

# USER MANUAL



SMART AGRICULTURE SYSTEM (BILLING SYSTEM)

**TABLE OF CONTENTS:**

SNO.	DESCRIPTION	PAGE NO.
1	Introduction	3
2	Getting Started	3
3	Configurations	3
4	System Operation	8
5	Maintenance and troubleshooting	16
6	Safety Precautions	17
7	Appendix	18
8	Conclusion	19
9	References	19

## **1. Introduction:**

- System Overview-

The Smart Agriculture System (Billing System) is an IoT project designed to address water resource management and environmental challenges in agriculture. It offers real-time monitoring and control of water resources and helps reduce stubble burning. This user manual will guide you through the setup and operation of the system.

- Purpose of the User Manual-

This user manual serves as a comprehensive guide for setting up, configuring, and using the Smart Agriculture System. It provides instructions, tips, and troubleshooting guidance to ensure you make the most of this system.

## **2. Getting Started:**

### Hardware Requirements-

The following components are required for this project-

- 1) Arduino nano 33 IOT
- 2) Raspberry Pi
- 3) MQ135 Air Quality Sensor
- 4) Soil Moisture Sensor
- 5) Ultrasonic Sensor
- 6) Internet Connectivity
- 7) L298N Motor Driver
- 8) Irrigation System
- 9) LCD

### Software Requirements-

The given system requires two software setup one for raspberry pi and other for Arduino nano using Arduino IDE.

### System Components:

1. Arduino nano 33 IOT- Used for data collection, sensor integration and communication with raspberry pi
2. Raspberry pi- Acts as central hub, processes data, handles billing system, provide remote access and user interface.
3. MQ135 Sensor- Used for calculating air quality index.
4. Soil Moisture Sensor- used for depicting the soil moisture level of fields.
5. Ultrasonic Sensor- Used for depicting the depth of water left in the reservoir or water container.
6. Water Pump- used for irrigation of fields.
7. L298N motor driver- Used for controlling and driving the motors on particular speed.

## **3. Configurations:**

Hardware Setup:

- Connect the Arduino Nano to the sensors.
- Power up the system components.
- Ensure all connections are secure.

