or normal face is displayed when sufficient water is detected and a sad or frowning face is displayed when the soil is dry and the plant needs water.This visual representation allows users to quickly understand the plant's condition at a glance.

Buzzer

The buzzer acts as an audio alert system. Depending on the soil moisture level, a water acknowledgment tone is produced when the plant has sufficient water, confirming healthy conditions. A no-water tone is activated when the soil moisture is low, alerting the user to water the plant.

Overall Working

The system operates in a continuous loop where the soil moisture sensor checks the soil condition, the Arduino Nano processes the data, and appropriate visual and audio outputs are generated. This ensures timely alerts and effective plant care through simple and intuitive feedback.

Code

Full Code location: https://github.com/TechMaverik/PA-Robot.git

```
#include "display_wrapper.h"

// --- Hardware Configuration ---
const int sensorPin = A0;
const int buzzerPin = 3;
const int threshold = 500;
const long BAUD_RATE = 115200;

// --- Musical Frequencies (Hz) ---
#define NOTE_G4  392
#define NOTE_D4  294
#define NOTE_AS3 233 // B Flat (Mournful)
#define NOTE_C5  523
#define NOTE_E5  659
#define NOTE_G5  784
#define NOTE_A2  110 // Low Angry Tone

// --- Eye State Struct ---
struct EyeState {
   int height;
   int width;
   int x;
   int y;
};

const int REF_EYE_HEIGHT = 40;
const int REF_EYE_WIDTH = 40;
const int REF_SPACE_BETWEEN_EYE = 10;
const int REF_CORNER_RADIUS = 10;

EyeState left_eye, right_eye;
bool wasDry = false;

// --- Timing & Animation Variables ---
unsigned long dryStartTime = 0;
unsigned long happyStartTime = 0;
unsigned long lastBlinkTime = 0;
bool animationDone = false;
int blinkInterval = 3000;
```

```
// --- Engine Functions ---

void draw_eyes() {
   g_draw_filled_round_rect(int(left_eye.x - left_eye.width / 2), int(left_eye.y -
left_eye.height / 2), left_eye.width, left_eye.height, REF_CORNER_RADIUS,
G_COLOR_WHITE);
   g_draw_filled_round_rect(int(right_eye.x - right_eye.width / 2), int(right_eye.y -
right_eye.height / 2), right_eye.width, right_eye.height, REF_CORNER_RADIUS,
G_COLOR_WHITE);
}

void reset_eyes(bool update = true) {
   left_eye.height = REF_EYE_HEIGHT; left_eye.width = REF_EYE_WIDTH;
   right_eye.height = REF_EYE_HEIGHT; right_eye.width = REF_EYE_WIDTH;
   left_eye.x = 64 - REF_EYE_WIDTH / 2 - REF_SPACE_BETWEEN_EYE / 2;
   left_eye.y = 32;
   right_eye.x = 64 + REF_EYE_WIDTH / 2 + REF_SPACE_BETWEEN_EYE / 2;
   right_eye.y = 32;
   if (update) { g_clear_display(); draw_eyes(); g_update_display(); }
}

// --- Masking Helpers ---

void apply_angry_mask() {
   g_draw_filled_triangle(left_eye.x - 21, left_eye.y - 21, left_eye.x + 21,
left_eye.y - 21, left_eye.x + 21, left_eye.y, G_COLOR_BLACK);
   g_draw_filled_triangle(right_eye.x - 21, right_eye.y - 21, right_eye.x + 21,
right_eye.y - 21, right_eye.x - 21, right_eye.y, G_COLOR_BLACK);
}

void draw_realistic_tear(int x, int y) {
   display.fillTriangle(x, y - 4, x - 2, y, x + 2, y, G_COLOR_WHITE);
   display.fillCircle(x, y + 1, 3, G_COLOR_WHITE);
}

// --- Melodies ---

void playCryingMelody() {
   // Mournful descending sequence
