IrrEEgation: Final Report

By: Tyler Dale, Nicolas Garcia, Ellen Halverson,

Kenneth Harkenrider, and Trenton Kuta

Table of Contents

Introduction

Flood irrigation is a process vital to the southwestern region of the United States. Flood irrigation denotes a process in which a field is soaked in a film of water. Typically, this is accomplished by opening a valve that causes water to surge at a high volumetric flow rate. While this water begins covering a field quickly, it becomes a thin film of water as it travels farther from its source. This film slowly covers the field until all the crops are watered. The larger the field, the longer it takes for a flood to be completed.

Unfortunately, farmers must carefully monitor the progress of the water. Allowing the water to flow from the source longer than it should greatly increases the amount of water used in a cycle, negatively impacting the environment by wasting water and affecting other farmers or civilians if water reaches their fields or roads. On the other hand, it is essential that the film of water reaches all crops in a field; otherwise some crops may not yield produce. A tight window exists in which all crops have been watered, but extra water has been minimized. At this time, a farmer must leave his house and manually shut off the valve controlling the water flow.

However, this window is unpredictable. Factors such as wind, temperature, and inconsistent water pressure can vary the rate and direction of the film of water. As a result, a farmer must constantly check the level of the water during the flooding process. For large fields, the flooding process can take anywhere from 24 to 36 hours. Toward the end of this period, the farmer must periodically walk the field at any hour of the day or night to check the water level and determine whether or not the water should be shut off. To make matters more difficult, the flooding process needs to be completed every two to three weeks. As a result, a great deal of a farmer’s time involves him or her being confined to his or her property and manually controlling the flooding process. Rather than taking up an unnecessary amount of a farmer’s time, the process of monitoring and controlling the flooding can be automated with a system involving a central hub, a web app, water sensing pylons, and a motorized valve.

The IrrEEgation Control System is managed by a central hub that monitors device statuses and user commands. Based on user inputs and device statuses, the central hub wirelessly relays commands to the other components of the system. The hub can be set up at the user’s residence or somewhere near the irrigating field; it only requires source of 120 VAC as provided by a typical power outlet. Theoretically, a DC battery could run the hub as well, but 120 VAC is used since the central hub is meant to be placed in the user’s home, where power outlets are abundant.

The hub identifies with several water-sensing pylons. Knowing his or her field, the farmer can place the pylons in the most advantageous positions to monitor the presence of water, such as near the edge of the field, to ensure the valve shuts off. The pylons, powered by a photovoltaic and lithium ion battery system, can sense the film of water and relay this status to the hub. If any valve are linked to that pylon (per the user’s instruction), the valve will automatically close.

The valve itself is simulated by a linear actuator motor rated for the force required to open a floodgate. Based on the state of board-mounted relays, the motor can extend or contract, simulating the opening or closing of the actual floodgate.

Finally, a web app can be accessed to observe the flood progress throughout the process. The web app writes to the directory containing files and instructions for the devices and prepares the commands to be sent out.

At the end of the semester, the final product met the basic requirements for having a well-functioning system capable of doing what was intended. At the final demonstration, the pylon was able to be placed in either an awake or sleep state from commands issued from the website interface. In the awake state, the pylon successfully reported its status (wet or dry) to the the central hub and listened for commands from the user. If the user did not detail new commands for the pylon, the pylon defaulted to its prior sleep setting and continued monitoring moisture.

The pylon battery was successfully charged from the photovoltaic cell. The pylon was added and removed from the system easily, and the pylon's abilities to connect to the server and remember its identity worked better than expected - the modular naming and syncing functions were present. More importantly, the valve responded appropriately to the status of the pylon associated with it. However, there were two major areas in which our design fell short. Due to the lack of continuous sunshine in South Bend, the solar charging system charged the battery very slowly. Thankfully, the battery's ability to hold a sufficient amount of charge for a long period of time compensated for the slow charging capability. It's important to note that this system was designed to function in areas with powerful and consistent sunlight, such as New Mexico, and the constant sunlight in these locations would eliminate this liability. The other major shortcoming in the design had to do with controlling the valve - it was not as straightforward as the team had hoped. A single valve board did not have the stability to both run the onboard microcontroller and power the relay/actuator system. For the device demonstration, the team compensated by wiring two identical boards together.

