EE 475 Project Report

Terrabox

Tristan Barr
Clayden Boyd
Bowen Zuo
Kahn Wang
Simon Liu
Oscar Zeng

Dept. of Electrical and Computer Engineering, Box 352500
University of Washington
Seattle, Washington 98195-2500 U.S.A.

March 14, 2022

Table of Contents

Team, Roles, and Responsibilities

Throughout the development process of TerraBox, we broke down the development to focus around three key aspects of our product: development of a mobile application and integration of that app with a cloud database, the internet and inter-device communication of commands and telemetry data, and the construction and control of the necessary hardware components for the box itself. The six of us ended up splitting into three teams of two that each focused on each one of these three core aspects of the TerraBox development process. Clayden and Oscar worked on creating a mobile app and tying the app to work with a cloud database, Simon and Tristan worked to get all the hardware components working with the STM32 and much of the box built, and Bowen and Kahn worked to bridge the gap between the mobile app/cloud database and the embedded electronics by working out how the communication between all of these devices would work to have a fully cohesive and functioning system where commands on the app would be able to reach the box, and telemetry data from the box would be able to get sent back to the app. Though team members ended up focusing on one of these three development aspects, everyone reached beyond their primary section of work to help group members in other sections solve problems when needed. A more detailed description of individual roles is listed below.

Bowen: Set up the Raspberry Pi. Developed Python software on Raspberry Pi for the STM end and Firebase end. Built the real-time functions from Firebase to STM and sensors. Helped with TerraBox construction with the hardware team.

Clayden: Created Firebase project to act as the main database for iOS to microcontroller communication. Built back-end functionality for iOS app so that UI features would communicate with the database. Drafted UI designs for feature testing. Established user-authentication, and community posts pages for iOS app, and built in the corresponding database structure in Firebase.

Kahn: Develop python software on Raspberry Pi, build data transmission between STM32 to Raspberry Pi, and Raspberry Pi to Firebase cloud storage. Design and document the data format. Communicate between different teams to align the design.

Oscar: Designed IOS app prototype UI in Figma and implemented main control / status page that reads and writes values to firebase to control Terrabox functionality. Also implemented live status indicators for reading and writing settings and measurements. Included features such as SVG rendering on climate selection and climate preset drop down menu.

Simon: Validated functionality of hardware components, designed how to best connect and control all the hardware components together, built the necessary components to get TerraBox working as if it were an actual product and built up the box itself.

Tristan: Developed all embedded software in C/C++ for the STM32F. Verified STM code functionality through testing and developed necessary connections between STM and components. Developed STM-to-Raspberry Pi connection via UART with Kahn and Bowen. Constructed Terrabox with Simon.

We never made any set team roles like a dedicated team leader or meeting manager or anything, but that was because everyone understood their responsibilities for the team and everyone worked to fulfill those responsibilities so nothing like that was necessary. Team members were diligent in asking questions to other group members, arranging meetings as needed, and keeping other group members in check so we never felt concerned that we wouldn't be able to complete our project or that we would have trouble meeting any deadlines.

Product Requirements Document (PRD)

Have you ever tried making an indoor plant terrarium only to have the plants die after a few weeks? Or have you tried growing tropical plants in your typical indoor climate? Growing plants is a hobby that requires constant attention, patience, resources, and maintenance of a plethora of additional environmental variables that many simply don't have the time for. Our Terrabox project is meant to directly address the climate, time, and resources, needed to grow a variety of plants in various climates. Many plants require native climate conditions such as temperature, humidity, soil moisture, etc. to grow properly and remain healthy. By utilizing a mobile front-end platform to communicate with a collection of sensors and climate-controlling components, we plan to implement a real-time terrarium automation system that will be able to replicate a plant's native growth environment to optimize plant health and growth rates.

To use Terrabox, users just need to place their plants inside the box and download the TerraBox app on their phone. After creating an account on the app, the user can log in and check the status of the environment in the box anytime they want. The user can also make changes to the box's internal environment through the app as needed. Heating, cooling, humidity, lighting, and watering are all only a simple tap away.

Our product can be adopted in a wide range of applications such as home gardening, scientific research, and education. Firstly, all plant hobbyists can benefit from this product by outright reducing the amount of work they will need to spend maintaining their plants. The TerraBox adds life and brings the environment back to nature for the average busy worker, and the automatic control system ensures that people won't have to be bothered to manually maintain their plants regularly. Even those who consider themselves a plant-challenge, as long as they follow the instructions to set all parameters correctly, they won't have to worry about the plants inside withering away. For researchers who raise many different species in laboratories, our product will save a lot of time since the automatic climate control means they will not have to care for each Terrarium individually. Scientists can easily set the desired parameter under different research conditions using our app. Terrabox is also a perfect container for reptiles, as it provides constant environmental conditions for pets that require special weather conditions.

Realistic Constraints and Engineering Standards

In any given engineering project, there must be realistic constraints and boundaries set to what is expected of the project - what will the end product / project be able to accomplish in terms of functionality, and what will be beyond its capabilities. Such constraints and expectations should be set at the start of the project and further refined as the project progresses. Setting these expectations define what the product can do and clarify any assumptions created about the project. Furthermore, in today's society, factors such as energy consumption and safety are held to high standards and quite important to such projects. Therefore, we must consider such standards, especially if the project is targeted towards the consumer markets.

For any consumer product, a constraint that is always considered is budget and pricing. Creating products for a mass market of consumers requires that we keep the product reasonably priced and budgeted for the quality of the overall product. As we developed and planned for Terrabox, budget was a constraint that we implicitly imposed on ourselves - to keep the overall project and its components reasonably priced such that we used components that would get the job done, but likely not top-of-the-line components that may have cost significantly more. Though this constraint was negotiable, we decided as a team to target a lower cost that we were willing to contribute to.

In terms of hardware functionality and capabilities, the Terrabox is multifunctional to a certain extent. Terrabox can maintain and adjust humidity levels within the given enclosed environment under a reasonable range. It cannot drop humidity to 0% but rather can passively lower humidity levels to surrounding humidity levels. Alternatively, increasing humidity is possible again under a reasonable limit, not to the extent of 100% humidity since the system is not completely sealed, however ~90% is a reasonable level that the Terrabox may be able to achieve. To adjust the humidity, the Terrabox activates a humidifier that pipes water vapor into the system, and to lower humidity the Terrabox achieves that goal passively. In terms of heating and cooling, this functionality is achieved via a heating pad and a fan-powered air circulation system. Terrabox can actively heat the enclosed environment to a tested upper range of 90°F (~32°C) and passively cool back down to room temperature. There is no active cooling system besides airflow and thus the lower range of temperature that Terrabox can regulate depends on the surrounding environment. For lighting, Terrabox can adjust lighting and timing as necessary and as such lights can be left on indefinitely or turned off as needed.

Terrabox has an accompanying iOS application that enables customization and control of the Terrabox. This application contains two core functionalities; The first core functionality is reading from the Terrabox sensors to update current readings on the app such as displaying water reservoir levels in 25% increments (this is a constraint that is controlled by the number of water sensors we have in our reservoirs, also a budgetary decision that was made), current humidity and temperature, and current status of the lights. This functionality of reading from the Terrabox is controlled with a “Refresh Status” button on the main page of the application. Tapping this button checks the Firebase values that are updated via the Raspberry PI, and reading values updates the status bar on the main app page. The second core functionality of the application is writing values to the Firebase Database with toggle buttons on the main page. Users are able to toggle up or down the temperature and humidity settings by +1 and -1, this is somewhat a constraint in the design decision since more buttons may have made the UI significantly more clustered. The sprinkler and light are timer based -

this means that the toggles associated with those functions are tied to either Off or a set timer setting, and thus cannot be adjusted otherwise. This was done so that there was a consistent variety of data within the Database and that the communication between Firebase - Raspberry Pi - STM was as efficient as possible.

System Requirements Document (SRD)

Functional Requirements

1.In order to operate the system, both raspberry pi, STM 32 and the plug form terrabox should be connected to power. 2 liters of water should be filled into the reservoir.
2.The system should be able to control the temperature and humidity in the set range, cycling the water in the box, and turning on the light in the set time.

Firmware Requirements

1.The water cycling system is able to drain the plant from above and collect the water from bottom.
2.The heating system should be able to heat up the box to 37℃, and cool to room temperature.
3.The led light can be turned on or off.

Software Requirements

1.The app will be able to add new users and store the login information for them.
2.The app can set time for light and sprinkler, temperature and humidity for the system.
3.The app can send data to firebase
4.The raspberry pi can receive and send data to STM32
5.The raspberry pi can receive and send data to Raspberry pi
6.The raspberry pi can control the flag for light and sprinkler based on the set time.
7.The STM is able to send and receive data from raspberry pi
8.The STM is able to maintain the target temperature using heating pad and fans.
9.The STM is able to maintain the target humidity using humidifier
10.The STM is able to turn on sprinklers and lights based on the flags.

Thermal Requirements

The box will be able to maintain temperatures of 19-37℃ in a room of 20℃.

User Interface Requirements

The app is designed to operate on the IOS system, the user must have an IOS system client to use the app and control the system.

Safety Requirements

The product consists of electricity and water, make sure not to flip or destroy any part or there will be a chance to cause electric shock.

Project Schedule

The entire project timeline spans from January 3 to March 14 of which it is divided into three phases. The first phase will last from January 7 - February 9. This phase will consist of establishing all of the team operating expectations, settling on a final project, and planning. Additionally, in Phase I, we built up the majority of the software for the project. Phase II allowed us to resolve any errors we had from Phase I and build up a full functional prototype. In phase III, we fixed any last bugs and finalized our product. A detailed representation of these phases and their corresponding tasks can be found in the GANTT Charts included below.

