Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

First Semester Report

Smart Watering System with Internet of Things

MathWorks

by

Team 29

Team Members

Ines Saavedra Villafruela inessv@bu.edu
Jordan Remar jremar@bu.edu
Kaede Kawata kkawata@bu.edu
Karin Luna kaluna@bu.edu
Nyah Madison nyahbm@bu.edu

Table of Contents

Executive Summary

Smart Watering System with the Internet of Things
Team 29

While 71% of the Earth's surface area may be covered by water, fresh water only accounts for 3.5% of the total supply. Furthermore, about half of this fresh water is permanently frozen in glaciers, permafrost, and snow[1]. Given that agriculture is a major consumer of this limited supply, any ways of limiting unnecessary water usage is vital. The goal of the project is to create a small, modular system that monitors the environmental conditions for a given crop, and uses machine learning to predict future water needs based on past data for any given location. Given that water shortages affect all socioeconomic classes, affordability will play a decisive role in the design of the system.

The system will make use of off-the-shelf sensors to monitor the environmental conditions (e.g. temperature, humidity, barometric pressure, etc). In order to minimize a given system's cost, the particular sensors used by a given system should be easily tailored to a given crop. An ESP32 will act as the central microcontroller to monitor the various sensors. Ideally, most sensors should communicate along a shared bus by using a predetermined protocol such as i²c. Such an approach would both simplify the hardware and software design. These various data points will then be parsed by an ESP32, and be sent to an offsite data repository. (Jordan Remar)

[1] Data provided by the USGS https://www.usgs.gov/special-topics/water-science-school/science/how-much-water-there-earth

1.0 Introduction

Water scarcity is a worsening problem around the world due to climate change and water overconsumption. The majority of the world's water usage is in agriculture. Farmers are in great need of low-cost systems that can help optimize water usage, allowing them to grow their crops and save water simultaneously.

Our goal is to use cutting-edge technology to help farmers optimize water consumption. To do this, We are building a low-cost system that will use the Internet of Things (IoT) to collect related data through sensors and push this data online to be analyzed. Based on the data collected, the system will alert the customer about when to water the crops and by how much.

As a team, we will build the product with three main parts: the hardware, software, and data analysis. The hardware will collect information about the environment. The data collected will be analyzed to decide when the client should water their crops. The software runs throughout the system from pushing the data to the network to be analyzed to showcasing the decision about when the user should water the plants on an app.

One feature we are aiming for is that our system should use low-cost and sustainable hardware. This enables more farmers with varying amounts of income to optimize water usage and promotes sustainability. (Nyah Madison)

2.0 Concept Development

Since the system will predict when to water crops, the sensors should measure environmental data, such as temperature, humidity, soil moisture and barometric pressure. This data needs to be stored in a database so that different platforms can access it. ThingSpeak is the customer's preferred IoT analytics platform. Thus, analysis of the data and visualizations will be done with ThingSpeak. The collected data will be uploaded there and to Firebase.

The end-user also needs a notification system so that they know what our system has decided. Other useful information required include weather forecast data and recommendations of what crop to grow based on a predictive model. All hardware used must be low-cost.

There will be two parts to this project. The first is the physical product that will measure the environmental data. The second part is the app that will allow the user to input crop parameters, display collected data, and receive a notification about when to water the crops.

The hardware includes the solar panels, sensors, microcontroller, and circuit contained in an enclosure. The sensors will collect the seven pieces of information needed. The microcontroller will capture the data and push it to the database. The circuit connects all the elements together and powers them. To make the system sustainable, it is going to be rechargeable by solar energy using solar panels that will generate enough electricity to have current in the microcontroller. This current will allow the system to work without burning the sensors.

The software includes code for the microcontroller as well as the back-end and front-end of the app. The purpose of the back-end is to store the data collected so that the analytics platform and app can retrieve necessary data. It will consist of a database for environmental data and user login authentication. The front end of the app will be a user-friendly interface that allows information input about the crops that may be needed, such as crop type. Buttons to select crop type and text size will be big to help the user more easily interact with the app. Lastly, data analysis will predict when the crops should be watered. The data will move to ThingSpeak via the microcontroller. ThingSpeak is connected to MATLAB, allowing easy visualizations and analysis without coding. Using ThingSpeak as an IoT platform, a predictive model will decide if the crops should be watered. Once it is decided that the crops need watering, the app will display the push notification.