Aside from these small hurdles, the system performed very well. The server read from and wrote information to the device directory in real time and in tandem with the looping functions; there was minimal disconnect between the user web application interface and the actions/status of the devices. Overall, aside from a lag built into the device sleep/awake cycles, the system as presented to the user was seamless, straightforward, and robust.

Detailed System Requirements

The current system for flood irrigation is ineffective, time-consuming for farmers, and wasteful of precious natural resources, which are already scarce in the region. In order to address each of those issues and to make the process more efficient, the previously necessary manual checks done by the farmer will be done reliably by a system of water-sensing pylons and automated valves. Ultimately, the system must allow the farmer to monitor the water levels remotely with confidence.

In order to adequately monitor the flood irrigation on a farm, the irrigation system has to maintain a minimum resolution; it needs to check the status of the water often enough to give the farmer useful real time information. Furthermore, it is unrealistic to expect the farmer to understand file maneuvering or circuit repair; the whole system must be user friendly. That being said, the general overlay of the system (water sensing pylon, valve device, and central hub) must be accessible and clear. For this reason, the team decided that an interactive web app should be part of the hub as the interface for the farmer.

First, the central hub must be responsible for supporting the communication framework for the pylons, the user interface, and valves. The central hub needs to continuously track data from all devices, publish this data to its interface, and relay commands from the user to the devices. Most importantly, the hub needs to perform these actions in real time, constantly checking data and responding immediately to the user's commands. The hub must also be a wireless device in terms of how it interacts with the devices in the field. It does not need to be self powered (it may be an appliance in the user’s residence or near the field) but it must have adequate range to maintain sufficient communication.

The necessary interface must be part of the central hub - or at least associating extremely closely with it. The interface must display the status of all pylons and all valves to the user in real time in addition to giving the user an array of options to affect the system with. These options include: renaming a device already on the system, resetting a device on the system, setting devices to awake or sleep mode, syncing devices, commanding valves to open or close, and assigning a valve to a pylon. The interface must clearly present the options and promptly carry out the user’s orders.

The pylon subsystem must have a reliable and accurate means of determining whether or not water is present. The moisture sensor itself must be either cheap and easily replaceable or robust and long lasting if installation is bothersome. As a standalone wireless system, the pylon must also have adequate range (at least 200 m) and wireless capability to keep the hub notified of its moisture status. This also implies that the pylon system will need an independent source of power, most likely a solar cell-battery combination. The pylon system will also need a mode of regulating down time; it will need sleep and awake functionality for efficient energy use during different duty cycles. This also implies that crucial information for the function of the pylon needs to be stored in some sort of non-volatile memory. If the system goes down, it needs to remember any necessary wireless credentials or process-critical information upon boot. Finally, the pylon system will be standing out in a field for several weeks at a time - it needs to be robust and weather resistant.

The valve system has similar constraints to the pylon system: it must communicate wirelessly with the hub and receive instructions remotely. In addition, it must successfully open or close in response to commands from the hub. It is also critical that the status of the valve be reported back to the central hub and to the farmer by extension. As with the pylon, the valve's wireless communications must be reliable and robust. Because the valve system has to be a wireless node, it faces the same power and range constraints of the pylon system. However, in addition to needing at least 200 m of range, the valve system also needs sufficient independent power to open or close the motor. Of course, the valve also needs to be weatherproof, have sleep/awake capabilities, and be able to store essential identifiers to non-volatile memory.

Additionally, from a system perspective, the valve must close in response to its assigned pylon's "wet" signal. However, this command path may also be routed through the hub.

In reality, the valve is a large metal panel, which is raised or lowered in order to seal a water pipe. This arrangement is impossible to build for and test given the team's budget constraints. Rather, the team has chosen a more reasonable constraint: the action of opening or closing can be simulated with a small, powerful linear actuator.

These are the most pressing constraints the system as a whole needs to address in order to guarantee a user friendly interface with maximal functionality.

Detailed Project Description

System Theory of Operation

This product combines several devices via wireless MQTT communications to create an autonomous irrigation system. The system incorporates a user interface, valves, pylons, and a centralized hub, which, together, are able to monitor and control the presence of water in the field. Operation begins and ends with the valves. When the user wants to begin irrigating his or her crops, he or she can remotely open any valve using either the app or the LCD touch screen on the hub. The hub then sends an “open” command via MQTT to the valve. Once the valve is opened, water begins flowing throughout the field. When a pylon senses the presence of water, it relays this information to the central hub. The hub processes this information and writes it to the relevant file in the database. The app updates to reflect the new information and the user is able to monitor the status of all available pylons throughout the irrigation process and remotely close the valve at any time. Alternatively, the user can also choose to have any valve close automatically when a specific pylon is triggered. In sum, this system completely automates the flood irrigation process while also providing options for manual control and customization.