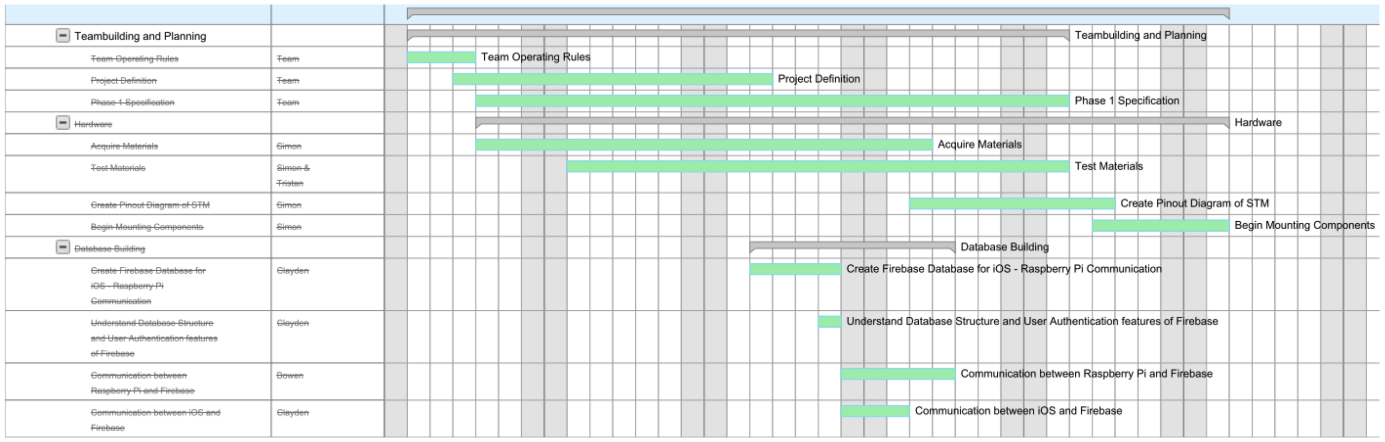


Figure 1: Phase 1 GANTT Chart (1/2)

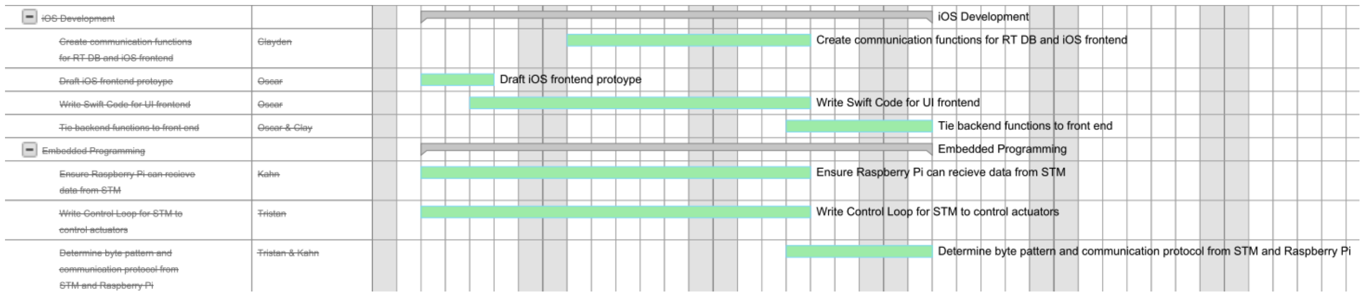
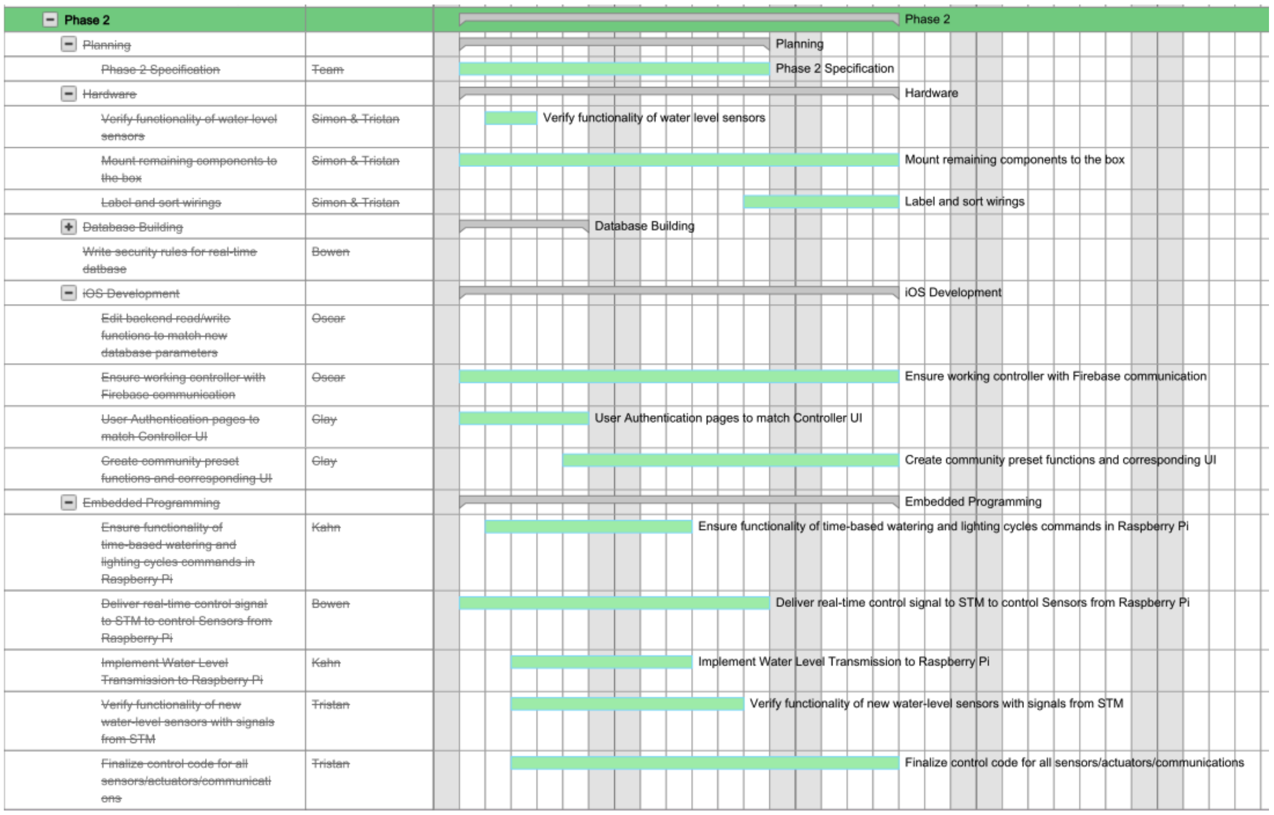
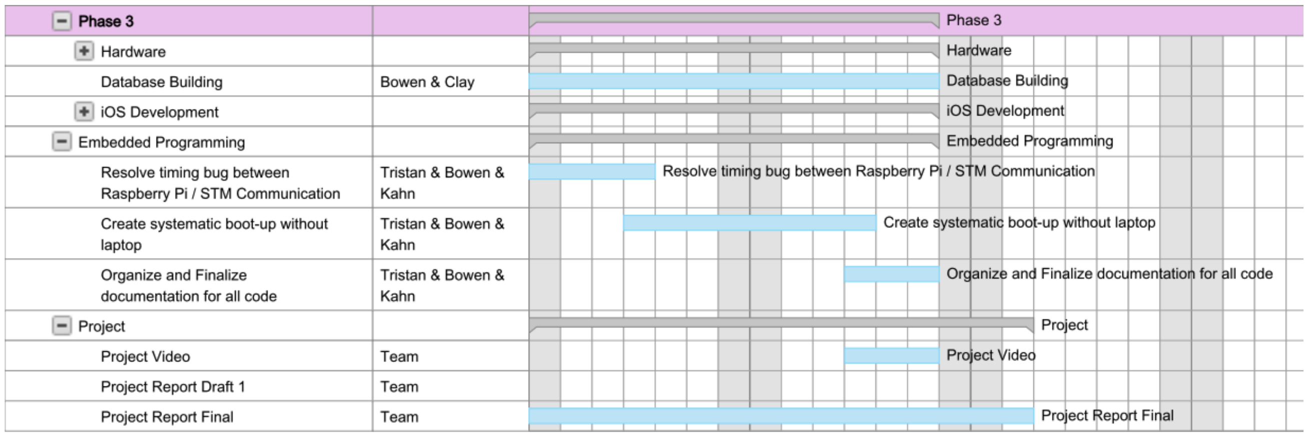


Figure 2: Phase 1 GANTT Chart (2/2)



Phase 2	
~~Planning~~	
~~Phase 2 Specification~~	~~Team~~
~~Hardware~~	
~~Verify functionality of water level sensors~~	~~Simon & Tristan~~
~~Mount remaining components to the box~~	~~Simon & Tristan~~
~~Label and sort wirings~~	~~Simon & Tristan~~
~~Database Building~~	
~~Write security rules for real-time datbase~~	~~Bowen~~
~~iOS Development~~	
~~Edit backend read/write functions to match new database parameters~~	~~Oscar~~
~~Ensure working controller with Firebase communication~~	~~Oscar~~
~~User Authentication pages to match Controller UI~~	~~Clay~~
~~Create community preset functions and corresponding UI~~	~~Clay~~
~~Embedded Programming~~	
~~Ensure functionality of time-based watering and lighting cycles commands in Raspberry Pi~~	~~Kahn~~
~~Deliver real-time control signal to STM to control Sensors from Raspberry Pi~~	~~Bowen~~
~~Implement Water Level Transmission to Raspberry Pi~~	~~Kahn~~
~~Verify functionality of new water-level sensors with signals from STM~~	~~Tristan~~
~~Finalize control code for all sensors/actuators/communications~~	~~Tristan~~

Figure 3: Phase 2 GANTT Chart



Phase 3	
Hardware	
Database Building	Bowen & Clay
iOS Development	
Embedded Programming	
Resolve timing bug between Raspberry Pi / STM Communication	Tristan & Bowen & Kahn
Create systematic boot-up without laptop	Tristan & Bowen & Kahn
Organize and Finalize documentation for all code	Tristan & Bowen & Kahn
Project	
Project Video	Team
Project Report Draft 1	Team
Project Report Final	Team

Figure 4 : Phase 3 GANTT CHART

Project Resources

Various external resources were needed in designing and constructing Terrabox. The resources utilized are detailed within this section and include working space, hardware components, external software, tools, and funding.

The ECE Lab 045 in the Allen Center of Electrical and Computer Engineering provided us with a safe workspace in which to construct our prototype box. Lab time was not limited, as we were able to access the lab at any hour of any day, which enabled us to complete our prototype on time. The lab was outfitted with the necessary space and tools we needed to complete our project. Tools and hardware provided by the lab space are further detailed below.

