Wireless Measurement Project

Enea Pedretti

November 2024

1 Introduction

Video description of the project: here. I authorized any use of this.

My Wireless Measurement project involves an automatic watering system for a plant. I used a capacitive water sensor to gather information about the soil's moisture percentage. After collecting data for nearly two months, I identified a threshold of around 60%, at which point a simulated pump should activate to increase the moisture level. All data is also transferred to the Arduino IoT Cloud platform, allowing remote monitoring of the plant's moisture levels.

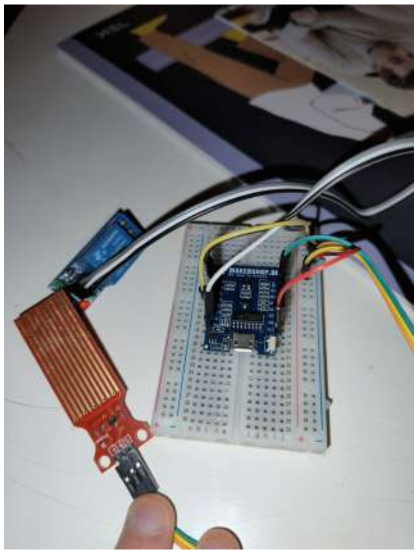
2 Hardware

- ESP 8266 micro controller
- Capacitive water sensor

- One relay to simulate the activation and deactivation of the pump

The wiring was done as the following:

- **3.3V Power (VCC)**: Connect the 3.3V pin on the ESP8266 to a common power row on the breadboard. Connect the VCC of both the water sensor and the relay to this same 3.3V row on the breadboard.
- **Ground (GND)**: Connect the GND pin on the ESP8266 to a common ground row on the breadboard. Connect the GND of both the water sensor and the relay to this same ground row.
- **Water Sensor Signal**: Connect the S (signal) pin of the water sensor to the analog pin A0 on the ESP8266.
- **Relay Control Signal**: Connect the IN (signal) pin of the relay to D1 (GPIO5) on the ESP8266.



3 Software

3.1 Offline Arduino IDE code (only for gathering data from the sensor

```
#define POWER_PIN  13  // GPIO13 corresponds to D7 on NodeMCU
#define SIGNAL_PIN A0  // Analog pin A0 for the sensor

int value = 0;  // variable to store the sensor value
int percentage = 0;  // variable to store the water percentage
```

```
void setup() {
  Serial.begin(9600);
  pinMode(POWER_PIN, OUTPUT);   // Configure GPIO13 (D7) as OUTPUT
  digitalWrite(POWER_PIN, LOW); // turn the sensor OFF initially
}

void loop() {
  digitalWrite(POWER_PIN, HIGH);  // turn the sensor ON
  delay(10);                      // wait 10 milliseconds
  value = analogRead(SIGNAL_PIN);  // read the analog value from the sensor
  digitalWrite(POWER_PIN, LOW);   // turn the sensor OFF

  // Map the sensor value to a percentage (0 to 100)
  percentage = map(value, 13, 696, 0, 100);

  // Clamp the percentage between 0 and 100
  percentage = constrain(percentage, 0, 100);

  Serial.print("Water percentage: ");
  Serial.print(percentage);
  Serial.println("%");

  delay(1000);  // wait 1 second before the next reading
}
```