   tone(buzzerPin, NOTE_G4, 150); delay(200);
   tone(buzzerPin, NOTE_D4, 150); delay(200);
   tone(buzzerPin, NOTE_AS3, 300); delay(400);
   noTone(buzzerPin);
```

```
}

void playHappyMelody() {
   // Cheerful ascending sequence
   tone(buzzerPin, NOTE_C5, 80); delay(100);
   tone(buzzerPin, NOTE_E5, 80); delay(100);
   tone(buzzerPin, NOTE_G5, 150); delay(150);
   noTone(buzzerPin);
}

// --- Animation Routines ---

void fast_blink() {
   int speed = 20;
   reset_eyes(false);
   for(int i = 0; i < 2; i++) {
       left_eye.height -= speed; right_eye.height -= speed;
       g_clear_display(); draw_eyes(); g_update_display();
   }
   for(int i = 0; i < 2; i++) {
       left_eye.height += speed; right_eye.height += speed;
       g_clear_display(); draw_eyes(); g_update_display();
   }
}

void look_around_animation() {
   int moveAmount = 15;
   for(int i = 0; i < 2; i++) {
       left_eye.x -= moveAmount; right_eye.x -= moveAmount;
       g_clear_display(); draw_eyes(); g_update_display();
       delay(100);
       left_eye.x += moveAmount * 2; right_eye.x += moveAmount * 2;
       g_clear_display(); draw_eyes(); g_update_display();
       delay(100);
       left_eye.x -= moveAmount; right_eye.x -= moveAmount;
   }
   reset_eyes(true);
}

// --- State Functions ---

void show_happy() {
   reset_eyes(false);
```

```
  g_clear_display();
  draw_eyes();
  g_update_display();
}

void show_crying() {
  reset_eyes(false);
  left_eye.height = 12; // 30% open
  right_eye.height = 12;
  g_clear_display();
  draw_eyes();
  draw_realistic_tear(left_eye.x - 8, left_eye.y + 12);
  draw_realistic_tear(right_eye.x + 8, right_eye.y + 12);
  display.setTextSize(1);
  display.setCursor(35, 55);
  display.print("NEED WATER");
  g_update_display();
}

void show_angry() {
  reset_eyes(false);
  g_clear_display();
  draw_eyes();
  apply_angry_mask();
  display.setTextSize(1);
  display.setCursor(35, 55);
  display.print("NEED WATER");
  g_update_display();
}

// --- Setup & Loop ---

void setup() {
  Serial.begin(BAUD_RATE);
  pinMode(buzzerPin, OUTPUT);
  g_init_display();
  reset_eyes(true);
}

void loop() {
  int sensorValue = analogRead(sensorPin);

  if (sensorValue < threshold) {
```

```
      if (wasDry) {
          playHappyMelody();
          wasDry = false;
          happyStartTime = millis();
          animationDone = false;
      }

      if (!animationDone && (millis() - happyStartTime > 2000)) {
          look_around_animation();
          animationDone = true;
      } else {
          if (millis() - lastBlinkTime > blinkInterval) {
              fast_blink();
              lastBlinkTime = millis();
              blinkInterval = random(3000, 6000);
          } else {
              show_happy();
          }
      }
      noTone(buzzerPin);
  } else {
      if (!wasDry) { dryStartTime = millis(); wasDry = true; }
      long elapsed = (millis() - dryStartTime) % 5000;

      if (elapsed < 3000) {
          show_crying();
          playCryingMelody(); // Plays every ~1 second while crying
      } else {
          show_angry();
          // Aggressive angry pulse
          tone(buzzerPin, NOTE_A2, 100); delay(150); noTone(buzzerPin);
      }
  }
  delay(50);
}
```