Some hardware components were provided to us free-of-charge by the lab and course labkit. These included jumper and non-jumper wires, STM32F and the required debug cables, Raspberry Pi and the required debug cables, small fans, and desktop computers. Hardware components which were not directly provided include DHT11 temperature and humidity sensors, electric heating pad, water atomizer, extra small fans, water pump, silicon tubing, 12V DC power source, 12V relays, water tank, plastic box, water level switch sensors, and LED strips. The function of each of these components within our prototype are detailed below:

- Wires: Connect all electrical components within our system.
- STM32F: Directly read sensor values, and control all actuators for controlling box climate including pump, heater, lighting, humidifier, and fans. Transmits all readings to Raspberry pi for displaying real-time data to the mobile application.
- Raspberry Pi: Bridge the gap between the STM32F and the Firebase. Takes sensor readings and transmits them to our real-time database. Determines when to activate the water pump and lights based on a timing cycle.
- Fans: There are three fans in total, one for cooling and venting, one for dispersing heat produced from the heating pad, and one for dispersing humidity.
- DHT11: Three of these sensors are used in total. Gather temperature and humidity readings to determine actuator commands.
- Heating pad: Used to heat up the box.
- Water atomizer: Vaporizes water to increase humidity levels within the box.

- Water pump and silicon tubing: Pump water through silicon tubing to simulate rainfall within the box.
- 12V DC power supply: Provides power to all actuators requiring 12V.
- Relays: Used to switch on and off actuators. Six relays used in total.
- Water tank: Serves as a water source for the pump and humidifier systems as well as a drain for water to leave the inside of the box.
- Plastic box: Used to house the entire system.
- Water level sensors: Switch sensors to determine water level within the in-system reservoir.
- LED strips: Provides light source for within the box.

External software describes software not developed by our team members and includes the firebase real-time database, open-source C libraries, open-source Python libraries, and IOS related libraries. Additionally, DHT11 sensor code was adapted from online resources.

Below is a list of tools used in completion of our project:

- Soldering iron and solder
- Ventilation fans for safe soldering
- Power Drill
- Manual Saw
- Wood shelves
- Hot glue gun
- Duct tape and electrical tape

The project was largely funded by group members. External donations were collected on an individual basis.

Outline of Experiments

The STM program was developed to meet the functional requirements we set. The goals of the STM program are as follows:

- Be able to read real-time temperature and humidity within the box
- Be able to determine whether to heat, cool, or maintain the temperature within the box

- Be able to determine whether to increase, decrease, or maintain the humidity within the box
- Be able to receive commands and target temperature and humidity from the Raspberry Pi
- Be able to send the real-time temperature and humidity readings to the Raspberry Pi
- Be able to read water levels within the water tank
- Be able to send water level readings to the Raspberry Pi
- Be able to send actuator status to the Raspberry pi

From these requirements, a first draft STM program was developed. The flowchart below depicts this initial program:

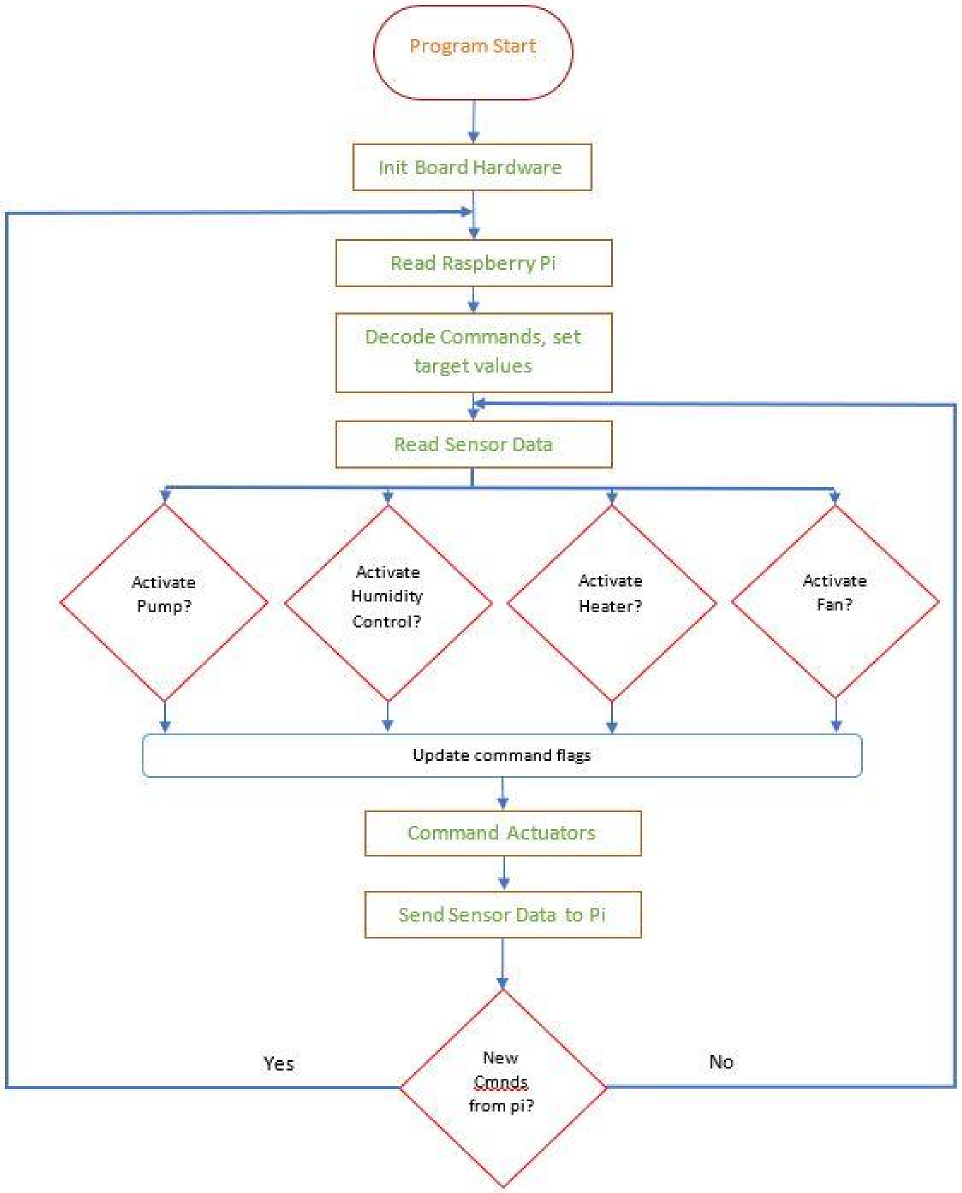


Figure 5: Flowchart of initial STM program.

Through experimentation, it was determined that the program was functional, but not completely optimized. UART transmission between the Raspberry Pi and STM could halt the program's operation in the initial design. In order to overcome this, UART communications were configured as interrupts. This allowed for UART sends and receives to happen in the background of the program execution. This sped up the control loop execution time and increased the refresh rate of front-end data. Below is the final STM program flowchart:

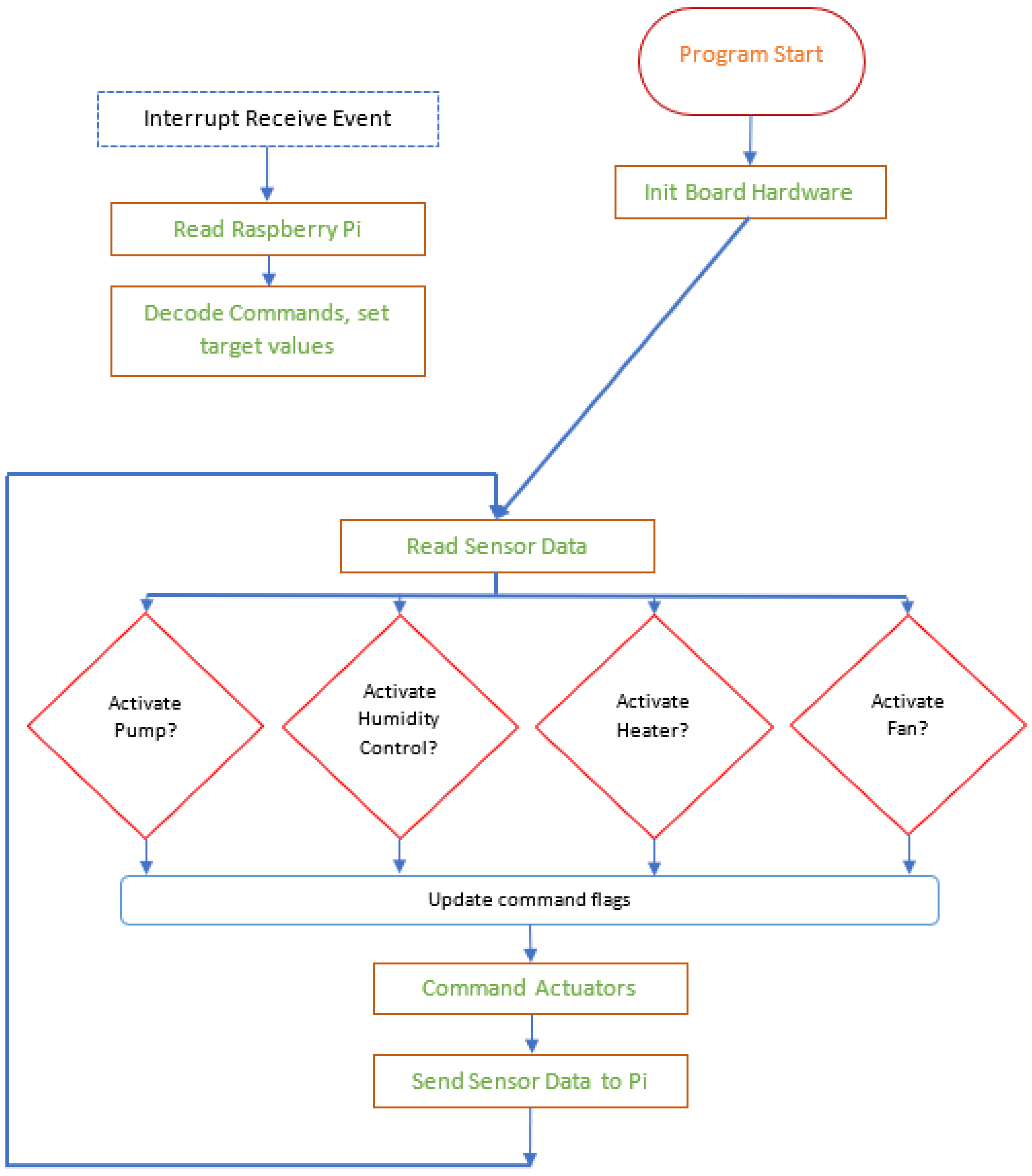


Figure 6: Finalized STM program flowchart.