We are starting to build the product by focusing on building the data pipeline that the product will use. The pipeline begins at the hardware level with the sensors collecting data. The data we collect will go to two places - a database and an IoT analytics platform. From the IoT platform, the data will be analyzed and the system will predict whether or not it is best for the farmer to water their crops now. If the crops need watering, that result will be pushed to the database where the app can retrieve that information and notify the user to water their crops. This solves the problem of water overconsumption by telling the farmers when to water so that they only water when necessary but still optimize crop growth.

Many of the elements that will be used in our product are the preferences of the customer, such as ThingSpeak and MATLAB. However, since the project is open-ended, there are other elements that we decided to use after brainstorming other ideas. We decided to start by measuring four pieces of data because that data will give us basic information about the soil environment and weather. Our design is modular so that we can add more sensors later, if needed, without difficulty. For the back-end, we chose to use a Backend-as-a-Service. Two services that could have been used include Google Firebase and Parse Platform. We decided to use Google Firebase so that we can have a database and user login authentication without building our own server and deploying it with a cloud service. For our notification system, we decided to use React Native to build the app due to previous experience and efficiency of Expo Go. We chose to use an app so that we can collect other information from the end-user, such as type of crop, and notify the user within the same module. For the prediction, ThingSpeak is going to be the IoT platform used. Once the hardware device is connected to the channels in ThingSpeak, the data can be analyzed and visualized. There are going to be limits in the features measured by the hardware so that when a parameter is exceeded a notification will be sent to the user. ThingSpeak allows you to send an SMS, Tweet or a notification to a React App. (Ines Villafruela)

3.0 System Description

The physical design of the product consists of the primary microcontroller, the sensors that will measure the environmental conditions, and the circuit to supply and store power to the system. Since there may be limited access to the Mains, our product makes use of solar panels to generate the power needed to run. After testing various configurations, it was found that a configuration of two 6V 7.5W solar panels in parallel reached the best compromise of minimizing cost whilst being sufficient to supply both microcontroller and sensors. Given this testing was done during the winter season at 42° latitude, different configurations may prove sufficient for different locations/seasons, so the panels are connected via screw terminals for easy substitution.

The power from the solar panels is then fed to the main system as well as charging a 4.2V LiPo battery. This is done via a TP4056 module to act as the battery management system (BMS) needed to safely charge LiPo batteries. This module also allows for pass-through charging in order for the system to remain operational whilst charging. Battery levels will be gauged by measuring the voltage of the battery via the ESP32. Given the Successive Approximation ADC used on the ESP32 is only 3.3V tolerant, our product makes use of a ½ voltage divider to keep the voltages within safe range. Since such a resistor-based voltage divider would be constantly drawing current even when measurements were not being made, a NPN BJT (PN2222A or equiv.) will be used to allow current to flow through the measurement circuit only when a 3.3V logic signal is sent from the ESP32 to its base. The output voltage of the TP4056 is fed into a boost converter module and stepped up to 5VDC, which is sufficient to power the ESP32 via the V_{in} pin. Given this pin is fed into a 3.3V LDO voltage regulator built into the ESP32 development board, better efficiency may be achievable by using a buck converter that steps the output voltage of the TP4056 down to 3.3V, but given the noisy nature of buck/boost converters, this would need to be filtered further to ensure the supply is regulated sufficiently. This avenue may be pursued if the LDO proves too inefficient.

Given both the modular design, along with the goal of minimizing cost, most sensors communicate with the microcontroller via the I^2C protocol. Since this serial protocol makes use of only two lines for data transmission for up to 128 devices (given unique addresses), the complexity of the sensor-to-microcontroller wiring can be minimized. Furthermore, the detection of which sensors are implemented for a given installation can be easily detected via the addressing of each sensor. Therefore, the number of sensors implementing this protocol was maximized. This also minimizes the number of GPIO pins connected to ADC0 in use (ADC capable pins of the ESP32 are limited to 5 since the second SAR ADC, ADC1, is not available when WiFi is in use).