System Block Diagram

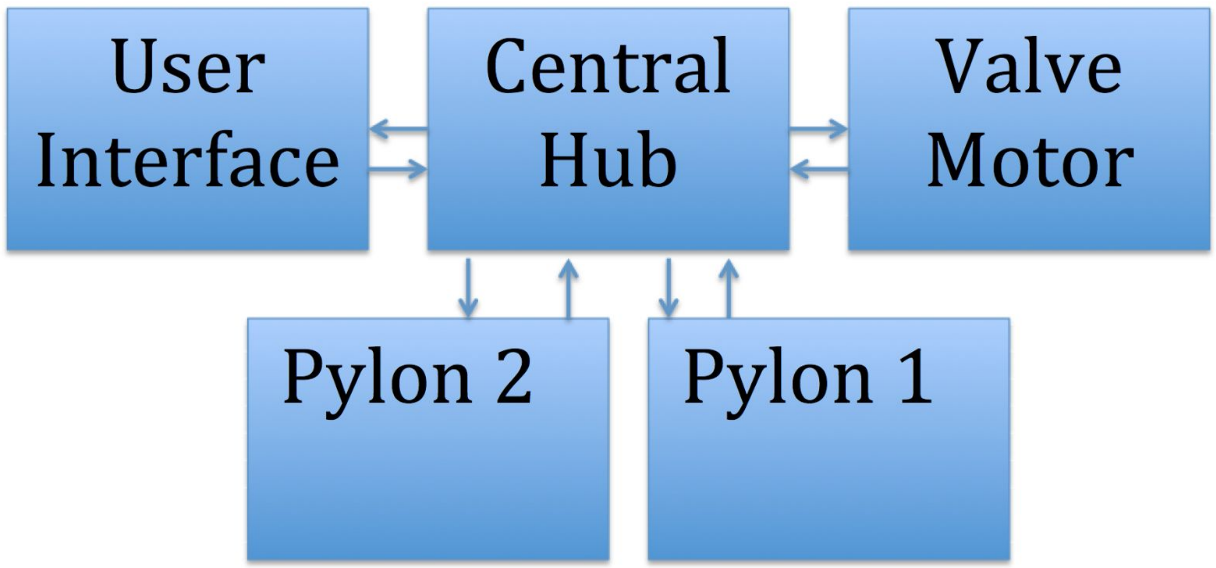

Figure 1. Block diagram showing connections between each subsystem.