A discussion on how the STM accomplishes its goals on a lower-level is provided below:

- Since a single line is used to get temperature and humidity readings, the STM utilizes a series of timed writes and reads of the pin connected to the data line of each DHT sensor. To initialize the sensor, the STM pulls the data line high for around 20 milliseconds. The pin is switched to input mode and the DHT11 sensor will pull the pin low for 80 microseconds and then high for 80 microseconds. This signals that the sensor is ready to transmit, and is determined by reading the data line two times 80 microseconds apart. Transmits are defined as ‘0’ if the pin is pulled high for only 30 microseconds and ‘1’ if the pin is high for 70 microseconds. Therefore, the STM reads the pin at set intervals to determine if the data line transmits a ‘1’ or ‘0’. Final temperature and humidity readings are taken from the second and fourth byte of the transmitted data, with the fifth byte serving as a checksum for error checking.

- Heat control is accomplished via three actuators: cooling fan, heating fan, and heating pad. If the read temperature is lower than the target temperature, the heating pad and heating fan will activate by writing the pin to the corresponding relays. Similarly, the cooling fan is activated when temperature is lower than desired.
- Humidity control is accomplished by writing high to the pin corresponding to the water atomizer if humidity readings are too low, and writing high to the pin corresponding to the ventilation fan if humidity readings are too high.
- Target temperature and humidity, as well as commands for the lights and pump are received from the Raspberry Pi in the form of UART via interrupts.
- Readings and actuator status are sent to the Raspberry pi via UART as well.

To test the functionality of the STM program, we first decided to test individual tasks within the flowchart independently and then test everything together once the functionality of the individual components were verified. Below lists how we tested each individual component:

- Sensors: Read from each sensor and observe average values from all sensors together. Cross referenced temperature/humidity readings with temperature and humidity reader located in the same room. Read from water level sensors with water present and not present, ensured readings were consistent.
- Command flags: Used sensor readings to produce command flag values. Ensured command flags in response to sensor readings were consistent with the goals of the box.
- Actuators: Ensured relays could be controlled via GPIO output pins on the STM32. Verified command flags triggered the proper actuators.
- UART Sends: Sent dummy data to Raspberry Pi in the expected structure for actual runtime. Verified Raspberry Pi received the proper data.

- UART Receives: Sent dummy data from Raspberry Pi in the expected structure for actual runtime. Verified STM received proper data.

Combining all of these individual tasks proved seamless. To test the entire STM program we manipulated target values directly in the firebase and ensured the proper actuators were activated as a result. Ensured the temperature readings in the firebase were updated correctly.

Trial Designs

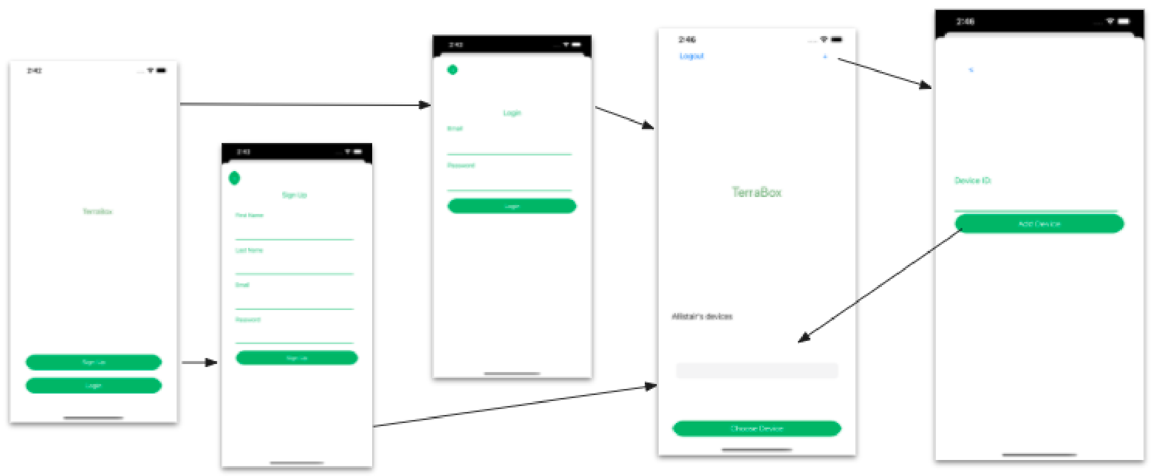


Figure 7: Initial UI design for testing database communication

As illustrated by Figure 7, in order to build the backend functionality of the app, a temporary frontend UI was created for testing. The figure includes each of the views associated with different pages for user login, sign up, and the main home page where the user can select the device to interact with. The backend functions for the main controller functionality were created in parallel with the UI shown in Figure 8.

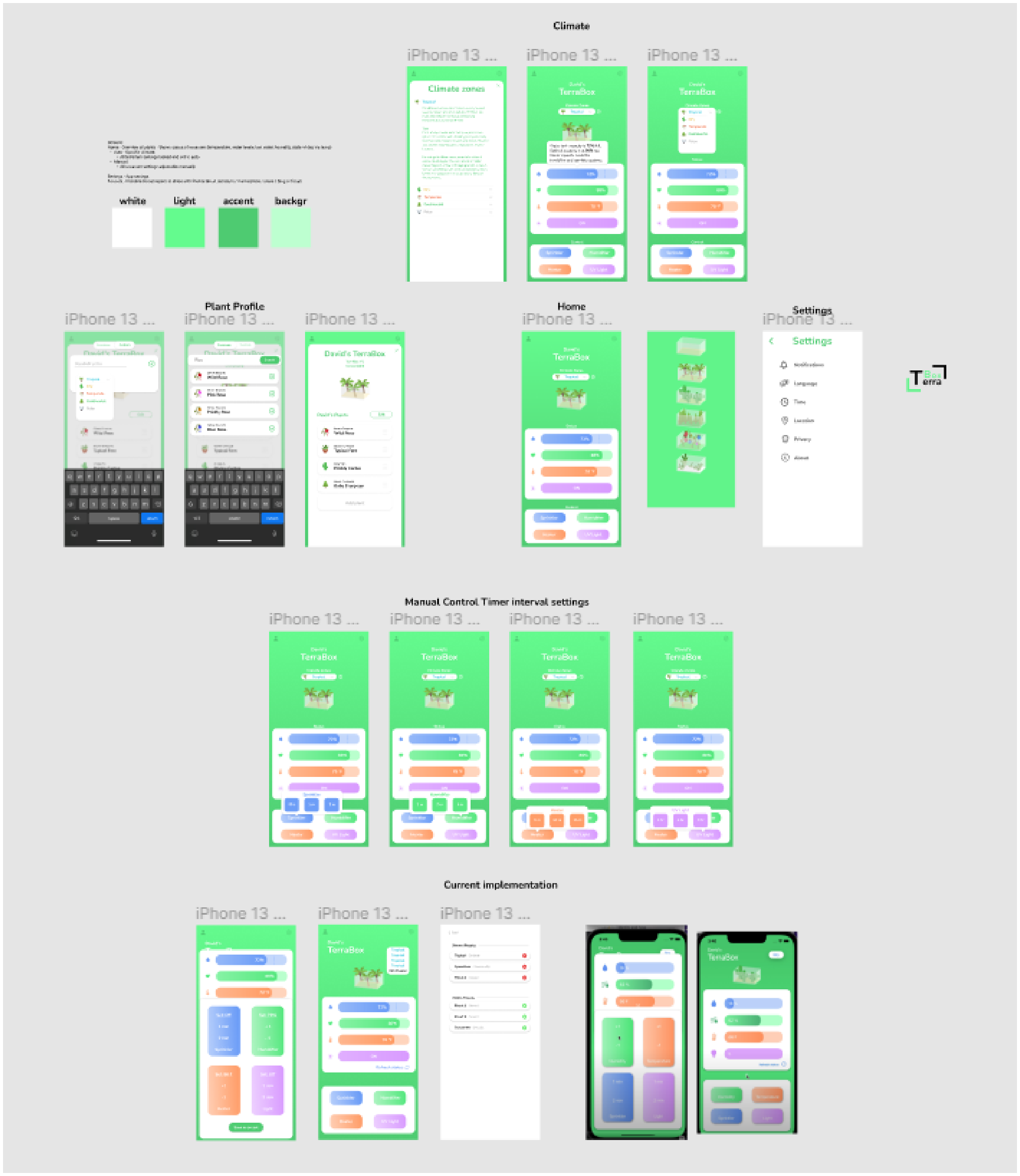


Figure 8: UI prototypes and ideas for Terrabox Application

Figure 8 shows the various UI prototypes that were designed during the process of creating this application. The top left corner addresses the app needs to facilitate the overall design and functionality needed. Each variation of the UI prototype depicts buttons and considerations that were made in the design process. The views with the keyboard represent an extra feature that was considered in design but wasn't really necessary and thus the idea was scrapped in the end. Ultimately, the bottom row most closely represents the current iteration though we made a few design changes without prototyping that aren't depicted above.