Code Explanation

Hardware & constants

```
const int sensorPin = A0;
const int buzzerPin = 3;
const int threshold = 500;
```

sensorPin: analog input (moisture)
buzzerPin: piezo buzzer for sound
threshold: below this = wet / happy, above = dry / upset

```
const long BAUD_RATE = 115200;
```

Used for serial debugging (though not heavily used here).

Sound notes (melodies)

You define musical note frequencies:

```
#define NOTE_G4  392
#define NOTE_D4  294
#define NOTE_AS3 233
...
#define NOTE_A2  110
```

These are used to create:

Sad, descending melody when crying
Happy, ascending melody when water is restored
Low angry buzz when very dry

Eye system (graphics engine)

EyeState struct

```
struct EyeState {
   int height;
   int width;
   int x;
   int y;
};
```

Each eye has:

- size (width, height)
- screen position (x, y)

Two global eyes:

```
EyeState left_eye, right_eye;
```

Reference dimensions

```
const int REF_EYE_HEIGHT = 40;
const int REF_EYE_WIDTH = 40;
```

These define the “neutral” eye size and spacing.

State tracking

```
bool wasDry = false;
unsigned long dryStartTime = 0;
unsigned long happyStartTime = 0;
unsigned long lastBlinkTime = 0;
```

These variables help the system **remember past conditions**, not just the current sensor value.

Example:

wasDry lets the code detect when water was *just* added
dryStartTime controls crying → angry timing
lastBlinkTime handles natural blinking

Drawing the eyes

draw_eyes()

g_draw_filled_round_rect(...)

Draws two white rounded rectangles
One for each eye
Uses the EyeState values for position and size

---

reset_eyes()

Resets eyes to:

Default size
Centered on the display

Optionally redraws the screen.

This is the **baseline pose** every emotion starts from.

Facial expressions (masking)

Angry mask

apply_angry_mask()

Draws **black triangles** over the top of the eyes to create:

Slanted eyebrows
Angry expression

This is a clever trick: you’re not redrawing eyes, just masking them.

Tears

draw_realistic_tear()

A tear is:

- A small triangle (top)
- A circle (bottom)

Simple but expressive.

Sound behaviour

Crying melody

playCryingMelody()

- Descending notes
- Slower timing
- Feels sad / needy

Happy melody

playHappyMelody()

- Ascending notes
- Short and upbeat
- Played **once** when water returns

Animations

Fast blink

fast_blink()

Quickly shrinks eye height
Then expands it
Simulates blinking

Blink timing is randomised later so it feels alive.

Look-around animation

look_around_animation()

Moves eyes left → right → centre
Happens once after becoming happy
Adds personality

Emotional states

Happy

show_happy()

Normal open eyes
No text
Calm idle behaviour (blinking, looking around)

Crying (dry, but not too long)

show_crying()

Eyes half-closed
Tears under both eyes
Text: "NEED WATER"
Plays sad melody repeatedly

This happens during the **first 3 seconds** of dryness.

Angry (very dry)

show_angry()

Normal eyes + angry eyebrows
Text: "NEED WATER"
Low, aggressive buzzer pulses

This kicks in after crying times out.

---

setup()

```
void setup() {
  Serial.begin(BAUD_RATE);
  pinMode(buzzerPin, OUTPUT);
  g_init_display();
  reset_eyes(true);
}
```

Initialises serial, buzzer, and display
Draws neutral eyes at startup

Main loop (the brain)

Read sensor

int sensorValue = analogRead(sensorPin);

If sensor is WET (happy)

if (sensorValue < threshold)

Transition from dry → wet

```
if (wasDry) {
   playHappyMelody();
   wasDry = false;
}
```

This ensures:

- Happy sound plays **once**
- Not every loop

After 2 seconds of happiness

```
look_around_animation();
```

Then:

- Idle blinking
- Random blink intervals
- Normal happy face

If sensor is DRY

```
else {
```

Start dry timer

```
if (!wasDry) {
   dryStartTime = millis();
   wasDry = true;
}
```