#### Software Configuration :

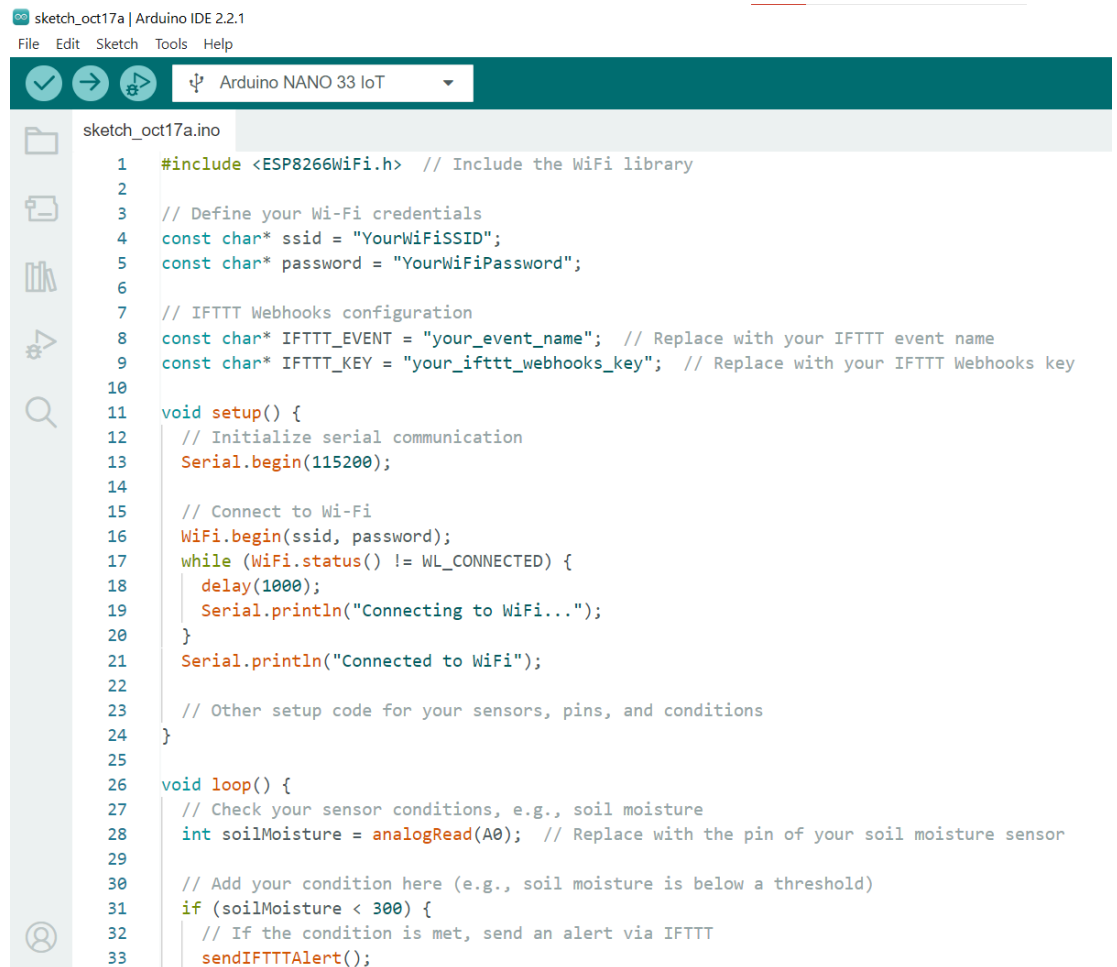
- For setting up raspberry pi user can follow the steps:
  1. Downloaded Raspberry Pi OS: <https://www.raspberrypi.org/downloads/>
  2. Format the SD card as FAT 2) Install the Raspberry Pi OS into the SD card using Raspberry Pi Imager. (a quick tutorial available on how to on <https://www.youtube.com/watch?v=ntaXWS8Lk34> 3)
  3. Insert the card into the Raspberry Pi and boot it up.
  4. Make sure you plug a monitor, keyboard and mouse into the RPi first.
  5. Power on the Raspberry Pi and verify that the operating system has been installed properly.

For any other details user can follow the given documentation- <https://www.raspberrypi.com/documentation/computers/getting-started.html>
- Setup IFTTT integration for alerts:
 

Steps:

  1. Connect the Arduino Nano to the Internet:
  2. Ensure your Arduino Nano has internet connectivity through a Wi-Fi module (like ESP8266) or an Ethernet shield. You need this connection to send data to IFTTT.
  3. Create an IFTTT Account: If you don't already have one, sign up for an IFTTT account on the IFTTT website or mobile app.
  4. Create a New Applet: In IFTTT, create a new applet. An applet consists of a trigger (if) and an action (then).
  5. Select a Trigger Service: Choose a service that will trigger the alert. For example, you can use "Webhooks" (or any other appropriate service) as the trigger. Webhooks allow you to receive data from external sources.
  6. Configure the Trigger: Set up the trigger conditions. In the case of Webhooks, you will define an event name. You'll use this event name in your Arduino code to trigger the alert.
  7. Create an Action Service: Choose the action service you want to perform when the trigger condition is met. This could be sending a notification, email, SMS, or performing any other action provided by IFTTT.
  8. Configure the Action: Set up the action based on your requirements. For example, if you want to receive a notification when the soil moisture level is low, you can configure IFTTT to send a notification to your smartphone.
  9. Finish Creating the Applet: Review and confirm the applet configuration.
  10. Obtain an IFTTT Webhooks Key
  11. Write Arduino Code
  12. Test Your Code
  13. Receive Alerts: If everything is set up correctly, you will receive alerts based on the conditions you defined in your IFTTT applet.