The physical enclosure should allow for a minimum ingress protection rating of IP66 such that the mainboard and power management board minimizes the water ingress. This will allow for the system to be safely exposed to the outdoors along with being able to handle water exposure from irrigation sources. This will mean that the product should use weatherproof connectors when given sensors may be mounted separately from the primary enclosure.

When appropriate, major components will be socketed (such as the ESP32 Dev Board) or make use of solderless connectors (i.e. JST-XHP connectors). Use of connectors also allows for given sensors to be mounted further away from the main PCB when appropriate (i.e. a soil moisture sensor).

Our final product will have six sensors including a soil moisture sensor, a temperature and humidity sensor, a barometric pressure sensor, NPK sensor, and a rainfall sensor. These sensors will be socketed into a PCB, along with the microcontroller. For the microcontroller, we are using an ESP32. The ESP32 will push the data collected from the sensors to our database built using the Backend-as-a-Service Google Firebase. The set-up for Google Firebase includes two databases, the Realtime Database and the Firestore Database. The data will move from the hardware to the Realtime Database using the ESP32. Then, the data will move from the Realtime Database to the Firestore Database for long-term storage. The ESP32 will also push the data to ThingSpeak, our data analytics platform. ThingSpeak will have one channel that stores all the data from our device. The channel will have seven fields for each of the seven pieces of information we collect. Once the data is on ThingSpeak, a predictive model will be built to decide whether the crops should be watered. Once a conclusion has been made that the crops should be watered, that conclusion will go to the Firestore Database as a .json object.

Opening the app, the user will be directed to a login page. If they have not created an account, they will first need to do so. Account information, including name, email, and ID, will be sent to the Firestore database. Having this information stored allows the app to run login authentication and keep data about their crops private, allowing access from any device to the smart watering system. After logging in, the user will choose what type of crop they are growing from a list provided. Their choice will get sent to our Firestore Database. ThingSpeak will be working on receiving the data from the hardware device and comparing it to the limits established. If the data collected is not within the limits and predicted to need watering, the notification to water the plants will be sent to the user. This decision will be sent to the database. Aside from this feature, the app will also have another tab in which the user can input the conditions of the land. These conditions will be humidity, rainfall, temperature and soil nutrients. With that information, the MATLAB predictive model will be executed and it will tell the farmer the optimal crops to grow.

The collected data from the on-site sensors will be uploaded to the Realtime Database, then sent to the Firestore Database. Then, the app can access and display this data. Unless the size of the prediction model is overwhelming, the analysis in Matlab will be run through React Native. The main app interface will display real time measurements that refresh as new ones are being added to the database. This will include current weather, through using a weather API that will also be used in data analysis, and models of the collected data including past data, such as showing the moisture of the soil through the past week.

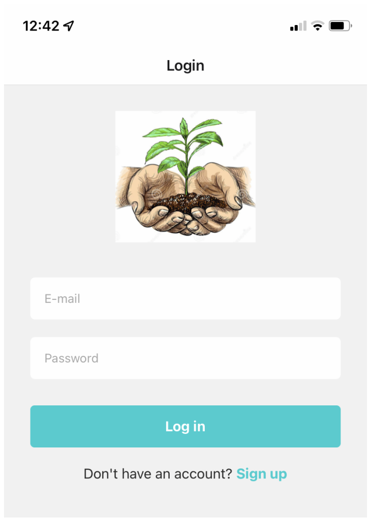
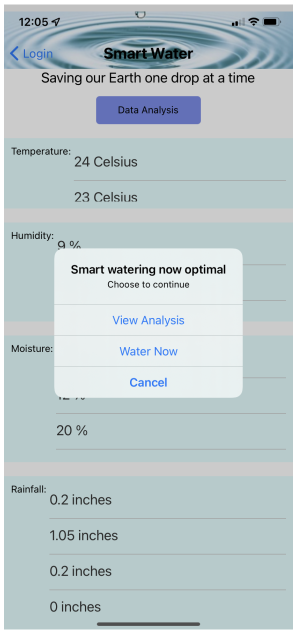


