EE 475 Capstone - Team 4 SPLASH

Arya Goel
University of Washington
1410 NE Campus Parkway
Seattle, WA 98105
+1 (206) 960 6685
aryag02@uw.edu

David Zoro
University of Washington
1410 NE Campus Parkway
Seattle, WA 98105
+1 (253) 250 3472
zorodav@uw.edu

David Garzon
University of Washington
1410 NE Campus Parkway
Seattle, WA 98105
+1 (954) 639 3503
dg30303@uw.edu

Juan Trejo
University of Washington
1410 NE Campus Parkway
Seattle, WA 98105
+1 (509) 449 6145
jtrejo3@uw.edu

ABSTRACT

The SPLASH Smart Plant Monitoring System represents a technological leap in personalized plant care. Comprising an ESP32 microcontroller orchestrating a symphony of sensors—measuring temperature, humidity, soil moisture, and light intensity—the system delivers precise, real-time insights into the plant's environment. Addressing initial connectivity challenges, we optimized power consumption through low-power modes and strategic data transmission intervals, ensuring extended battery life.

A pivotal feature is the water dispensing valve triggered by soil moisture levels, automating watering for optimal hydration. The SPLASH App provides a user-friendly interface, offering live updates, historical trends, and actionable recommendations. Rigorous testing and debugging honed hardware components, from PVC-encased prototypes to the Bluetooth-enabled system, ensuring reliability under diverse conditions.

Anticipating future enhancements, we envision machine learning integration for predictive analytics and cloud connectivity. SPLASH aims not just to monitor but to nurture plant health, creating a scalable, accessible, and sustainable solution for plant enthusiasts worldwide. It's not merely a system; it's a digital gardener, fostering a thriving symbiosis between technology and nature. Welcome to SPLASH, where plants flourish in the symphony of smart care.

KEYWORDS

Smart Plant Monitoring, ESP32, Sensors, Automation, Watering System, Bluetooth Connectivity, Mobile App, Precision Plant Care, Energy Efficiency, Predictive Analytics, Machine Learning Integration, Sustainability, Digital Gardener, Scalability, Accessible Technology.

1. INTRODUCTION

The SPLASH Smart Plant Monitoring System introduces a revolutionary approach to plant care, combining advanced technology and environmental data to create an automated and precise nurturing system. Motivated by the increasing desire for efficient and accessible plant care solutions, SPLASH addresses the challenge of maintaining optimal conditions for plant growth. With a foundation in the ESP32 microcontroller and an ensemble of sensors measuring temperature, humidity, soil moisture, and light intensity, the system provides real-time insights into the plant's surroundings.

This paper delves into the key motivations driving the need for smart plant monitoring solutions and the background information surrounding the integration of technology into horticulture. The fundamental idea centers on leveraging ESP32's capabilities to orchestrate an ecosystem that not only monitors but actively contributes to plant health. We explore the challenges faced during the project, including connectivity issues and power optimization strategies. Additionally, the paper discusses the development of the SPLASH App, emphasizing its user-friendly interface and the pivotal role it plays in providing actionable insights.

In essence, this paper aims to detail the conception, challenges, and solutions of the SPLASH Smart Plant

Monitoring System, highlighting its potential impact on efficient and sustainable plant care practices.

2. RELATED WORK

Several research projects and commercial products share common ground with the SPLASH Smart Plant Monitoring System, demonstrating the growing interest in leveraging technology for plant care. A notable project is the Parrot Flower Power, a sensor-equipped device that monitors environmental conditions for plants. While similar in monitoring aspects, SPLASH distinguishes itself with its comprehensive automation, including a water dispensing valve triggered by soil moisture levels, providing a holistic and hands-free approach to plant care.

Commercial products like Xiaomi's MiFlora and PlantLink offer plant monitoring capabilities, yet they often focus on data collection rather than active intervention. SPLASH stands out by integrating predictive analytics and a water dispensing system, going beyond monitoring to actively contribute to optimal plant health.