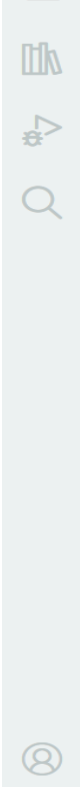
Sample code is given below:



```
sketch_oct17a | Arduino IDE 2.2.1
File Edit Sketch Tools Help

Arduino NANO 33 IoT

sketch_oct17a.ino
1  #include <ESP8266WiFi.h> // Include the WiFi library
2
3  // Define your Wi-Fi credentials
4  const char* ssid = "YourWiFiSSID";
5  const char* password = "YourWiFiPassword";
6
7  // IFTTT Webhooks configuration
8  const char* IFTTT_EVENT = "your_event_name"; // Replace with your IFTTT event name
9  const char* IFTTT_KEY = "your_ifttt_webhooks_key"; // Replace with your IFTTT Webhooks key
10
11 void setup() {
12     // Initialize serial communication
13     Serial.begin(115200);
14
15     // Connect to Wi-Fi
16     WiFi.begin(ssid, password);
17     while (WiFi.status() != WL_CONNECTED) {
18         delay(1000);
19         Serial.println("Connecting to WiFi...");
20     }
21     Serial.println("Connected to WiFi");
22
23     // Other setup code for your sensors, pins, and conditions
24 }
25
26 void loop() {
27     // Check your sensor conditions, e.g., soil moisture
28     int soilMoisture = analogRead(A0); // Replace with the pin of your soil moisture sensor
29
30     // Add your condition here (e.g., soil moisture is below a threshold)
31     if (soilMoisture < 300) {
32         // If the condition is met, send an alert via IFTTT
33         sendIFTTAlert();
34     }
35 }
```



```

33     sendIFTTAlert();
34 }
35
36 // Add other code for your project's functionality
37
38 delay(10000); // Delay for 10 seconds before checking again
39 }
40
41 void sendIFTTAlert() {
42     // Create an HTTP client object
43     WiFiClient client;
44
45     // Make a POST request to the IFTTT Webhooks URL
46     client.connect("maker.ifttt.com", 80);
47     client.print("POST /trigger/");
48     client.print(IFTTT_EVENT);
49     client.println("/with/key/");
50     client.println(IFTTT_KEY);
51     client.println("Host: maker.ifttt.com");
52     client.println("Connection: close\r\n\r\n");
53
54     // Check for a successful connection
55     if (client.connect("maker.ifttt.com", 80)) {
56         Serial.println("Sending alert to IFTTT");
57         client.stop(); // Close the connection
58     } else {
59         Serial.println("Failed to connect to IFTTT");
60     }
61 }
62

```

- Setting Up Communication between Arduino and raspberry Pi:

There are multiple ways to establish communication between an Arduino Nano and a Raspberry Pi using methods like UART serial communication, I2C, SPI, USB serial (Virtual COM Port), WiFi, Ethernet, Bluetooth, radio communication (RF, LoRa), and MQTT, depending on factors like range, data transfer speed, and project requirements.

This project uses Wi-Fi to share data from Arduino to Raspberry pi-

1) Code for Arduino-

```
sketch_oct17a.ino
1  #include <WiFiNINA.h>
2  char ssid[] = "";
3  char pass[] = "";
4  WiFiClient client;
5  void setup() {
6      // Initialize serial communication for debugging
7      Serial.begin(9600);
8
9      // Connect to Wi-Fi network
10     while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
11         Serial.println("Attempting to connect to WiFi...");
12         delay(1000);
13     }
14     if (WiFi.status() == WL_CONNECTED) {
15         Serial.println("Connected to WiFi");
16         Serial.print("IP Address: ");
17         Serial.println(WiFi.localIP());
18     }
19     else {
20         Serial.println("Connection failed.");
21     }
22     Serial.println("Connected to WiFi");
23 }
24 void loop() {
25     senddata("HELLO EVERYONE I AM HERE");
26 }
27
28 void senddata(String data)
29 {
30     if (client.connect("192.168.151.251", 12345)) {
31         // Connected to Raspberry Pi
32         Serial.println("Connected to Raspberry Pi");
33
34         // Data to send
35         String dataToSend = data;;
36
37         // Send the data
38         client.println(dataToSend);
39
40         // Close the connection
41         client.stop();
42
43         // Wait for a moment before sending more data
44         delay(5000);
45     } else {
46         Serial.println("Connection failed");
47     }
48 }
49
```