Crying vs angry

```
long elapsed = (millis() - dryStartTime) % 5000;
```

First 3 seconds → crying
Next 2 seconds → angry
Loop repeats

Crying phase

```
show_crying();
playCryingMelody();
```

Angry phase

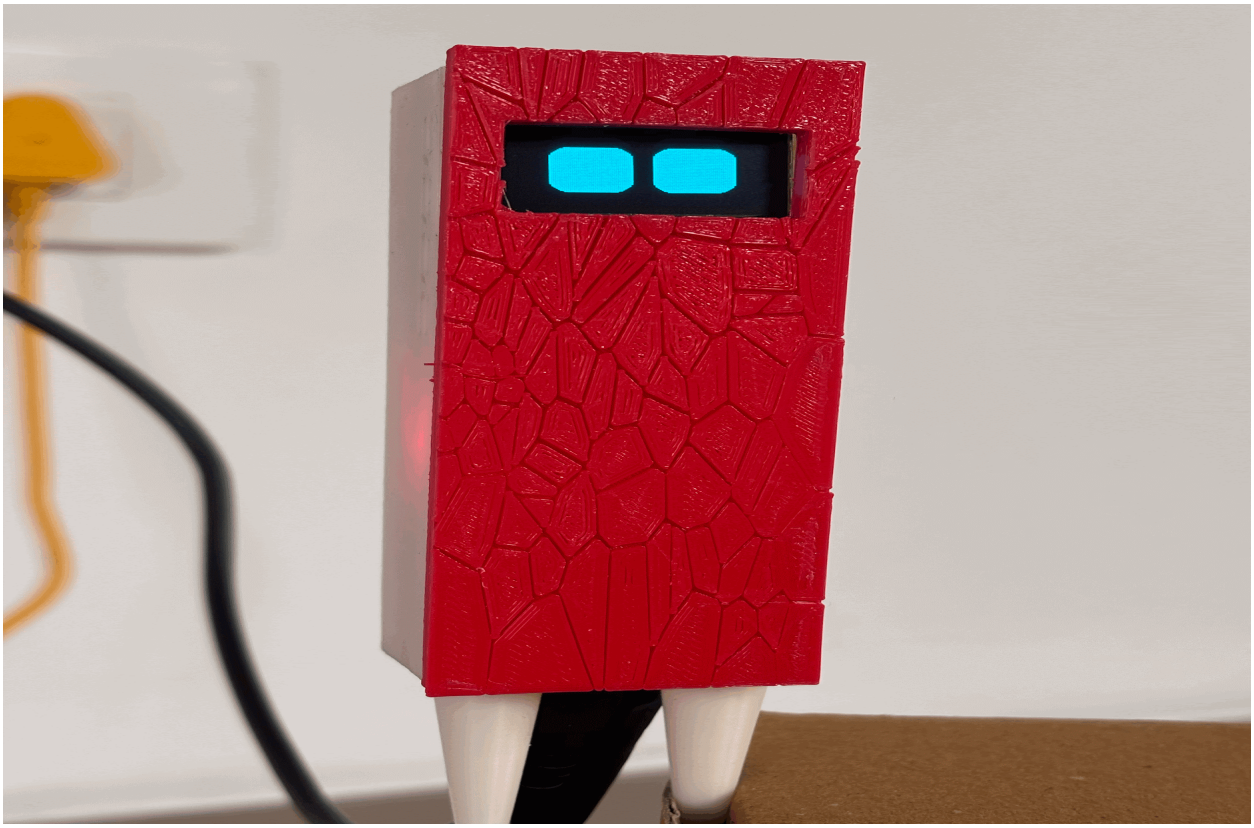
```
show_angry();
tone(buzzerPin, NOTE_A2, 100);
```