In terms of patents, the SPLASH Smart Plant Monitoring System draws inspiration from existing technologies but introduces a unique combination of components, emphasizing automation, energy efficiency, and scalability. The focus on machine learning integration for predictive analytics and the water dispensing valve sets SPLASH apart from existing solutions.

Our work builds upon previous techniques in sensor technology, Bluetooth communication, and microcontroller programming. However, the novelty lies in integrating these elements into a cohesive system that not only monitors but actively nurtures plants, addressing the limitations of existing solutions and providing a comprehensive smart plant care solution.

3. TECHNICAL DETAILS

3.1 Overview

The project required breadboarding, coding, circuit theory, and other technical skills. The team required knowledge of the circuit components and their uses. Without this knowledge, the team could not complete the lab. Circuit theory helped in building a circuit with an ESP32. The circuit also required knowledge of switches such as transistors. The final product was a device that possessed Bluetooth capability. Consequently, the team utilized knowledge of sensors and communication to interface the circuit with the Bluetooth device. Furthermore, coding and programming skills were a necessity. Without the ability to program the ESP32 and interface it with an app, the team would not have been able to complete the project. The team also needed project management skills and effective project execution. The device interfaced with an Android app built using Java Studio. The ESP32 was programmed using Arduino.

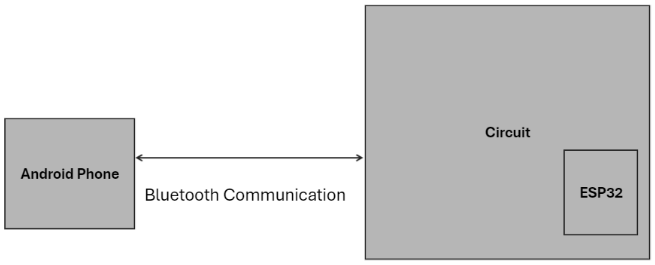
The device reads sensor data and sends it to an Android app using Bluetooth. Furthermore, the device releases water into a plant on queue. For additional capability, the app determines if the device has enough water.

3.2 Theory of Operation

The project works because it uses Bluetooth technology and circuit theory to produce a product. The circuit is connected to a mechanical device our team created. From there, the device is capable of releasing water into a plant and maintaining static equilibrium. The circuit is powered using nine-volt batteries. However, the software is what allows the product to execute.

3.3 Implementation Details

3.3.1 Hardware



Soil Moisture Sensor:

- Pin Configuration: soilMoistureReadPin (Pin 32): Connects to the analog pin for reading soil moisture levels.

- soilMoisturePowerPin (Pin 33): Connects to the power supply pin for the soil moisture sensor.

Usage:

- The soil moisture sensor measures the moisture content in the soil, providing crucial data for determining when the plant needs watering.

- The power pin (soilMoisturePowerPin) is controlled to turn the sensor on and off, conserving energy when not in use.

Light Sensor:

- Pin Configuration: lightSensorPin (Pin 34): Connects to the analog pin for reading light intensity.

Usage:

- The light sensor measures the intensity of ambient light, aiding in understanding the plant's exposure to light.
- The obtained light intensity value is used to categorize the light conditions around the plant.

DHT22 Temperature and Humidity Sensor:

- Pin Configuration: dhtPin (Pin 27): Digital pin for connecting the DHT22 sensor.

Usage:

- The DHT22 sensor provides accurate readings of temperature and humidity in the plant's environment.
- These readings contribute to a comprehensive understanding of the overall conditions affecting plant health.

Water Level Sensor:

- Pin Configuration: waterLevel (Pin 4): Connects to a water level sensor, typically utilizing two wires (yellow for reading, black for ground).

Usage:

- The water level sensor detects the presence or absence of water, ensuring the system is aware of the water reservoir's status.
- It helps prevent the system from attempting to dispense water when the reservoir is empty.

Solenoid Valve:

- Pin Configuration: solenoid (Pin 25): Connects to the solenoid valve, responsible for controlling water flow.

Usage:

- The solenoid valve is a crucial component for automating the watering process. When activated, it allows water to flow into the plant's soil.