2) Code For raspberry Pi-



```
import socket

# Raspberry Pi's IP address and port to listen on
host = '192.168.151.251' # Listen on all available network interfaces
port = 12345 # Use the same port as on the Arduino

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the address and port
server_socket.bind((host, port))

# Listen for incoming connections
server_socket.listen(5)

print("Listening for incoming connections...")

while True:
    # Accept a connection
    client_socket, client_address = server_socket.accept()
    print(f"Accepted connection from {client_address}")

    # Receive data from the Arduino
    data = client_socket.recv(1024).decode()
    print(f"Received data: {data}")

    # Process the received data as needed

    # Close the client socket
    client_socket.close()
```

#### 4. System Operation-

- Real-Time Monitoring:  
Explore the real-time data displayed on the user interface.  
Understand how to interpret sensor readings.
- Controlling Water Flow:  
Learn how to manually control water flow using the buttons present on the GUI of the system.  
Understand the automated water management process.
- Receiving Alerts:  
Configure alert preferences and thresholds in the system.  
Learn how to respond to IFTTT alerts.
- Integrate all Sub- codes to get complete execution of whole system-  
Final code of this system is provided below-  
1<sup>st</sup> one is for Arduino nano  
2<sup>nd</sup> one is for raspberry pi



```

finalprojectnano_copy_20231016001457.ino
1  #include <ArduinoHttpClient.h>
2  #include <LiquidCrystal_I2C.h>
3  #include <Wire.h>
4  #include <WiFiNINA.h> // Include the WiFiNINA library
5
6  // WiFi network credentials
7  char ssid[] = "aryan"; // Replace with your WiFi network SSID
8  char pass[] = "24700124"; // Replace with your WiFi network password
9
10 // Define pin constants for sensors
11 #define SOIL_MOISTURE_SENSOR A0
12 #define MQ135_SENSOR A1
13
14 // Define LCD configuration
15 LiquidCrystal_I2C lcd(0x27, 16, 2);
16
17 // Define IFTTT Webhooks constants
18 const char* apiKey = "dLfauJTccstZhrqErwcjurFLGFpHEalm1qFqZ-ZNj6b"; // Replace with your IFTTT Webhooks API key
19 const char* eventSoilMoisture = "email_to_farmer";
20 const char* eventAirQuality = "email_to_farmer";
21 const int soilMoistureThresholdMin = 90; // Adjust as needed
22 const int airQualityThreshold = 150; // Adjust as needed
23
24 WiFiClient client;
25
26 void setup() {
27     Serial.begin(9600);
28     pinMode(SOIL_MOISTURE_SENSOR, INPUT);
29     Wire.begin();
30     lcd.init();
31     lcd.backlight();
32
33     // Connect to WiFi
34     while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
35         Serial.println("Attempting for Connecting to WiFi...");
36         delay(1000);
37     }
38     if (WiFi.status() == WL_CONNECTED) {
39         Serial.println("Connected to WiFi");
40         Serial.print("IP Address: ");
41         Serial.println(WiFi.localIP());
42     }
43     else {
44         Serial.println("Connection failed.");
45     }
46     Serial.println("Connected to WiFi");
47 }
48
49 void loop() {

```

finalprojectnano\_copy\_20231016001457.ino

49

void loop() {

50

int soilMoistureValue = readSoilMoistureSensor();

51

int airQualityValue = readMQ135Sensor();

52

53

// Display sensor values on LCD

54

lcd.setCursor(0, 0);

55

lcd.print("Soil Moisture:");

56

lcd.setCursor(0, 1);

57

lcd.print(" ");

58

lcd.setCursor(0, 1);

59

lcd.print(soilMoistureValue);

60

lcd.print("%");

61

delay(2000);

62

lcd.clear();

63

64

lcd.setCursor(0, 0);

65

lcd.print("Air Quality:");

66

lcd.setCursor(0, 1);

67

lcd.print(" ");

68

lcd.setCursor(0, 1);

69

lcd.print(airQualityValue);

70

delay(2000);

71

lcd.clear();