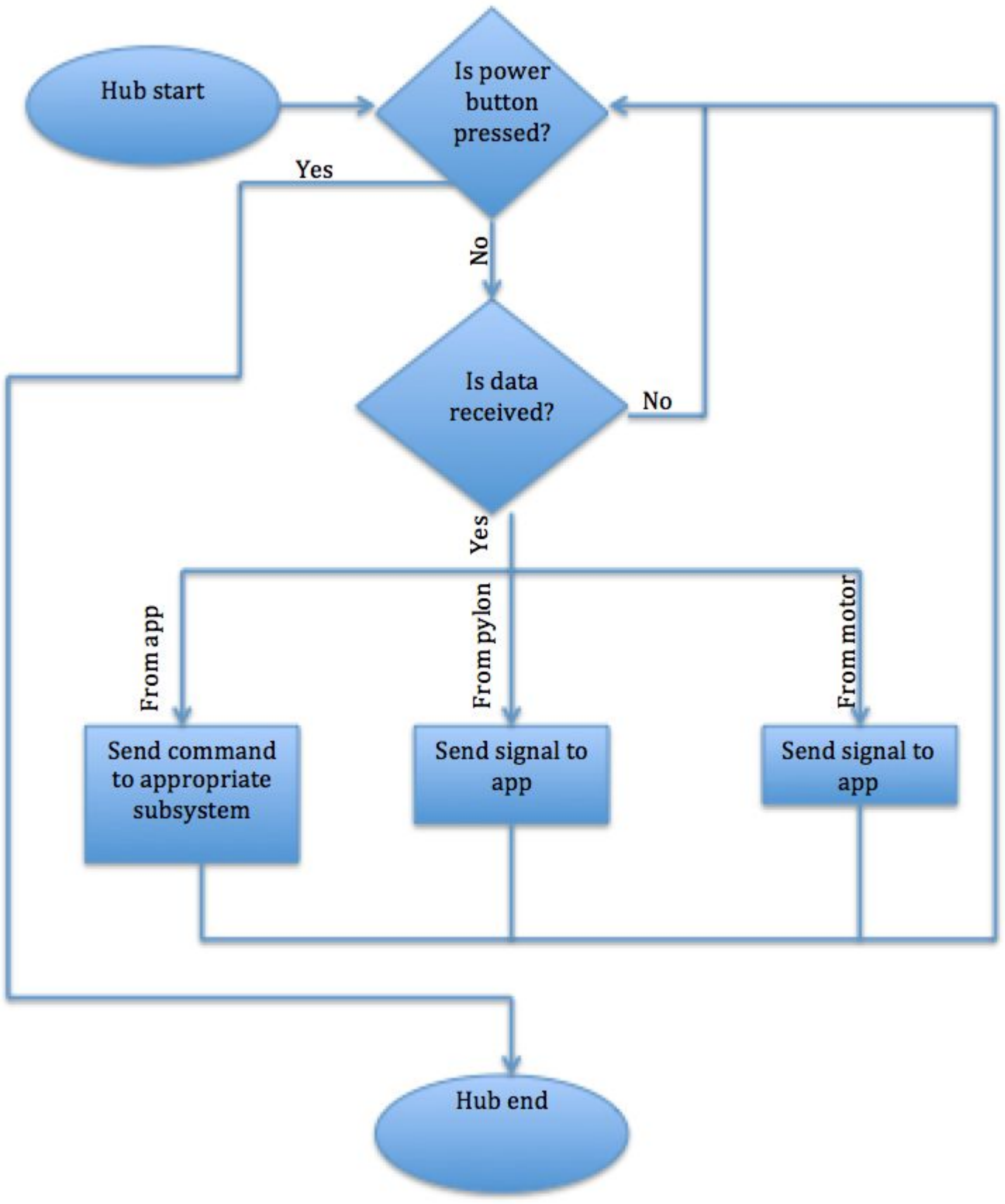

Figure 2. Block diagram for the central hub.