3.2 Addition of the pump activation/deactivation

```
    #define POWER_PIN  13       // GPIO13 corresponds to D7 on NodeMCU
#define SIGNAL_PIN A0       // Analog pin A0 for the sensor
#define RELAY_PIN  5        // GPIO5 corresponds to D1 on NodeMCU

int value = 0;              // Variable to store the sensor value
int percentage = 0;         // Variable to store the water percentage
int threshold = 30;         // Threshold percentage to activate the relay

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

  pinMode(POWER_PIN, OUTPUT);    // Configure GPIO13 (D7) as OUTPUT for sensor power
  digitalWrite(POWER_PIN, LOW);  // Turn the sensor OFF initially

  pinMode(RELAY_PIN, OUTPUT);    // Configure GPIO5 (D1) as OUTPUT for relay control
  digitalWrite(RELAY_PIN, HIGH); // Set relay to OFF initially (active LOW)
}

void loop() {
  // Power on the sensor
  digitalWrite(POWER_PIN, HIGH);
  delay(10);                      // Small delay to stabilize the sensor reading

  // Read the sensor value
  value = analogRead(SIGNAL_PIN);
  digitalWrite(POWER_PIN, LOW);   // Power off the sensor to save energy

  // Map the sensor value to a percentage (0 to 100)
  percentage = map(value, 13, 696, 0, 100);

  // Clamp the percentage between 0 and 100
  percentage = constrain(percentage, 0, 100);

  Serial.print("Water percentage: ");
  Serial.print(percentage);
  Serial.println("%");

  // Relay control: activate if moisture percentage is below threshold
  if (percentage < threshold) {
    digitalWrite(RELAY_PIN, LOW);   // Turn relay ON (active LOW)
    Serial.println("Pump Activated");
  } else {
    digitalWrite(RELAY_PIN, HIGH);  // Turn relay OFF
    Serial.println("Pump Deactivated");
  }

  delay(1000);  // Wait 1 second before the next reading
}
```

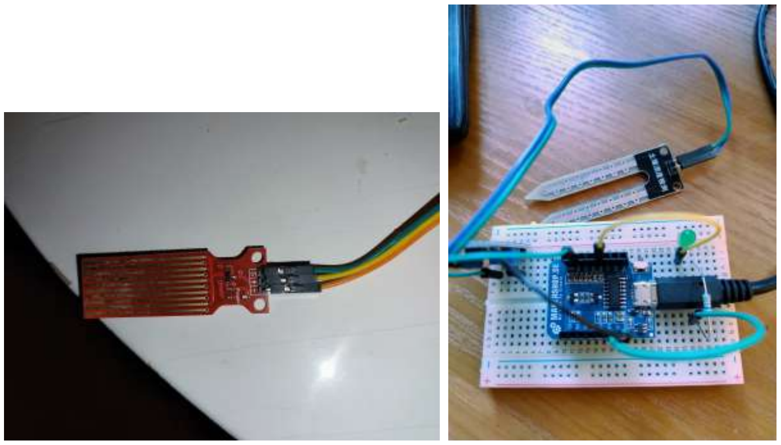
3.3 Arduino IoT Cloud sketch

```
    #include "thingProperties.h"

#define POWER_PIN  13       // GPIO13 corresponds to D7 on NodeMCU
#define SIGNAL_PIN A0       // Analog pin A0 for the sensor
#define RELAY_PIN  5        // GPIO5 corresponds to D1 on NodeMCU

int value = 0;              // Variable to store the sensor value
int percentage = 0;         // Variable to store the water percentage
int threshold = 60;         // Threshold percentage to activate the relay

void setup() {
  Serial.begin(9600);
  delay(1500);
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  pinMode(POWER_PIN, OUTPUT);
  digitalWrite(POWER_PIN, LOW);

  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, HIGH);
}

void loop() {
  ArduinoCloud.update();

  digitalWrite(POWER_PIN, HIGH);
  delay(10);
  value = analogRead(SIGNAL_PIN);
  digitalWrite(POWER_PIN, LOW);

  percentage = map(value, 13, 696, 0, 100);
  percentage = constrain(percentage, 0, 100);

  waterPercentage = percentage;  // Update cloud variable

  if (percentage < threshold) {
    digitalWrite(RELAY_PIN, LOW);
    relayState = true;  // Update cloud variable
  } else {
    digitalWrite(RELAY_PIN, HIGH);
    relayState = false; // Update cloud variable
  }

  delay(1000);
}

// Define the onWaterPercentageChange function
void onWaterPercentageChange() {
  // This function is called when the waterPercentage variable changes
}

// Define the onRelayStateChange function
void onRelayStateChange() {
  // This function is called when the relayState variable changes
}
```

4 Chronological summary

4.1 The choice fo the sensor

I received both a capacitive (left) and a resistive water sensor (right). I decided not to use the resistive sensor because, over time, its electrodes may corrode due to the effect of electrode polarization. This happens not only because they are in contact with the soil but also because the DC current flowing through them causes electrolysis, which leads to sensor degradation. The sensor consists of two probes used to measure the volumetric water content. The two probes allow current to pass through the soil, which enables the sensor to measure the resistance value and, in turn, determine the moisture level. When there is more water, the soil conducts electricity better, which means there will be less resistance, and the moisture level will be higher. Dry soil, on the other hand, conducts electricity poorly. So, when there is less water, the soil conducts less electricity, resulting in higher resistance and a lower moisture level. The chosen sensor measures soil moisture levels by capacitive sensing rather than resistive sensing, due to the change of dielectric. It prevents corrosion because it is made from corrosion-resistant material, which gives it a long service life.