72

73

// Check soil moisture threshold and send email alert if exceeded

74

if (soilMoistureValue <= 40) {

75

| sendEmailAlert("email\_to\_farmer", "Soil Mosturing Sensor", "LOW Moisture");

76

}

77

else if(soilMoistureValue > 40 && soilMoistureValue <= 80) {

78

| sendEmailAlert("email\_to\_farmer", "Soil Mosturing Sensor", "MID Moisture");

79

}

80

else if(soilMoistureValue >80 && soilMoistureValue<=100) {

81

| sendEmailAlert("email\_to\_farmer", "Soil Mosturing Sensor", "HIGH Moisture");

82

}

83

84

85

// Check air quality threshold and send email alert if exceeded

86

if (airQualityValue>0 && airQualityValue<=150) {

87

| sendEmailAlert("email\_to\_farmer", "Air Quality Alert", "Air Quality is GOOD");

88

}

89

else if(airQualityValue>150 && airQualityValue<=299)

90

{

91

| sendEmailAlert("email\_to\_farmer", "Air Quality Alert", "Air Quality is FAIR");

92

}

93

else if(airQualityValue>299)

94

{

95

| sendEmailAlert("email\_to\_farmer", "Air Quality Alert", "Air Quality is DANGEROUS");

96

}

97

```

finalprojectnano_copy_20231016001457.ino
98 // Continue displaying and serial printing sensor data
99 Serial.print("Soil Moisture: ");
100 Serial.print(soilMoistureValue);
101 Serial.println("%");
102 Serial.print("Air Quality: ");
103 Serial.println(airQualityValue);
104 senddata(soilMoistureValue,airQualityValue);
105
106 delay(2000);
107 }
108
109
110 int readSoilMoistureSensor() {
111     int soilMoistureValue = analogRead(SOIL_MOISTURE_SENSOR);
112     int moisturePercentage = map(soilMoistureValue, 0, 1023, 0, 100);
113     return moisturePercentage;
114 }
115
116 int readMQ135Sensor() {
117     int airQualityValue = analogRead(MQ135_SENSOR);
118     return airQualityValue;
119 }
120
121 void sendEmailAlert(const char* event, const char* value1, const char* value2) {
122     if (client.connect("maker.ifttt.com", 80)) {
123         String jsonData = "{\"value1\":\"" + String(value1) + "\",\"value2\":\"" + String(value2) + "\"}";
124         String url = "/trigger/" + String(event) + "/with/key/" + String(apiKey);
125
126         client.print("POST " + url + " HTTP/1.1\r\n");
127         client.print("Host: maker.ifttt.com\r\n");
128         client.print("Content-Type: application/json\r\n");
129         client.print("Content-Length: " + String(jsonData.length()) + "\r\n\r\n");
130         client.print(jsonData);
131         client.print("\r\n");
132
133         delay(1000);
134
135         while (client.available()) {
136             char c = client.read();
137             Serial.print(c);
138         }
139
140         client.stop();
141     } else {
142         Serial.println("Error connecting to IFTTT");
143     }
144 }
145 void senddata(int value1, int value2) {
146     if (client.connect("192.168.78.69", 12345)) {
147
148         void senddata(int value1, int value2) {
149             if (client.connect("192.168.78.69", 12345)) {
150                 // Connected to Raspberry Pi
151                 Serial.println("Connected to Raspberry Pi");
152
153                 // Data to send
154                 String dataToSend = String(value1) + "," + String(value2);
155
156                 // Send the data
157                 client.println(dataToSend);
158
159                 // Close the connection
160                 client.stop();
161
162                 // Wait for a moment before sending more data
163                 delay(5000);
164             } else {
165                 Serial.println("Connection failed");
166             }
167         }
168     }
169 }