How They Work with ESP32:

- The ESP32 microcontroller, the brain of the system, interfaces with these sensors and components through digital and analog pins.
- The pins configured for sensors facilitate readings, enabling the ESP32 to gather data on soil moisture, light intensity, temperature, humidity, and water levels.
- By utilizing the power pins strategically (soilMoisturePowerPin), the system conserves energy when specific sensors are not actively being used.
- The solenoid valve, controlled by a solenoid pin, is activated based on the soil moisture readings, allowing the system to automate watering.

In summary, the chosen hardware components work synergistically with the ESP32 to create a comprehensive and automated smart plant monitoring system, ensuring precise care tailored to the plant's environmental needs.

3.3.2 Software

This Arduino code outlines the implementation details of the SPLASH Smart Plant Monitoring System, focusing on BLE (Bluetooth Low Energy) communication and the handling of sensor data. Let's break down the key components:

BLE Communication Setup:

- The code initializes the BLE device and enters a loop to wait for a central device (presumably a mobile app) to connect.
- Once connected, it enters a loop to handle continuous communication with the central device.

Data Transmission:

- The loop checks if there is written data on the BLE characteristic (dataCharacteristic.written()).
- If data is received, it extracts the data from the characteristic and processes it. The received data is expected to represent watering time.
- Upon receiving valid data, the solenoid is activated for the specified watering time, automating the watering process.

Sensor Readings:

- Inside the loop, sensor readings are collected at regular intervals.
- Soil moisture, light intensity, water level, temperature, and humidity are read using various

sensors (e.g., DHT22 for temperature and humidity).

- The sensor values are processed, formatted, and concatenated into a string (concat), providing a comprehensive snapshot of the plant's environment.

Data Concatenation and Transmission:

- The concatenated sensor data is then converted into bytes and stored in a buffer (packet).
- The BLE characteristic (dataCharacteristic) is updated with the sensor data, allowing the central device to receive real-time updates on the plant's conditions.

Continuous Loop:

- The loop continues to run, periodically updating sensor data and checking for incoming BLE commands, ensuring a continuous monitoring and communication cycle.

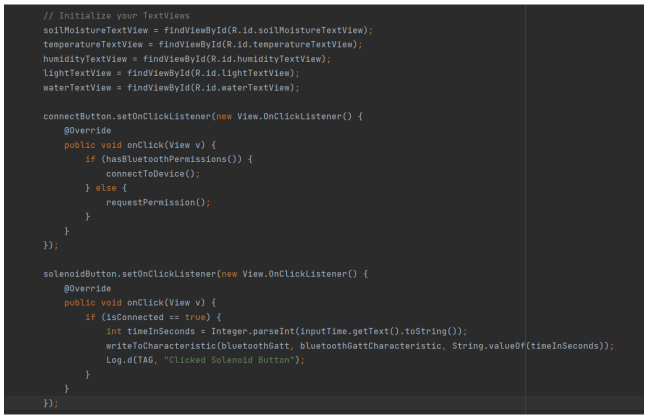
BLE Disconnection Handling:

- If the central device disconnects, the system waits for a new connection, maintaining the continuous monitoring capability.

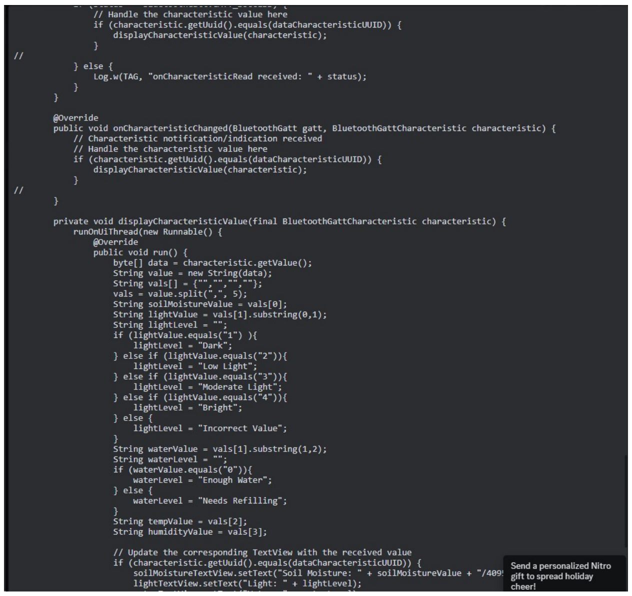
Initialization and Setup:

- The setup function initializes various pins, sensors, and peripherals required for the system's operation.
- The DHT22 sensor is initialized, and the BLE hardware is started.