Our original design of the electrical connectivity was as follows:

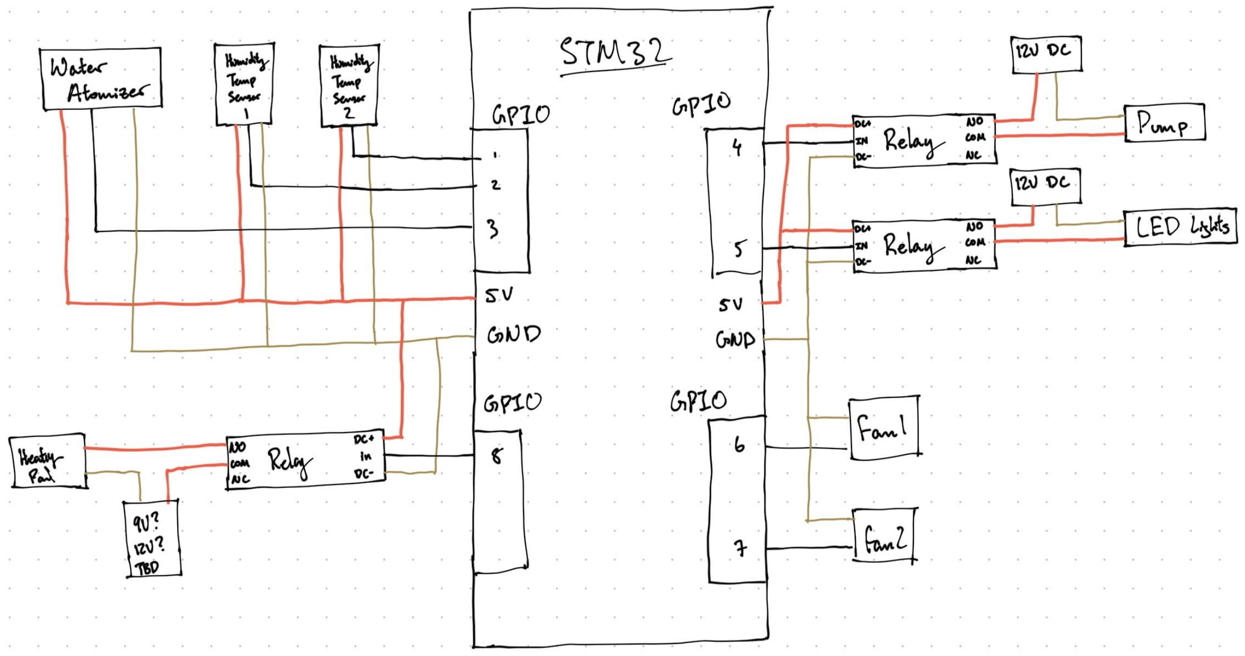


Figure 9: Original schematic plan.

Our design changed in response to the following discoveries:

- Interacting with the original design as a user would give no indication to water levels within the in-system water reservoir. Therefore, the user would have no way of determining when to refill the water tank, which is essentially the only non-automated aspect of the Terrabox. To overcome this, we incorporated water level sensors to alert the user when to refill the tank.
- Keeping our design as compact as possible forced us to find an alternative method of powering the STM. We decided to power the STM through the Raspberry Pi.
- For the same reason above, we integrated all components requiring 12V power to a single 12V line. This allowed for only 2 external power chords to be plugged in for the box to operate.

Our electrical schematic therefore changed to the following:

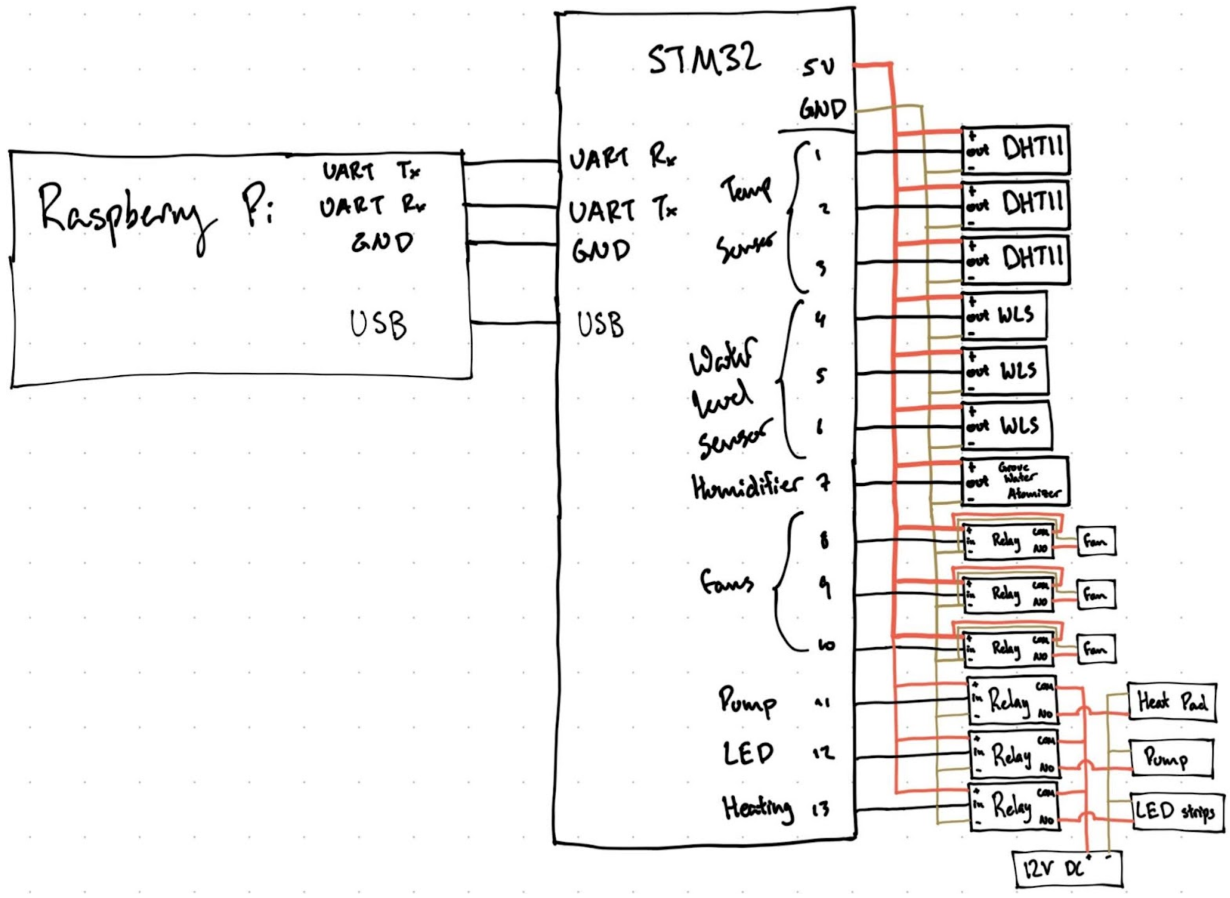


Figure 10: Final resulting schematic.

The overall flow of data and commands from the app down did not change throughout the course of this project. The figure below shows the sequence chart of top-to-bottom interaction for our system:

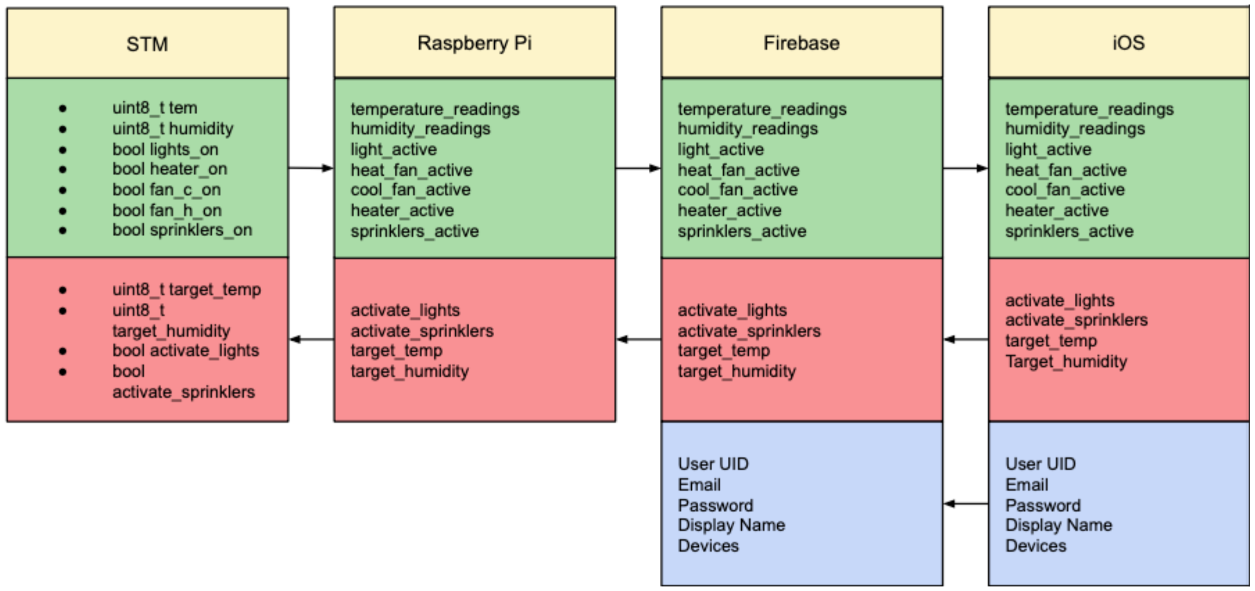


Figure 11: Data Flow UML chart.