```

```
import RPi.GPIO as GPIO
import time
import socket
import tkinter as tk
from threading import Thread

# Define the GPIO pins for TRIG (trigger) and ECHO (echo) of the ultrasonic sensor
TRIG_PIN = 19
ECHO_PIN = 20
SENSOR_HEIGHT_ABOVE_WATER = 14

# Define the GPIO pins
ENA = 17
IN1 = 18
IN2 = 27

GPIO.setmode(GPIO.BCM)
# Set up the GPIO pins
GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)

# Create PWM object
pwm = GPIO.PWM(ENA, 100) # 100 Hz frequency

# Start the PWM with 0% duty cycle (stopped)
pwm.start(0)

# Speed of sound in cm/s (approximately 34300 cm/s at room temperature)
SPEED_OF_SOUND = 34300

# Raspberry Pi's IP address and port to listen on
host = '192.168.78.69' # Listen on all available network interfaces
port = 12345 # Use the same port as on the Arduino

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the address and port
server_socket.bind((host, port))

# Listen for incoming connections
server_socket.listen(5)
# Initialize global variables for sensor data
soilMoisture = 0
airQuality = 0
```

```

distance = 0
depth = 0
volume = 0
amount = 0

# Create a function to continuously listen for incoming connections
def listen_for_connections():
    print("Listening for incoming connections...")
    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Accepted connection from {client_address}")
        data = client_socket.recv(1024).decode()
        print(f"Received data: {data}")
        process_arduino_data(data)
        client_socket.close()

# Create a thread to run the network communication
network_thread = Thread(target=listen_for_connections)
network_thread.daemon = True
network_thread.start()

# Create a function to update the GUI with sensor readings
def update_gui():
    # Update the GUI elements with sensor readings
    distance_var.set(f"Distance: {distance:.2f} cm")
    depth_var.set(f"Depth: {depth:.2f} cm")
    volume_var.set(f"Volume Used: {volume:.2f} cubic cm")
    bill_var.set(f"Bill: {amount:.2f} Rs")
    soil_moisture_var.set(f"Soil Moisture: {soilMoisture}")
    air_quality_var.set(f"Air Quality: {airQuality}")
    root.after(1000, update_gui) # Update the GUI every second

# Create a function to setup the ultrasonic sensor
def setup_sensor():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(TRIG_PIN, GPIO.OUT)
    GPIO.setup(ECHO_PIN, GPIO.IN)

# Create a function to measure distance using the ultrasonic sensor
def measure_distance():
    # (The measure_distance function from your original code here)
    GPIO.output(TRIG_PIN, GPIO.LOW)
    time.sleep(0.1)

    GPIO.output(TRIG_PIN, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, GPIO.LOW)

```

```

while GPIO.input(ECHO_PIN) == 0:
    pulse_start = time.time()

while GPIO.input(ECHO_PIN) == 1:
    pulse_end = time.time()

pulse_duration = pulse_end - pulse_start
distance = pulse_duration * SPEED_OF_SOUND / 2 # Calculate distance in cm
return distance

# Create a function to calculate depth from distance
def depth_from_distance(distance):
    # (The depth_from_distance function from your original code here)
    # Assuming water has a similar speed of sound as air (approx. 34300 cm/s)
    # and that distance is measured from the sensor to the water surface
    # Convert the distance to depth assuming the sensor is placed above the water
    depth = 0
    if distance > 0:
        depth = SENSOR_HEIGHT_ABOVE_WATER-distance
    return depth

# Create a function to calculate volume left
def volume_left(depth):
    initial_volume=991.847
    new_volume=0
    net_volume=0
    if depth>0:
        new_volume=3.14*4.75*4.75*depth
        net_volume=initial_volume-new_volume
    return net_volume

# Create a function to calculate the bill
def bill(net_volume):
    # (The bill function from your original code here)
    rate=0.2
    bill_amount=rate*net_volume
    return bill_amount

# Create a function to process data from the Arduino
def process_arduino_data(data):
    global soilMoisture, airQuality, distance, depth, volume, amount

    values = data.split(",")
    if len(values) == 2:
        soilMoisture = int(values[0])