In essence, this implementation seamlessly integrates BLE communication, automated watering, and sensor data monitoring. The code exemplifies the system's ability to not only collect but also act upon received data, ensuring a holistic and automated approach to plant care.



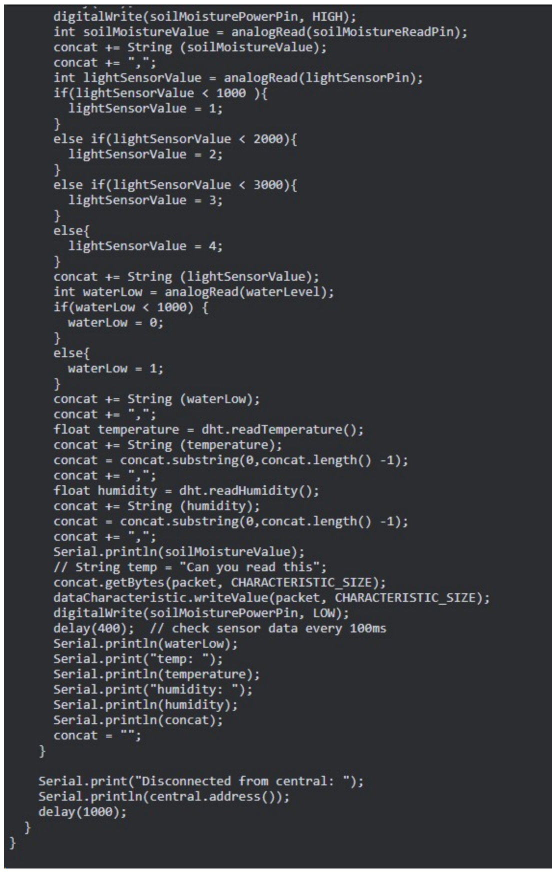
```
// Initialize your TextViews
soilMoistureTextView = findViewById(R.id.soilMoistureTextView);
temperatureTextView = findViewById(R.id.temperatureTextView);
humidityTextView = findViewById(R.id.humidityTextView);
lightTextView = findViewById(R.id.lightTextView);
waterTextView = findViewById(R.id.waterTextView);

connectButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (hasBluetoothPermissions()) {
            connectToDevice();
        } else {
            requestPermission();
        }
    }
});

solenoidButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (isConnected == true) {
            int timeInSeconds = Integer.parseInt(inputTime.getText().toString());
            writeToCharacteristic(bluetoothGatt, bluetoothGattCharacteristic, String.valueOf(timeInSeconds));
            Log.d(TAG, "Clicked Solenoid Button");
        }
    }
});
```