The overall software design for the mobile application, raspberry pi, and STM32 did not change throughout the quarter, with the exception of adding water level sensors to the system. A UML chart of the finalized system is shown below:

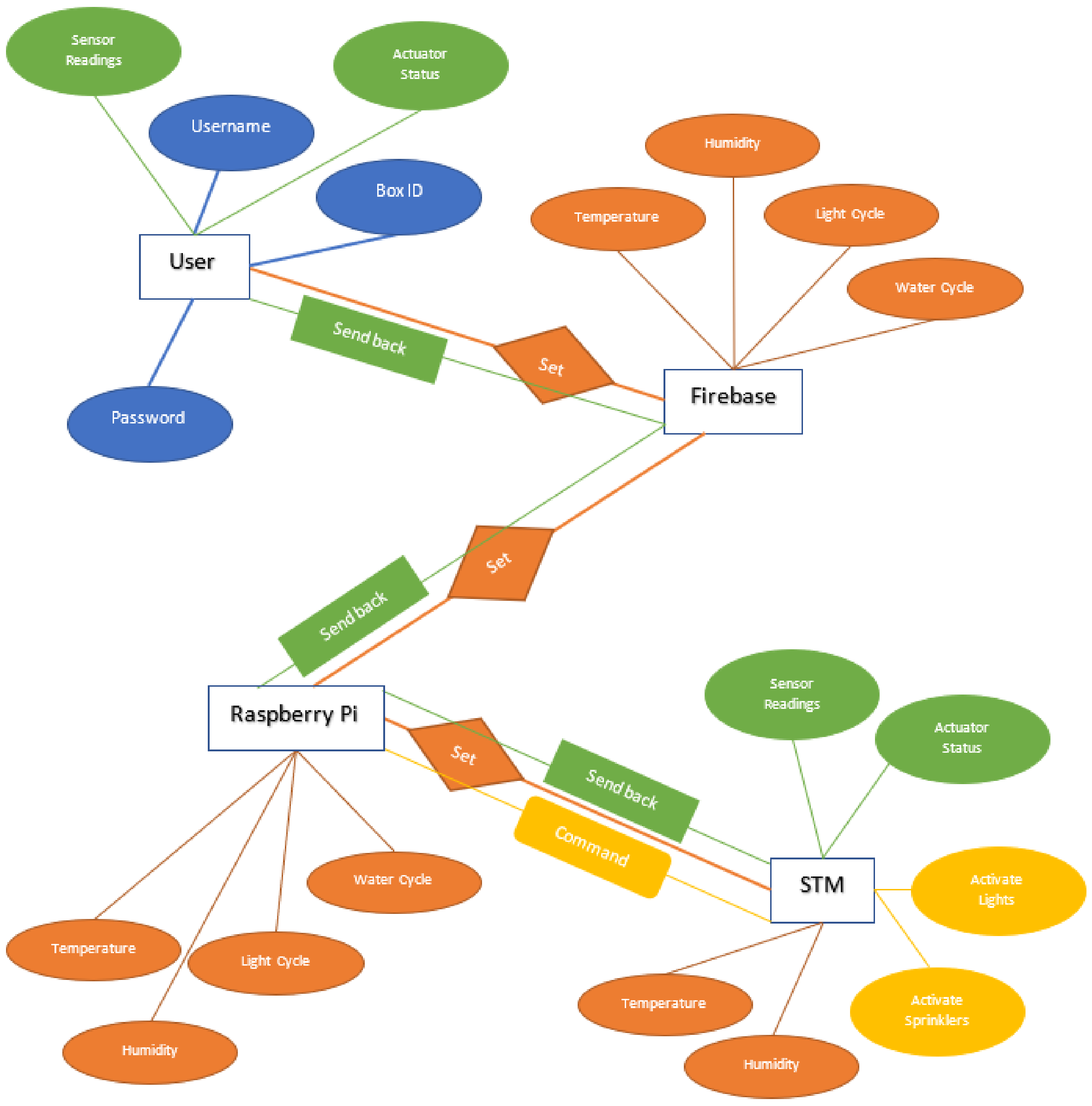


Figure 12: UML chart of finalized design.

A sequencing chart from end-to-end interaction is shown below:

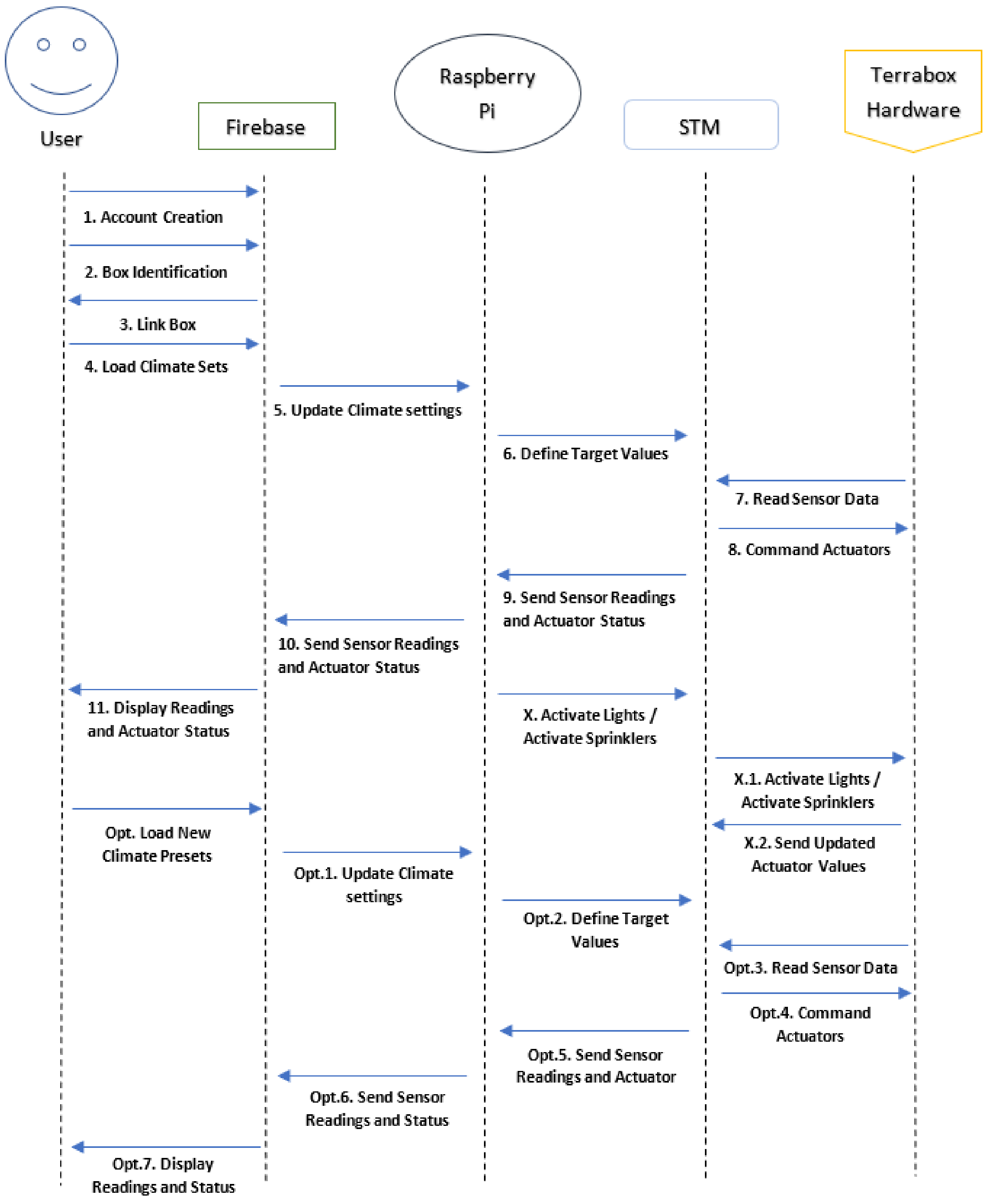

User
Firebase
Raspberry Pi
STM
Terrabox Hardware
1. Account Creation
2. Box Identification
3. Link Box
4. Load Climate Sets
5. Update Climate settings
6. Define Target Values
7. Read Sensor Data
8. Command Actuators
9. Send Sensor Readings and Actuator Status
10. Send Sensor Readings and Actuator Status
11. Display Readings and Actuator Status
X. Activate Lights / Activate Sprinklers
X.1. Activate Lights / Activate Sprinklers
X.2. Send Updated Actuator Values
Opt. Load New Climate Presets
Opt.1. Update Climate settings
Opt.2. Define Target Values
Opt.3. Read Sensor Data
Opt.4. Command Actuators
Opt.5. Send Sensor Readings and Actuator
Opt.6. Send Sensor Readings and Status
Opt.7. Display Readings and Status

Figure 13: Sequence diagram for Terrabox System with periodically occurring events (X) and optional user changes (Opt).

The hardware of the box went through a couple of iterations as well. The original box design was as follows:

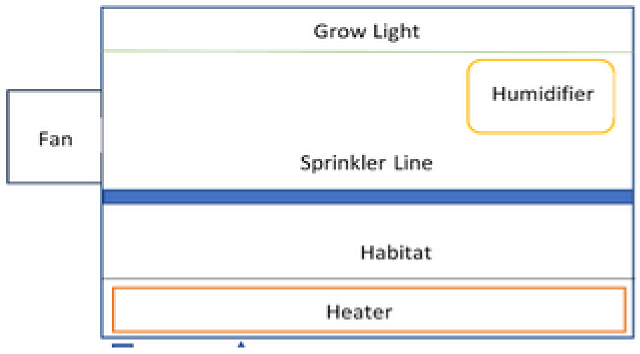


Figure 14: Original box layout concept

However, the following led us to make changes:

- Placing the heating pad beneath the soil would not generate the heat as we intended. We instead decided to attach the heating pad to a heating fan, which allowed us to disperse the heat via the air rather than through the soil, which proved more effective.
- The original design had insufficient space for all components. We decided to separate the box into two different compartments, one to contain the actual habitat and one to house the water tank and electrical components.
- We realized that the water reservoir was largely inaccessible to the user. We created a water intake which protrudes from the box in order to allow the user to refill the water tank in a non-invasive way.
- Humidifier operated by vaporizing water that was placed on the underside of the heating coin. The humidifier was placed within the water reservoir to achieve proper humidification.