```

```

        airQuality = int(values[1])

        # Process sensor data and update GUI
        setup_sensor()
        distance = measure_distance()
        depth = depth_from_distance(distance)
        volume = volume_left(depth)
        amount = bill(volume)

def turn_on_pump():
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    pwm.ChangeDutyCycle(50) # 50% speed
    status_label.config(text="Pump turned on.")

# Function to turn off the pump
def turn_off_pump():
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.LOW)
    pwm.ChangeDutyCycle(0) # 0% speed
    status_label.config(text="Pump turned off.")

# Initialize the GUI
root = tk.Tk()
root.title("Sensor Data and Billing")

# Create Tkinter variables to display sensor data
soil_moisture_var = tk.StringVar()
air_quality_var = tk.StringVar()
distance_var = tk.StringVar()
depth_var = tk.StringVar()
volume_var = tk.StringVar()
bill_var = tk.StringVar()

# Create labels to display sensor data
distance_label = tk.Label(root, textvariable=distance_var)
depth_label = tk.Label(root, textvariable=depth_var)
volume_label = tk.Label(root, textvariable=volume_var)
bill_label = tk.Label(root, textvariable=bill_var)
soil_moisture_label = tk.Label(root, textvariable=soil_moisture_var)
air_quality_label = tk.Label(root, textvariable=air_quality_var)

# Pack the labels in the GUI
distance_label.pack()
depth_label.pack()
volume_label.pack()

```



```

bill_label.pack()
soil_moisture_label.pack()
air_quality_label.pack()

on_button = tk.Button(root, text="Turn On Pump", command=turn_on_pump)
off_button = tk.Button(root, text="Turn Off Pump", command=turn_off_pump)
status_label = tk.Label(root, text="Pump is off")

# Arrange the widgets in the window
on_button.pack()
off_button.pack()
status_label.pack()

# Start the GUI event loop
try:
    root.after(1000, update_gui) # Start updating the GUI every second
    root.mainloop()
except KeyboardInterrupt:
    print("Exiting the control script.")

finally:
    # Cleanup and cleanup GPIO settings
    GPIO.cleanup()

```

## 5. Maintenance and Troubleshooting

Maintenance and troubleshooting are critical aspects to ensure the system's continued functionality and effectiveness

### i. Sensor Maintenance:

- Calibration: Regularly calibrate the sensors, including the MQ135 Air Quality Sensor, Soil Moisture Sensor, and Ultrasonic Sensor. Ensure that they provide accurate data by comparing their readings to known values or standards.
- Periodic Checks: Establish a maintenance schedule to check the sensors for physical damage or contamination. Clean the sensor surfaces if needed, and replace any worn-out or damaged components.
- Battery Replacement: If your sensors are battery-powered, periodically check and replace the batteries to ensure uninterrupted data collection.

### ii. Cleaning and Upkeep:

- Protection from Environmental Factors: Place the sensors in protective enclosures to shield them from dust, water, and other environmental elements. This helps extend the lifespan of the sensors and ensures consistent data collection.
- Preventive Cleaning: Clean the sensor surfaces to remove any dirt or debris that might affect their performance. Regular cleaning can help maintain sensor accuracy.
- General System Maintenance: Ensure that all components of the system, including the Arduino Nano, Raspberry Pi, and motor control mechanisms, are clean and free from dust or debris. Over time, dust can accumulate and affect the proper functioning of these components.

### iii. Troubleshooting Guide:

- **Sensor Data Inconsistencies:** If you notice irregularities in sensor data, check the connections between the sensors and the Arduino Nano. Ensure that the sensors are powered correctly and that their pins are properly connected.
- **Motor Control Issues:** If the motor control mechanisms are not functioning as expected, inspect the connections, and confirm that the L298N motor driver is operating correctly. Check for loose connections or damaged wires.
- **Communication Problems:** In case of communication issues between the Arduino Nano and Raspberry Pi, inspect the wiring, verify that the correct communication protocol (e.g., serial, I2C, or SPI) is configured, and ensure the devices are properly powered.
- **User Interface Problems:** If the user interface on the Raspberry Pi experiences issues, check the software for any errors or crashes. Verify that the Raspberry Pi is adequately powered and connected to the network.