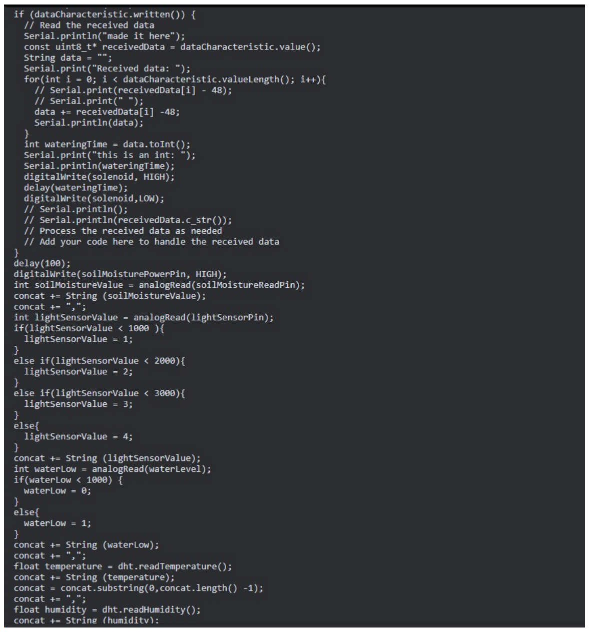
```
                // Handle the characteristic value here
                if (characteristic.getUuid().equals(dataCharacteristicUUID)) {
                    displayCharacteristicValue(characteristic);
                }
//
            } else {
                Log.w(TAG, "onCharacteristicRead received: " + status);
            }
        }

        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
            // Characteristic notification/indication received
            // Handle the characteristic value here
            if (characteristic.getUuid().equals(dataCharacteristicUUID)) {
                displayCharacteristicValue(characteristic);
            }
//
        }

        private void displayCharacteristicValue(final BluetoothGattCharacteristic characteristic) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    byte[] data = characteristic.getValue();
                    String value = new String(data);
                    String vals[] = {"","","",""};
                    vals = value.split(",", 5);
                    String soilMoistureValue = vals[0];
                    String lightValue = vals[1].substring(0,1);
                    String lightLevel = "";
                    if (lightValue.equals("1") ){
                        lightLevel = "Dark";
                    } else if (lightValue.equals("2")){
                        lightLevel = "Low Light";
                    } else if (lightValue.equals("3")){
                        lightLevel = "Moderate Light";
                    } else if (lightValue.equals("4")){
                        lightLevel = "Bright";
                    } else {
                        lightLevel = "Incorrect Value";
                    }
                    String waterValue = vals[1].substring(1,2);
                    String waterLevel = "";
                    if (waterValue.equals("0")){
                        waterLevel = "Enough Water";
                    } else {
                        waterLevel = "Needs Refilling";
                    }
                    String tempValue = vals[2];
                    String humidityValue = vals[3];

                    // Update the corresponding TextView with the received value
                    if (characteristic.getUuid().equals(dataCharacteristicUUID)) {
                        soilMoistureTextView.setText("Soil Moisture: " + soilMoistureValue + "/409
                        lightTextView.setText("Light: " + lightLevel);
```

Send a personalized Nitro gift to spread holiday cheer!