Fig. 1. App login page



Saving our Earth one drop at a time

Temperature: 24 Celsius
23 Celsius

Humidity: 9 %

Smart watering now optimal
Choose to continue

Moisture:
20 %

Rainfall: 0.2 inches
1.05 inches
0.2 inches
0 inches

Fig. 2. App notification and interface

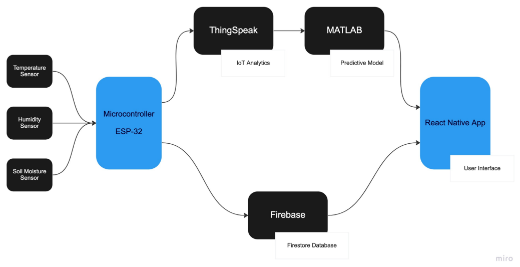


Fig. 3. Block diagram of the Smart Watering System

In summary, as visualized above, data collected from the sensors will be relayed to Firebase and ThingSpeak through the microcontroller. The data will go from ThingSpeak to MATLAB where the predictive model will make a decision. Then, both the decision and data from Firebase will be retrieved by the React Native app. (Ines Villafruela) (Jordan Remar) (Kaede Kawata) (Karin Luna) (Nyah Madison)

4.0 First Semester Progress

Our goal for this semester was to build the data pipeline that moves data from the hardware to the app by our First Deliverable Test. We were able to achieve much of our goal. Soldered onto our PCB are sockets where the ESP32 and sensors can be placed, allowing easy replacement of old and damaged components. Right now, we just have a TMP36 to measure temperature. The ESP32 transfers data from the TMP36 to the Realtime Database on Google Firebase. Besides this database, our Firebase project also has a Firestore Database that stores past data concerning temperature readings and login information.

For our First Deliverable Test, we wanted to test whether the ESP32 could relay data collected from the temperature sensor to our cloud database. To test this, we connected the TMP36 and ESP32 to a breadboard and wired them together. We used a computer to power the microcontroller. We wanted to run three trials. In the first trial, the temperature sensor will measure the temperature of the room and we will see that temperature displayed on the Firebase console. Then, we heated the temperature sensor with our fingers to see if the temperature on the console increased. For the last trial, we held a wet napkin on the temperature sensor to see if the temperature displayed on the console decreases.

During testing, the TMP36, first, measured the temperature of the room, which was about 20 degrees Celsius. This was displayed on the Realtime Database section on Google Firebase. Afterward, we warmed up the TMP36 with our fingers. About 5 seconds later, Firebase displayed a temperature reading that was 1 degree higher at 21 degrees Celsius. 5 seconds after placing a wet napkin on the TMP36, the temperature reading went down to 20 degrees Celsius and in the next interval, down to 19 degrees Celsius. This test proves that temperature data can be read from the TMP36 and that the

ESP32 can communicate to our database on Google Firebase.

On the software side, the connection between Firebase and the app is efficient, working both in sending and fetching data. Through authentication, accounts maintain private data by having collected data as sub-collections of each user. Through using the alert API, the app notifies the user by running a constant check if the data is out of range, thus needing watering, and calling the alert if so. The data is displayed in scrollable flatlists on the main home interface. (Nyah Madison) (Karin Luna)

5.0 Technical Plan

As observed in the Appendix 2-Gantt Chart, it is planned to finish the project by the end of March. The tasks are described below.

Task 1. Soldering hardware
To begin with, since we are building a modular system, we must finish soldering the sensors to the plates. Thanks to the modularity of the project, the number of sensors can be expanded according to the needs of the system. Once the sensors are connected, it must be verified that the current solar panels generate enough current to power the circuit and the circuit can sufficiently handle momentary power transients.
Lead: Jordan Remar
Assisting: Kaede Kawata

Task 2. Increase Firebase security
License problems with Firebase shall be resolved since it is Back-end-Service and affects the entire software. Increasing security in the database is vital to ensure the integrity of the users information. This security should be tested with different users and roles. Even with increased security, app users as well as the ESP32 should still be able to communicate with both the Realtime and Firestore databases.
Lead: Nyah Madison
Assisting: Karin Luna