A final layout of the box and its components is provided below:

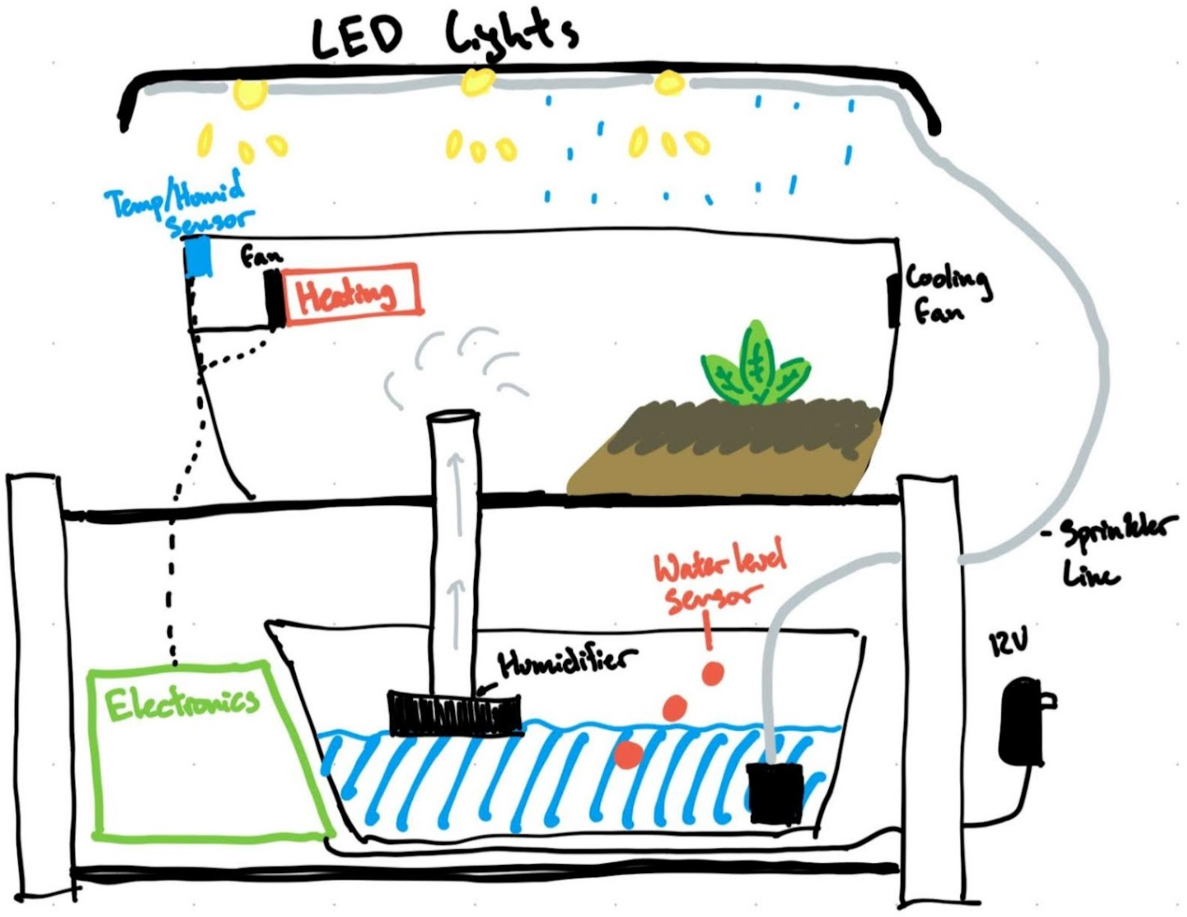


Figure 15: Finalized box layout

The STM program underwent two distinct iterations as a result of experimentation. The previous section of this document outlines the different designs and reasoning behind these designs.

The final bill of materials is shown below, this bill of materials accounts for all major hardware components in our prototype:

Component	Quantity	Total Cost ($)
Grove Water Atomizer	1	10.90
DHT11 Temperature/Humidity Sensor	3	6.99
Heating Pad	1	5.95

5V DC Fan	3	2.95
12V DC Power Source	1	8.99
Water Pump	1	10.99
LED Strips	1	14.97
Relay	6	18.57
Plastic Box	1	6.48
PVC Pipe	1	4.96
Tube Funnel	1	0.98
Total Cost	$92.73	

Experimental Outcomes

Our project includes the following features:

1. Humidity control via a humidity sensor in conjunction with a humidifier.
2. Temperature control utilizes temperature sensors to identify the box temperature, heating pads which heat up the box, and a fan which cools down the box.
3. LED lights which serve as the sunlight source for the box. The lights are used to simulate daylight and natural heat from the sun.
4. Simulated rainfall via a sprinkler system to cool down terrarium wildlife and water terrarium plants. Sprinkler system consists of a single mister connected to a micro irrigation system.
5. Real time App control on the environment in the box
6. Real time environment status updates that can be viewed on the App

These features provide the basic necessary functionality that our Terrabox will need to operate. These features allow our product to effectively control the environment for plants and animals within the box.
In addition to the climate-controlled box, our team will deliver a mobile application which allows for complete control of the box environment. This mobile application will

be compatible for iPhone and will serve as the control unit for climate control. Users may customize their box climate from within the app and the app will have preset climate conditions to choose from. The app will display data about climate conditions within the box, which will be obtained from sensor readings within the box. Secondary deliverables for this application include cloud connection, for storing data within and retrieving data from the cloud, and multiple box linking, for connecting multiple climate boxes together.

Humidity control was measured by measuring the humidity increase within the box from a state of equilibrium with the room to the upper humidity limit and then back down to a lower limit. Through this experimentation, we found that humidity control within our box was quite effective. We found an average humidity increase of about 1 percent every 10 seconds, with an upper limit of around 85 percent humidity. Naturally, the slope of increase for humidity within our box asymptotically converges on the maximum humidity. We found a time to reach maximum humidity from room levels of about 5 minutes. Venting the box proved to be effective as well, as we could reach lower humidity levels of around 44 percent (in a room with about 48 percent humidity) in about 12 minutes from 80 percent humidity. Again, humidity levels asymptotically converged on this lower limit. Rate of change at the beginning of the venting process was around 2 percent every 5 seconds, with the rate of change nearest the lower limit dropping to about 1 percent every 2 minutes. Below is a graph of our findings:

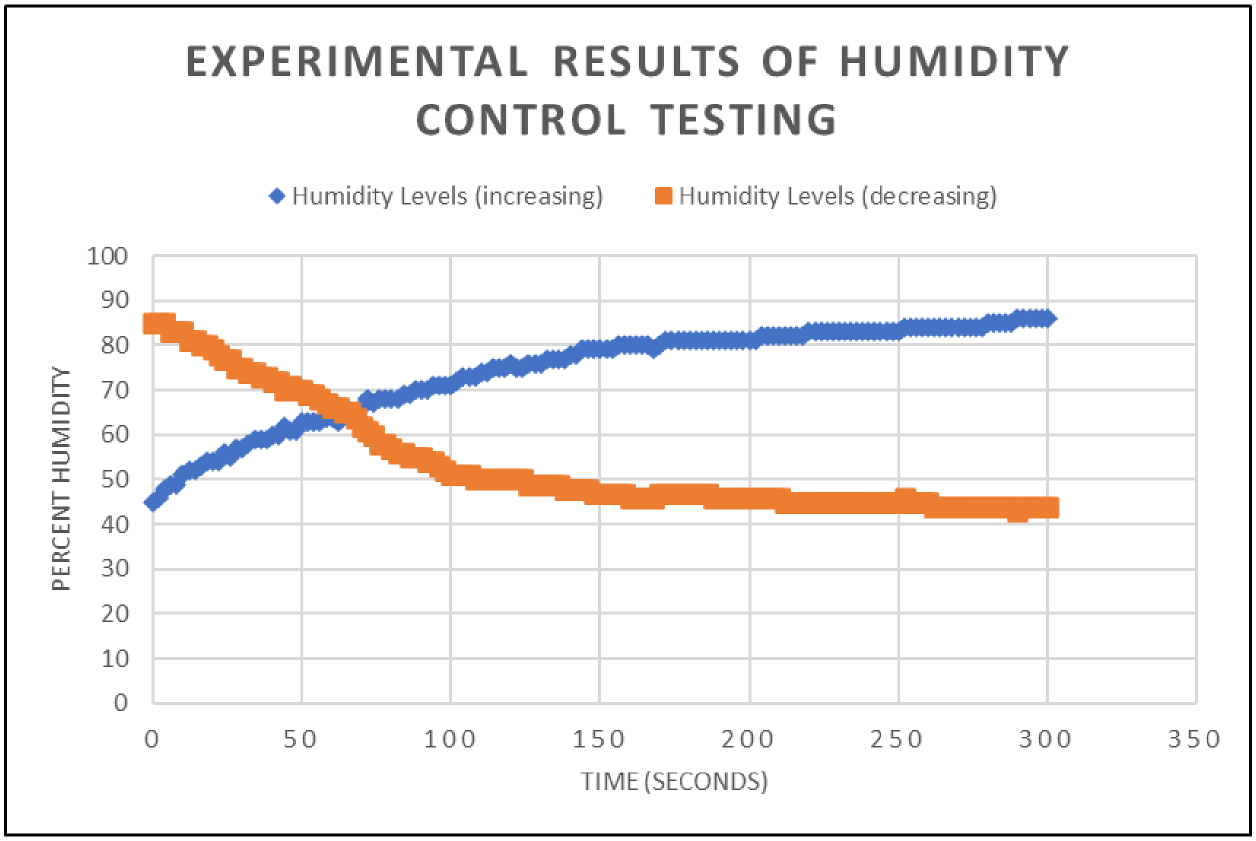


Figure 16: Results of humidity control testing.

Temperature control was measured in the same way, with heating from room temperature to a maximum temperature and then cooling. We found that temperature control was less rapid than the humidity control, however, this is to be expected given the size of the box. From a room of about 21 degrees celsius, we found that we could reach a maximum temperature of 34 degrees celsius in about 25 minutes. Although this had a slower than time-to-reach target value than humidity control, it was determined that temperature control was sufficient to meet our application needs, since the Terrabox was designed for long-term cultivation. Cooling proved to be quicker than heating. Cooling from 34 degrees celsius, we found it took 15 minutes to reach 20 degrees celsius. Currently, the Terrabox has no means of reducing temperature more than 1 degrees celsius less than room temperature. We noticed that temperature control was much more linear compared to humidity control. Below are graphs of our findings:

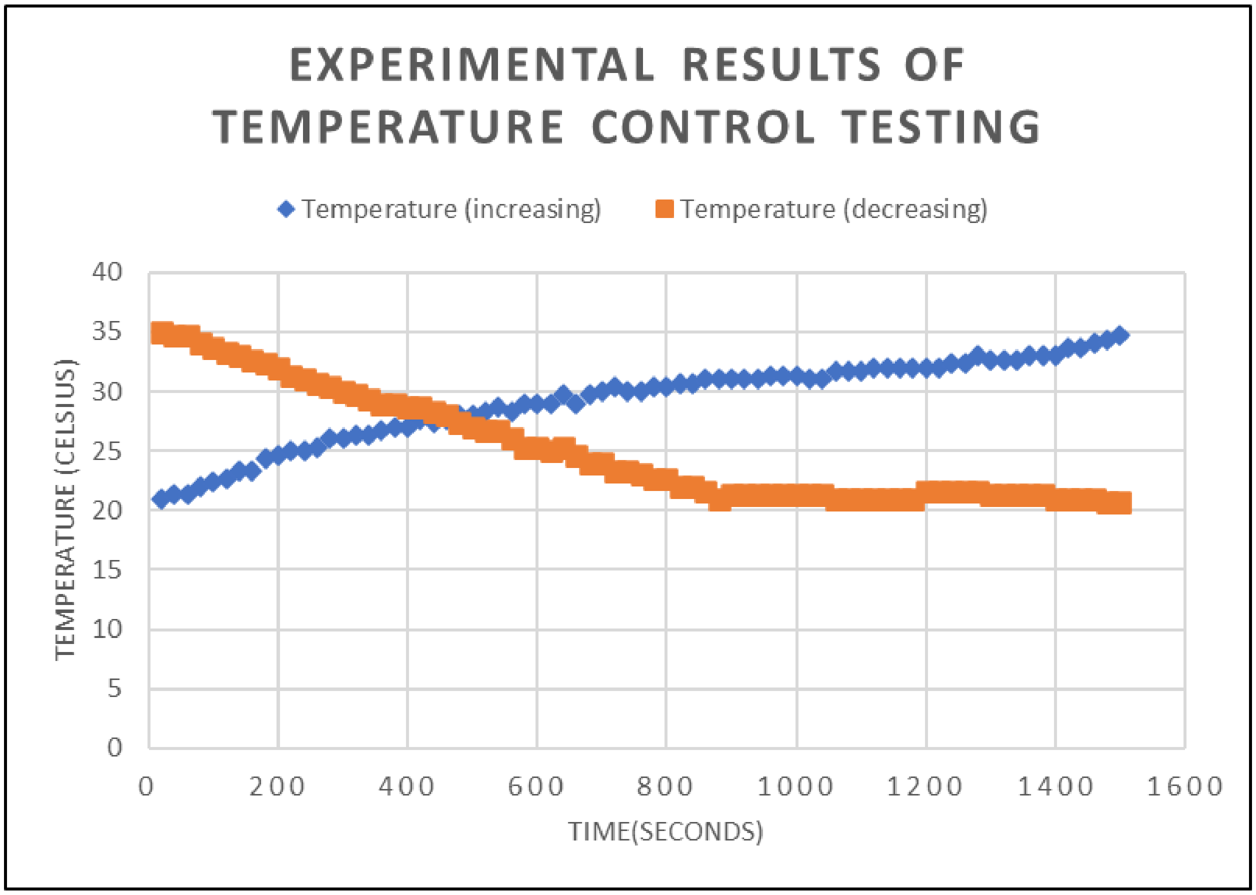


Figure 17: Results of temperature control testing.

The sprinklers were tested by installing a 20 second watering cycle, meaning the sprinklers activated for 20 seconds, every 20 seconds. This test had two purposes:

- Confirm that the timing generation algorithm of the Raspberry Pi was functional
- Confirm that the holes in the fiberglass tubing were of the proper size

We found that the code was functional and that the tubing was sufficient to meet our purposes.

The lighting was tested in the same manner as the sprinklers. Using a lighting cycle of 20 seconds, we were able to observe the lights switch on/off at the desired times.

Sprinklers and lights were tested together to ensure the timing of one did not interfere with the timing of another.

Through our testing, we found that there were slight delays in the intended time of our cycles for sprinklers and lights. The Raspberry Pi must wait each control loop for the STM to send data. However, the STM must allow for a 2 second delay between sensor reads in order to allow for the sensors to reset. This caused some discrepancies in the lighting and watering cycles we created. Therefore, the Raspberry Pi code accounts for these delays in software. Still, there are very slight delays in the desired time vs the actual time. For a 20 second cycle, we get a true cycle that is slightly higher than 20 seconds. With a 1 minute cycle, we observe a true cycle of around 61 seconds. Since the purpose of these watering cycles are for watering plants, we would expect a worst-case timing cycle of maybe a month (that is, a month without watering). Using the numbers provided below, we can calculate that a month-long cycle would result in a true cycle of one month plus a few hours. This was deemed to be acceptable for our purposes, since plants requiring this little water would not need precise watering times. Furthermore, this expectation is set based on worst-case behavior. It is possible that the watering cycles experience only about a one second delay regardless of water time. Due to time constraints, we were unable to fully test and therefore proceed with the assumptions given above.

Impact and Consequences

For any engineering product, those involved must consider the full scope of the impacts their product can have. This section will outline a broader scope of the product's potentials, both positive and negative.

Terrabox is designed to make home plant cultivation easier and more accessible. It provides growers with a fully automated habitat for plants to thrive, lessening time spent watering and tending to plants and expanding a growers selection of plants that require unique environments. However, growing plants in environments which they are not indigenous to can pose some threats. English Ivy, for example, was brought to the Americas by European explorers and settlers. As a result, the invasive ivy species has overtaken much of the native plantlife. It is generally unacceptable to introduce non-native plants to new areas for this reason. While the Terrabox allows for growing unique plants in areas they are not meant to grow, the user should be aware of these dangers and avoid introducing the plant to the wilds of their non-native habitat.

The Terrabox mobile application introduces a way for users to interact with others and establish a community which will hopefully promote safe and effective plant cultivation. The exchanging of ideas between users across many different ethnic and national backgrounds can help propel progress in the world of home cultivation. The presets feature can help those struggling with growing healthy and happy plants get the success they desire. However, all online platforms such as this can pose threats to the safety and mental health of its users. The application currently does not support any way to sensor or filter the content its users see. As a result, some users may experience unwanted harassment or vulgar content. The user, as well as us developers, should be made aware of this and establish workarounds for users experiencing any unwanted content.

The motivation for Terrabox was partly driven by improving environmental and social sustainability. Some plants, for example, are capable of feeding entire families in some areas but cannot be cultivated in other areas. Terrabox can allow families in the non-native climates of these plants to afford to feed themselves. Increasing the amount of individuals that can grow their own food can indirectly reduce the carbon footprint of farms. We can see that, more often than not, social sustainability and environmental sustainability can be improved in conjunction. Future iterations of the Terrabox will be designed with this goal in mind, by increasing the scope and usability of the box we can potentially help solve these ongoing global issues.

Conclusions and Recommendations

The slightest mistake in a home terrarium can be the difference between a successful or failed venture into the world of exotic plants and animals, and hobbyists have all experienced the pains of how fickle and high maintenance their plants and animals can be. To those people, the Terrabox is here to help. The Terrabox will help spread the hobby of growing and caring for plants or exotic pets by providing a new level of ease and accessibility never seen before. Where before, users would have to meticulously adjust the temperature, humidity, lighting, and watering cycles slowly and manually, now with the Terrabox, everything can be automated and made far easier for the end user. The many different environmental factors that can be controlled with the Terrabox will allow users to raise all sorts of different plants or animals, and the convenience provided by a phone application means that any adjustments that need to be made to the terrarium are only a few taps away.

While this Terrabox prototype delivers on the promised core functionality, there are certain aspects that must be improved upon for future iterations of the Terrabox design. Temperature control is limited by the surrounding room temperature, and while we can reach a sufficient upper limit temperature, plants that thrive in arctic environments will not find a good home in the Terrabox. For future designs, a cooling unit must be implemented to expand the amount of supported plants. All electronics are housed below the Terrabox habitat, and as such take up a lot of space. For future designs, electronics should be housed within a PCB that can easily be placed in a lower compartment to the habitat itself. Likewise, the water reservoir should be incorporated into the habitat itself. Lastly, due to budget constraints a plastic bin was used for the box itself. This should be upgraded to a more robust material and it is recommended that future iterations of the Terrabox be constructed from plexiglass.

Terrabox can have multiple applications, including supplying food, cultivating exotic plants, housing climate-sensitive animals, and more. The mobile application serves as a way to directly control and monitor the climate within the Terrabox, but also to generate a sense of community and progression as the presets feature gives users a way to exchange ideas, tips, and information. The basis for the Terrabox extends beyond just growing, it provides at-home hobbyists and naturalists alike a way to enjoy a more complete growing experience both individually and collectively.

References, Acknowledgements, and Intellectual Property

Software Side:

Rachel, Pawan, Sudeshna, Jaymeen, Niels, Mac, H. Craddock, J. Rota, andrej_m, vlads_twttr_and_ig_reports, Kévin, A. Targaryen, Alex, C. Ching, Mc, and Guest, “Uipickerview example and tutorial (updated 2018),” *CodeWithChris*, 25-Sep-2019. [Online]. Available: https://codewithchris.com/uipickerview-example/. [Accessed: 10-Mar-2022].

IOS Academy, “Swiftui: Email / password sign in with ... - youtube.com,” *Youtube*, 03-May-2021. [Online]. Available: https://www.youtube.com/watch?v=vPCEIPL0U_k. [Accessed: 10-Mar-2022].

Replicode, “Ordering & paginating data - SWIFT + firebase part 8 - youtube,” *Youtube*, 28-Aug-2018. [Online]. Available: https://www.youtube.com/watch?v=4y_NLkYT6NU. [Accessed: 10-Mar-2022].

Twitter. [Online]. Available: https://twitter.com/. [Accessed: 10-Mar-2022].

Apple XCode forums
Geeks for Geeks
Controllers Tech

S. Allen, “Swiftui - TabView tutorial - YouTube,” *YouTube*, 28-Jan-2021. [Online]. Available: https://www.youtube.com/watch?v=tnNFoZ7CkP8. [Accessed: 10-Mar-2022].

How to get the current time in Python. Stack Overflow. (2009, January 6). Retrieved February 20, 2022, from https://stackoverflow.com/questions/415511/how-to-get-the-current-time-in-python

Programiz. (n.d.). How to get current date and time in python? Programiz. Retrieved February 20, 2022, from https://www.programiz.com/python-programming/datetime/current-datetime

Python. (n.d.). Datetime - basic date and time types¶. datetime - Basic date and time types - Python 3.10.2 documentation. Retrieved Feburary 25th, 2022, from https://docs.python.org/3/library/datetime.html

Hardware Side:

adafruit, “DHT11 basic temperature-humidity sensor + extras”. [Online]. Available: https://www.adafruit.com/product/386?gclid=CjwKCAiA4KaRBhBdEiwAZi1zzh8KMSYd3DQgKGefg2wdxeyCI-MZzTIbuFs6Xc6XvULibJ7O1rl3ZhoCj5kQAvD_BwE. [Accessed: 16-January-2022].

Seeed, “Grove - Water Atomization”. [Online]. Available: https://www.seeedstudio.com/Grove-Water-Atomization-v1-0.html. [Accessed: 16-January-2022].