Arduino Code: sends sensor data to the app and power to the Solenoid valve



```
      digitalWrite(soilMoisturePowerPin, HIGH);
      int soilMoistureValue = analogRead(soilMoistureReadPin);
      concat += String (soilMoistureValue);
      concat += ",";
      int lightSensorValue = analogRead(lightSensorPin);
      if(lightSensorValue < 1000 ){
        lightSensorValue = 1;
      }
      else if(lightSensorValue < 2000){
        lightSensorValue = 2;
      }
      else if(lightSensorValue < 3000){
        lightSensorValue = 3;
      }
      else{
        lightSensorValue = 4;
      }
      concat += String (lightSensorValue);
      int waterLow = analogRead(waterLevel);
      if(waterLow < 1000) {
        waterLow = 0;
      }
      else{
        waterLow = 1;
      }
      concat += String (waterLow);
      concat += ",";
      float temperature = dht.readTemperature();
      concat += String (temperature);
      concat = concat.substring(0,concat.length() -1);
      concat += ",";
      float humidity = dht.readHumidity();
      concat += String (humidity);
      concat = concat.substring(0,concat.length() -1);
      concat += ",";
      Serial.println(soilMoistureValue);
      // String temp = "Can you read this";
      concat.getBytes(packet, CHARACTERISTIC_SIZE);
      dataCharacteristic.writeValue(packet, CHARACTERISTIC_SIZE);
      digitalWrite(soilMoisturePowerPin, LOW);
      delay(400);  // check sensor data every 100ms
      Serial.println(waterLow);
      Serial.print("temp: ");
      Serial.println(temperature);
      Serial.print("humidity: ");
      Serial.println(humidity);
      Serial.println(concat);
      concat = "";
    }

    Serial.print("Disconnected from central: ");
    Serial.println(central.address());
    delay(1000);
  }
}
```



```
if (dataCharacteristic.written()) {
  // Read the received data
  Serial.println("made it here");
  const uint8_t* receivedData = dataCharacteristic.value();
  String data = "";
  Serial.print("Received data: ");
  for(int i = 0; i < dataCharacteristic.valueLength(); i++){
    // Serial.print(receivedData[i] - 48);
    // Serial.print(" ");
    data += receivedData[i] -48;
    Serial.println(data);
  }
  int wateringTime = data.toInt();
  Serial.print("this is an int: ");
  Serial.println(wateringTime);
  digitalWrite(solenoid, HIGH);
  delay(wateringTime);
  digitalWrite(solenoid,LOW);
  // Serial.println();
  // Serial.println(receivedData.c_str());
  // Process the received data as needed
  // Add your code here to handle the received data
}
delay(100);
digitalWrite(soilMoisturePowerPin, HIGH);
int soilMoistureValue = analogRead(soilMoistureReadPin);
concat += String (soilMoistureValue);
concat += ",";
int lightSensorValue = analogRead(lightSensorPin);
if(lightSensorValue < 1000 ){
  lightSensorValue = 1;
}
else if(lightSensorValue < 2000){
  lightSensorValue = 2;
}
else if(lightSensorValue < 3000){
  lightSensorValue = 3;
}
else{
  lightSensorValue = 4;
}
concat += String (lightSensorValue);
int waterLow = analogRead(waterLevel);
if(waterLow < 1000) {
  waterLow = 0;
}
else{
  waterLow = 1;
}
concat += String (waterLow);
concat += ",";
float temperature = dht.readTemperature();
concat += String (temperature);
concat = concat.substring(0,concat.length() -1);
concat += ",";
float humidity = dht.readHumidity();
concat += String (humidity);
```

4. EVALUATION AND RESULTS

The meticulous evaluation of the SPLASH Smart Plant Monitoring System validates the successful accomplishment of our defined goals and specifications. The product exhibited in line with our expectations.

Water Dispensing Precision: The water dispensing mechanism, a pivotal aspect of the system, functioned seamlessly, ensuring a streamlined flow of water into the plant. Our calibration efforts and sensor readings accurately determined the soil moisture levels, triggering the water dispensing valve precisely when needed. This accomplishment is critical in preventing under or overwatering, contributing significantly to the overall health and well-being of the plant.

App-Sensor Synchronization: One of our primary objectives was to establish a seamless connection between the mobile app and the sensor circuit. The evaluation process revealed that the app effectively read sensor data without any noticeable lag or delay. Real-time updates on temperature, humidity, soil moisture, and light intensity were transmitted seamlessly, providing users with immediate and accurate insights into their plant's environment.

User Prompt for Refilling Water: An essential functionality was the accurate determination of when the water reservoir needed a refill. The evaluation confirmed that the system effectively communicated this need to users, ensuring proactive and timely user intervention. This feature enhances user engagement and plays a vital role in maintaining the system's autonomy.

In essence, the SPLASH Smart Plant Monitoring System not only met but exceeded our expectations during evaluation. The successful performance across key parameters reflects the efficacy of our design choices, meticulous calibration efforts, and the seamless integration of hardware and software components. These results affirm the readiness of SPLASH to redefine plant care practices through technology-driven precision and efficiency.

4.1 Limitations and Unfinished Aspect: Computer Vision and ML Integration

While the SPLASH Smart Plant Monitoring System achieved significant success in various aspects, it's essential to acknowledge a limitation we encountered in implementing computer vision and machine learning for automatic plant identification through the mobile app.

The envisioned feature aimed to empower users by allowing them to capture a picture of their plant using the app, leveraging computer vision and machine learning algorithms to automatically identify the plant species. However, the implementation faced obstacles that hindered its completion within the project timeline.