Task 3. Moving data to Firestore
Right now, every new data collected is kept in a Real Time Database in Firebase. This shall be changed to Firestore which is a fixed Database that works with noSQL and that builds on the successes of the Real Time Database with a more intuitive data model. Firestore also features richer and faster queries.
Lead: Karin Luna
Assisting: Inés Saavedra, Kaede Kawata

Task 4. Connect ThingSpeak to React App
This task is essential to meet the client's objectives. ThingSpeak should be used as the IoT analytics platform. It must be connected to the application to be able to show the farmer when to irrigate his land. Once the data is analyzed, an implementation to act on the data should be executed in order to send a notification to the app.
Lead: Inés Saavedra
Assisting: Karin Luna

Task 5. Create different users
In order to upgrade the software and make the system as efficient as possible, different users with different roles should be described. This way, the admin user will be able to read, write and execute, and the workers who only irrigate the land will just be able to read the notifications.
Lead: Karin Luna
Assisting: Inés Saavedra

Task 6. Weather App

To create a complete application that has all the possible tools, a weather measurement system can be developed so that the farmer is aware of possible sudden changes in temperatures and thus also helps to optimize the irrigation system. This shall be made using a weather API.
Lead: Karin Luna
Assisting: Inés Saavedra, Nyah Madison, Kaede Kawata

Task 7. Data Modeling
In the app, the collected data will be displayed to the user. Currently, it is in flatlists. This will be improved upon by having visual models such as graphs or plots displaying the progression over time of the measurements. These models will be easy to understand, visually pleasing, and able to utilize past data.
Lead: Karin Luna
Assisting: Inés Saavedra

Task 8. Adding LoRa or SIM card module
Right now, the network for ESP32 is based on Wi-Fi connection. Because our target user is less likely to have Wi-Fi available in the large field, we decided to use either LoRa module or have a SIM card module that the user can put their own SIM card in to connect the hardware to the internet.
Lead: Kaede Kawata
Assisting: Jordan Remar

Task 9. Registering ESP32 to each user
To connect the measured data from ESP32 to Mobile App, it needs to differentiate one ESP32 to another. Using a Bluetooth module, the user should be able to register their device and set up the network so that the right data is shown to the right user.
Lead: Kaede Kawata
Assisting: Nyah Madison

Task 10. Connection between ESP32 and ThingSpeak
To send data to be analyzed by the MATLAB prediction model, ESP32 needs to send data to ThingSpeak channel with HTTP protocols and API key.
Lead: Kaede Kawata
Assisting: Inés Saavedra

Task 11. Test Whole System
Lastly, once the system is complete, we want to deploy it using AWS so that MATLAB is not running in a personal's laptop but in a server. To ensure that everything works, a testing script should be created to test different vulnerabilities so that we can offer a secure product.
Lead: All (Ines Villafruela) (Kaede Kawata)

6.0 Budget Estimate

Item	Description	Cost
1	Soil Moisture Sensor	$7
2	Temperature and Humidity Sensor	$7
3	Barometric Pressure Sensor	$8
4	ESP-32	$19
5	NPK Sensor	$150
6	LoRa Antenna	$12
7	GPS	$25
	Total	$228

The NPK sensor will be our most expensive item on our budget. This sensor is needed to measure levels of Nitrogen, Phosphorus, and Potassium in the soil. This data is relevant for the designed prediction model. The $150 cost is the highest price found, though a lower cost is ideal once a registered vendor is found. (Nyah Madison) (Kaede Kawata)

7.0 Attachments

7.1 Appendix 1 – Engineering Requirements

Team # 29

Project Name: Smart Watering System with the Internet of Things_______

Requirement	**Value, range, tolerance, units**
Enclosure Dimensions	6in x 5.5in x 5.5in
Input Supply (Minimum)	4.5V to 6.0V @ 100mA (minimum)
Communication Range	> 0.5 miles
Communication Protocol	IEEE 802.1 b/g/n, LoRaWAN
Sensor Accuracies (Minimum)	2-5% within actual value
Ingress Protection	IP66 Rated

(Jordan Remar)