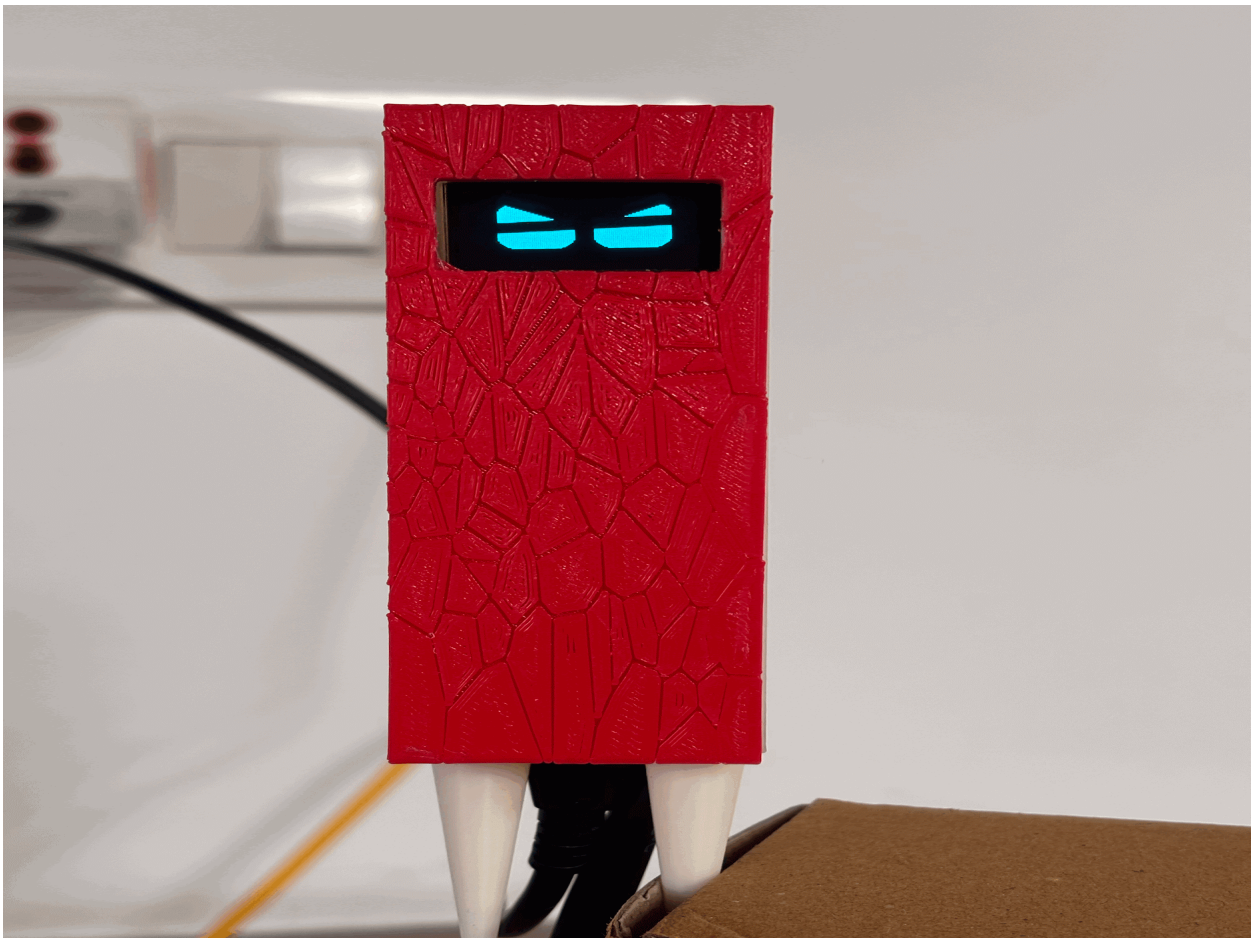
Final touch

```
delay(50);
```

Keeps the loop stable and avoids flicker.

Project Execution and Output





Operation:

1. Connect the nano with a usb cable and power it using an adapter.
2. Connect the PA Robot probe inside the soil.
3. Wait for voice acknowledgement from the robot.

Troubleshooting

1. **OLED is not showing any emojis**
   Check for the SCL SDA connections of oled
   Check the VCC cable for the oled display

2. **No Power is coming in the Arduino Nano**
   Check the usb cable is faulty or not

Price Estimation

Components	Price in ₹
Arduino Nano	300
Soil Moisture Sensor	50
OLED Display	200
Buzzer	20
PLA	200

Total Estimated: 770 ₹

Conclusion

The Plant Ambassador project successfully demonstrates an effective and user-friendly solution for monitoring plant water requirements using embedded systems. By integrating an Arduino Nano, soil moisture sensor, OLED display, and buzzer, the system is able to continuously monitor soil conditions and provide timely visual and audio feedback to the user.

The use of expressive facial indicators on the OLED display makes the system intuitive and easy to understand, even for users with little technical knowledge. The buzzer further enhances the alert mechanism by ensuring that the user is notified when the plant requires water. This reduces the chances of plant neglect and promotes healthier plant growth.

Overall, the project highlights how simple electronic components and sensors can be combined to create an interactive and practical plant-care solution. The Plant Ambassador system can be further enhanced by adding features such as wireless communication, mobile notifications, or automated watering, making it suitable for smart agriculture and home gardening applications.