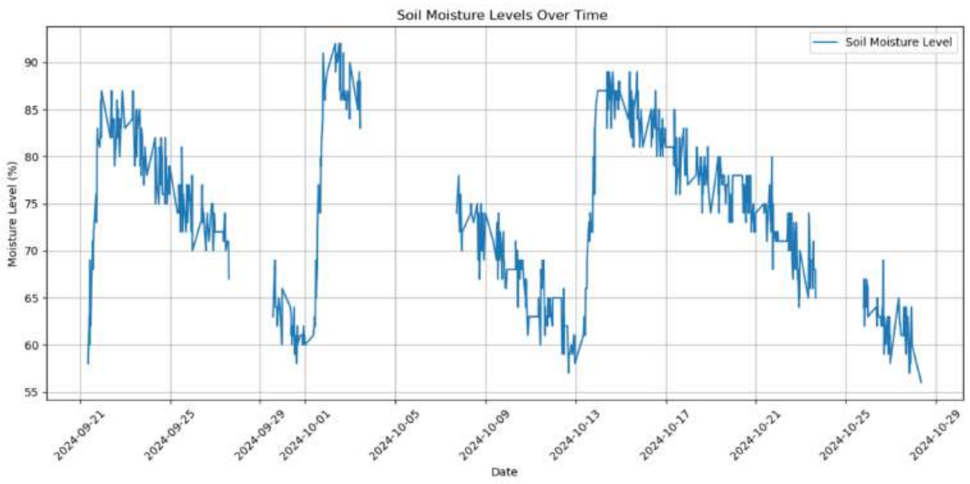


4.2 Calibration of the Sensor

To properly calibrate the sensor, I aimed to generate as many data points as possible on a chart using known water concentrations. Unfortunately, creating different known concentrations of water would require a substance like hexane. Therefore, I could only use 0% and 100% concentrations. I took readings from the sensor in the air (0% moisture) and in a glass of water (100% moisture). Since it's a capacitive sensor, and capacitors typically show linear behavior, I assumed a linear response.

4.3 Data Gathering

I manually recorded the soil moisture levels at random intervals over a period of two months. Here is the resulting chart:

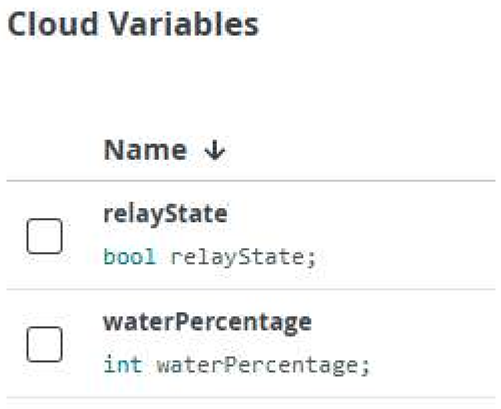


Eventually, I realized I could connect the micro controller to a phone charger to gather data remotely, as I had left my laptop in my apartment to power the micro controller. I monitored two plants: an African violet and a carrot top placed in soil to encourage growth. Unfortunately, the carrot developed mold within a few days.

I decided to water the plant when the soil felt sufficiently dry to the touch. After three watering cycles, as shown in the chart, I chose 60% as the threshold for activating the pump to water the soil. Also from the chart, we can notice that water needs some days to fully diffuse into the soil. After that, the moisture level decreases almost linearly for several weeks until it reaches the threshold. To ensure accurate readings, I decided not to move the sensor from its initial position, as multiple sensors may be needed to estimate the actual water concentration in the soil accurately, given that water may not diffuse uniformly. By placing the sensor in one location (near the roots, as close to the center as possible), I get an accurate estimate of the moisture in that specific region of the soil.