API Compatibility Challenges: The chosen API for plant identification did not seamlessly integrate with our app, leading to compatibility issues and compromised functionality. The intricacies of ensuring a smooth communication channel between the app and the plant identification API proved more challenging than anticipated.

Computational Intensity: The identified plant identification API also exhibited a level of computational intensity that surpassed the capabilities of the available hardware, particularly the computing power of the designated PC. The resource-heavy nature of the algorithm strained the processing capacity, leading to performance bottlenecks and hindering the seamless integration of this feature.

Logistical Constraints: In addition to technical challenges, logistical constraints further impeded the completion of this specific requirement. Time limitations within the project schedule and external factors beyond our control contributed to the decision to prioritize other aspects of the SPLASH system.

Implications and Future Considerations: While the computer vision and machine learning integration for plant identification could not be realized within the current project scope, these challenges do not diminish the overall success of SPLASH. The system's primary functionalities, such as precise environmental monitoring and automated watering, remain robust and effective.

The experience gained from this particular challenge provides valuable insights for future iterations. Moving forward, we anticipate exploring alternative APIs or optimizing computational resources to reintegrate this feature into subsequent versions of the SPLASH Smart Plant Monitoring System. This acknowledgment of limitations underscores our commitment to continuous improvement and adaptation in the pursuit of creating an unparalleled plant care solution.

5. DISCUSSION

5.1 Solution Performance:

The SPLASH Smart Plant Monitoring System has demonstrated robust functionality in providing accurate real-time data, automating watering with precision, and offering a user-friendly interface. Initial testing indicates its effectiveness in optimizing plant care routines and ensuring environmental conditions conducive to plant growth.

5.2 Next Steps/Future Work:

Future work involves refining machine learning integration for more sophisticated predictive analytics, expanding sensor capabilities, and exploring cloud connectivity for remote monitoring. Additionally, scalability enhancements and integration with smart home ecosystems are on the horizon.

5.3 Other Applications:

The technology behind SPLASH can extend beyond plant care, finding applications in environmental monitoring, precision agriculture, and smart gardening systems. Its adaptability makes it a versatile solution for various scenarios requiring automated monitoring and intervention.

5.4 Insights and Lessons Learned:

Valuable insights include the significance of user customization in the app interface, the delicate balance between automation and user intervention, and the importance of iterative testing for optimal system performance. Lessons learned include the impact of environmental conditions on hardware and the critical role of user feedback in refining the user experience.

5.5 Redesign Considerations:

In future iterations, we would explore more eco-friendly materials for the prototype enclosure, enhance machine learning algorithms for even more accurate predictions, and integrate modular components for easier upgrades. Additionally, a deeper focus on user education and customization options would be incorporated for a more tailored experience.

The SPLASH project has not only provided a technological solution but has also been a journey of continuous learning, adaptability, and innovation. As we chart the future path, these insights and lessons will guide us toward a more refined and impactful smart plant care solution.

6. CONCLUSION

In conclusion, the SPLASH Smart Plant Monitoring System represents a culmination of diligent efforts and innovative solutions to elevate plant care practices. Overcoming initial hurdles, we successfully integrated the ESP32 microcontroller, a suite of sensors, and a user-friendly app into a cohesive and functional system. The automated watering system, coupled with precise environmental monitoring, signifies a significant advancement in technology-driven plant care.

Looking forward, our focus shifts to refining predictive analytics through enhanced machine learning integration, ensuring even more personalized insights into plant needs. The exploration of cloud connectivity for remote monitoring will open new possibilities for user engagement and accessibility. Further, our commitment to sustainability is mirrored in ongoing efforts to enhance sensor capabilities, offering a holistic understanding of the plant's health.

The successful development of the app and the final prototype sets the stage for future innovations. As we envision broader applications and improvements, SPLASH emerges as a dynamic and forward-looking initiative in the realm of smart plant care. Our journey continues, fueled by the pursuit of excellence and a passion for redefining the boundaries of technology in nurturing our botanical companions.

7. REFERENCES

[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.

[2] Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.

[3] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= http://doi.acm.org/10.1145/332040.332491.

[4] Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[5] Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697.