## 6. Safety Precautions

The following safety measures should be implemented to get effective functioning of whole system-

- **Electrical Safety:**  
Ensure that all electrical connections are secure and well-insulated to prevent short circuits.  
Disconnect power sources and turn off equipment when making changes to the system's hardware.
- **Water-Related Safety:**  
If your project involves water systems or irrigation, be cautious around water sources to prevent electrical hazards.  
Ensure that water management components are properly sealed to prevent leakage.
- **Sensor Safety:**  
Handle sensors, especially those that involve chemicals or sensitive components (e.g., the MQ135 sensor), with care and follow the manufacturer's guidelines for safe usage.
- **Fire Safety:**  
Place the system and components away from flammable materials and sources of heat or open flames.  
Be aware of the risk of overheating and provide adequate ventilation.
- **Data and Privacy:**  
Safeguard sensitive data and personal information collected by the system. Ensure that data transmission is secure.
- **User Interface Safety:**

Design the user interface with simplicity and clarity to prevent user errors that could impact the system's operation.

## **7. Appendix**

### Technical Specifications:

1. Arduino Nano:  
Microcontroller: ATmega328P  
Operating Voltage: 5V  
Digital Pins: 14  
Analog Pins: 8  
Flash Memory: 32 KB  
SRAM: 2 KB  
Clock Speed: 16 MHz  
Communication: UART, I2C, SPI  
Power Supply: USB or External
2. Raspberry Pi:  
Model: Raspberry pi 4 Model B  
Processor: Broadcom BCM2711  
RAM: 4GB  
Operating System: Raspberry Pi OS
3. L298N Motor Driver:  
Supply Voltage: 5V - 35V  
Maximum Current: 2A per channel  
Control Pins: IN1, IN2, IN3, IN4  
Motor Types: DC motors, stepper motors
4. MQ135 Air Quality Sensor:  
Detection Gases: Ammonia, Benzene, Alcohol, Smoke, CO2  
Operating Voltage: 5V  
Interface: Analog output
5. Soil Moisture Sensor:  
Operating Voltage: 3.3V - 5V  
Output: Analog and Digital (threshold adjustable)  
Detection Range: Adjustable
6. Ultrasonic Sensor (HC-SR04):  
Operating Voltage: 5V  
Detection Range: 2cm - 400cm  
Interface: Trigger and Echo pins

#### Data Sheets:

1. <https://docs.arduino.cc/resources/datasheets/ABX00027-datasheet.pdf>
2. <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf>
3. [https://ifttt.com/docs/api\\_reference](https://ifttt.com/docs/api_reference)

## **8. Conclusion**

In the ever-evolving field of agriculture, where efficiency, sustainability, and environmental responsibility are paramount, the Smart Agriculture System (Billing System) represents a significant stride forward. This project addresses the crucial challenges of water resource management and environmental preservation by offering real-time monitoring and intelligent control over water resources.

By integrating cutting-edge technologies such as the Raspberry Pi, Arduino Nano, and a range of sensors, we've created a robust system capable of monitoring soil moisture levels, air quality, and water levels, all while allowing farmers to take control of their water resources. The system, combined with the power of IFTTT alerts, ensures that farmers are always informed and can take action proactively.

Through the design and implementation of this project, we've not only empowered farmers with the tools they need for more efficient farming practices but have also made significant strides in combating issues such as stubble burning and environmental pollution. The system's billing functionality ensures fair resource allocation while encouraging responsible water usage.

While this project has achieved remarkable results, it is essential to acknowledge that technology continuously evolves. Future iterations could explore even more advanced sensors, machine learning algorithms for predictive analysis, and broader connectivity to form a comprehensive ecosystem for modern agriculture.

The Smart Agriculture System (Billing System) is a testament to the power of technology to transform traditional practices, promote sustainability, and safeguard the environment. It exemplifies the potential of innovation to address pressing global challenges, and it is a step towards a more responsible and efficient agricultural future.

## **9. References**

- <https://www.sciencedirect.com/science/article/pii/S2468227623000364>
- <https://ieeexplore.ieee.org/abstract/document/8239320>
- <https://ieeexplore.ieee.org/abstract/document/8434710>
- <https://www.academia.edu/download/56821290/IRJET-V5I4420.pdf>
- <https://smartertechnologies.com/guides/the-complete-guide-to-smart-agriculture-farming/#chapter-5>