4.4 Arduino IoT Cloud

After creating a ”thing” on Arduino IoT, I added properties as in picture:



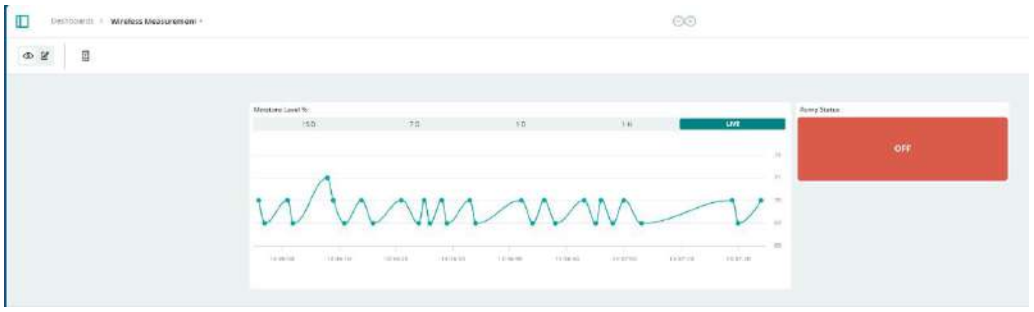
Cloud Variables

Name ↓

relayState
```
bool relayState;
```

waterPercentage
```
int waterPercentage;
```

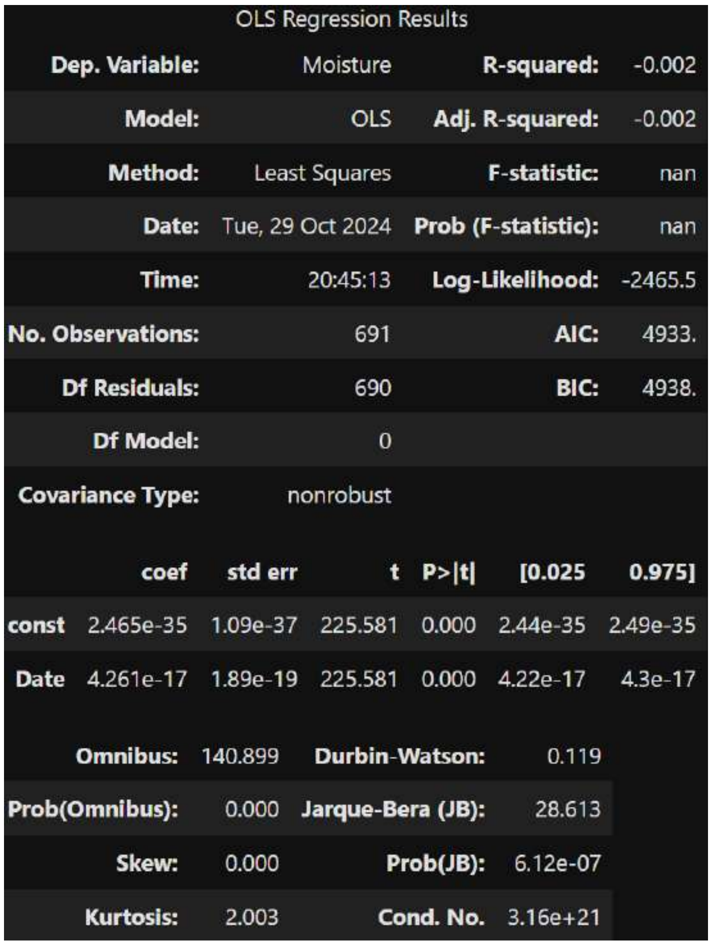
I specified the device specifications and uploaded the sketch code, though it was challenging. I encountered several issues, particularly with the Wi-Fi connection due to my TP-Link access point. After consulting various online forums, I discovered that I should avoid using the official TP-Link app and instead connect to the access point through my IP address. While not very convenient, this approach eventually worked. I also ran into some issues with the sketch, but after multiple attempts, LLMs provided the help I needed. After troubleshooting, I created a dashboard, although I cannot share it dynamically as it is behind a paywall. Here’s a screenshot:



4.5 Machine Learning for predicting water needs

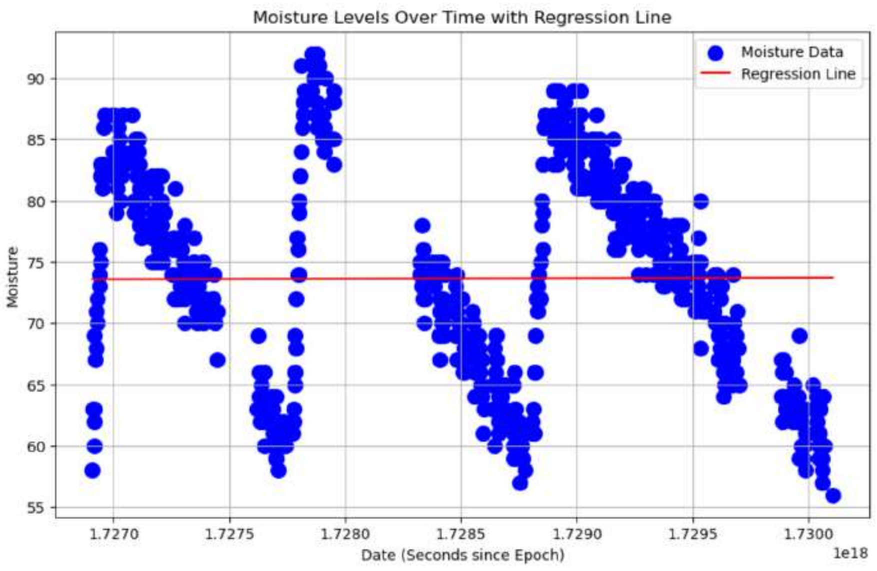
The following is preliminary, as I used a linear regression model that doesn't clearly fit the gathered data. For informational purposes, I am uploading the code:

```
df = pd.read_csv('soil_moisture_data.csv').
df['Date'] = pd.to_datetime(df['Date']).astype(np.int64)
df['Date'] = df['Date'].astype('int64')
df = df.dropna()
import statsmodels.api as sm
X = df['Date']
X = sm.add_constant(X)
y = df['Moisture']

model = sm.OLS(y, X)
result = model.fit()
result.summary()
```



```
                        OLS Regression Results
Dep. Variable:        Moisture              R-squared:        -0.002
Model:                OLS                   Adj. R-squared:   -0.002
Method:               Least Squares         F-statistic:      nan
Date:                 Tue, 29 Oct 2024      Prob (F-statistic): nan
Time:                 20:45:13              Log-Likelihood:   -2465.5
No. Observations:     691                   AIC:              4933.
Df Residuals:         690                   BIC:              4938.
Df Model:             0
Covariance Type:      nonrobust

         coef       std err    t        P>|t|   [0.025     0.975]
const    2.465e-35  1.09e-37   225.581  0.000   2.44e-35   2.49e-35
Date     4.261e-17  1.89e-19   225.581  0.000   4.22e-17   4.3e-17

Omnibus:          140.899   Durbin-Watson:       0.119
Prob(Omnibus):    0.000     Jarque-Bera (JB):    28.613
Skew:             0.000     Prob(JB):            6.12e-07
Kurtosis:         2.003     Cond. No.            3.16e+21
```

```
plt.figure(figsize=(10, 6))
plt.scatter(df['Date'], df['Moisture'], color='blue', label='Moisture Data', s=100)

# Plotting the regression line
plt.plot(df['Date'], result.fittedvalues, color='red', label='Regression Line')

# Formatting the plot
plt.title('Moisture Levels Over Time with Regression Line')
plt.xlabel('Date (Seconds since Epoch)')
plt.ylabel('Moisture')
plt.legend()
plt.grid()
plt.show()
```



I didn't proceed further because, based on the current trend line, the next watering would be predicted to occur in roughly a year, as the slope of the line is nearly zero.

5 Conclusions

As you can see from the project video description, I successfully implemented the project as intended, except for predictions using machine learning, as I need to learn more models. The data is now available remotely through Arduino IoT Cloud, and the pump activates and deactivates based on the soil moisture level. A future enhancement could involve implementing this project in my father's garden, using multiple sensors and a larger pump for automated watering.

No plant has been harmed in any way