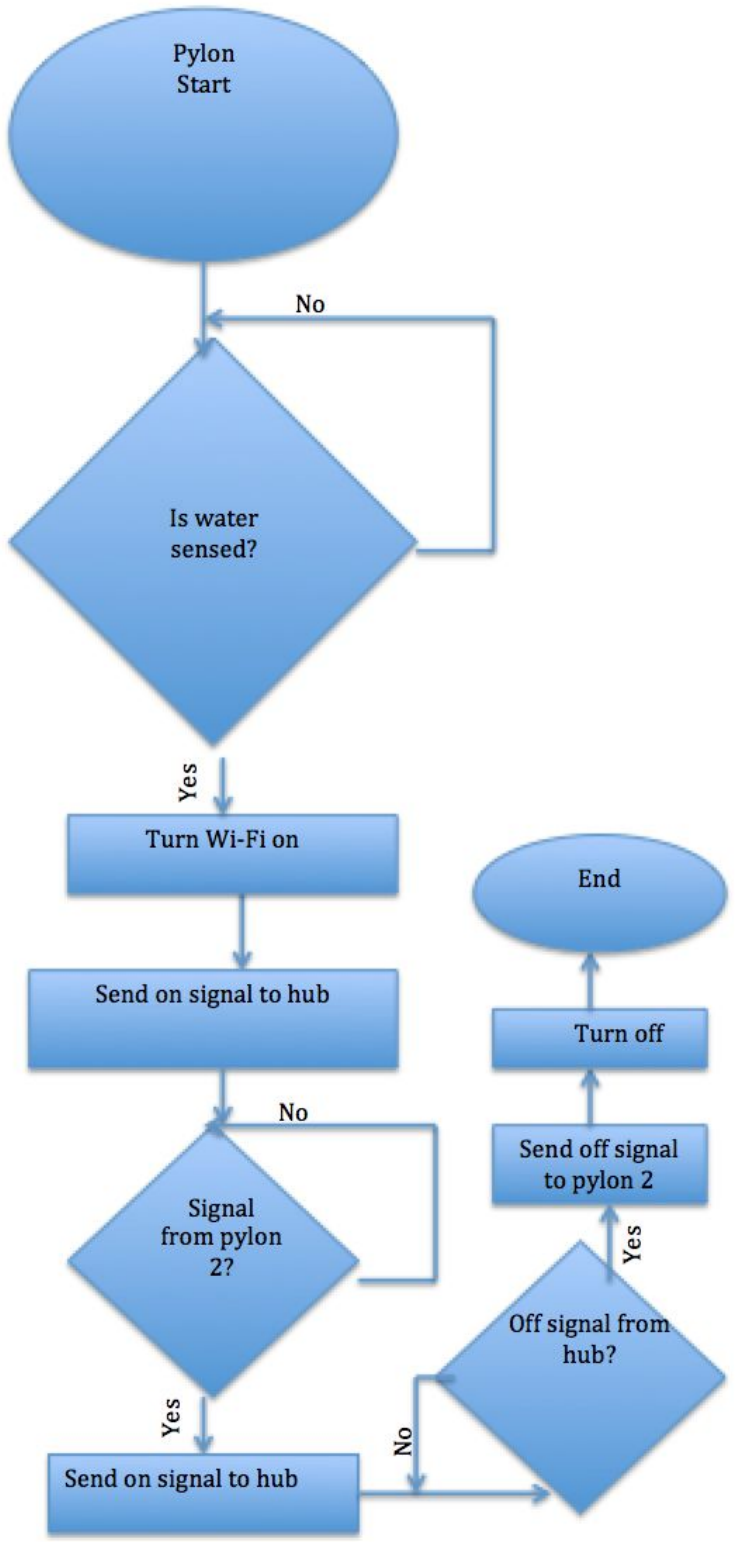

Figure 3. Block diagram for the pylons.

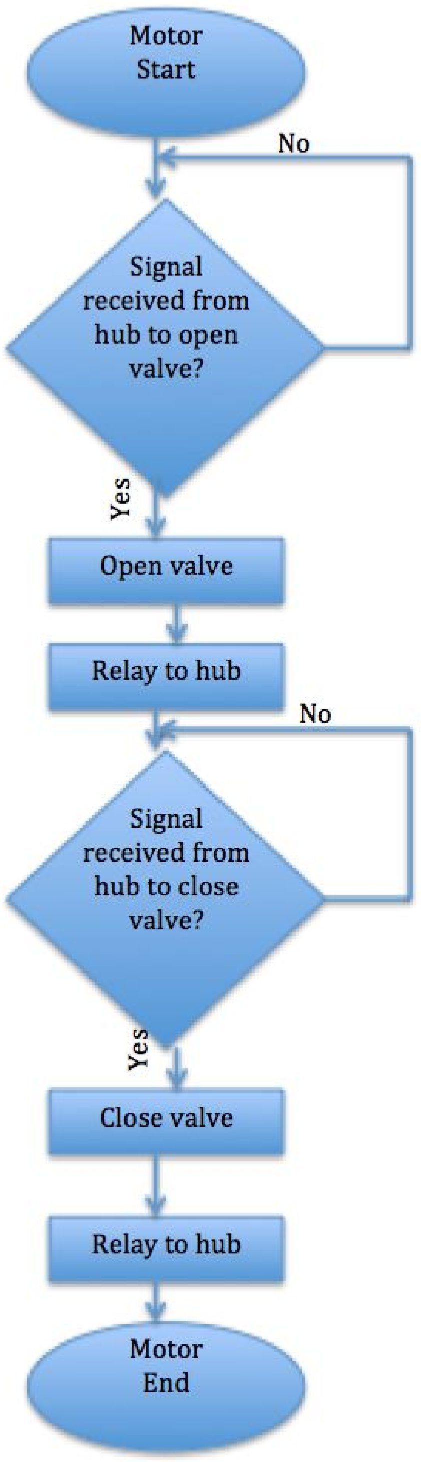
Motor Start
No
Signal received from hub to open valve?
Yes
Open valve
Relay to hub
No
Signal received from hub to close valve?
Yes
Close valve
Relay to hub
Motor End

Figure 4. Block diagram for the motor.

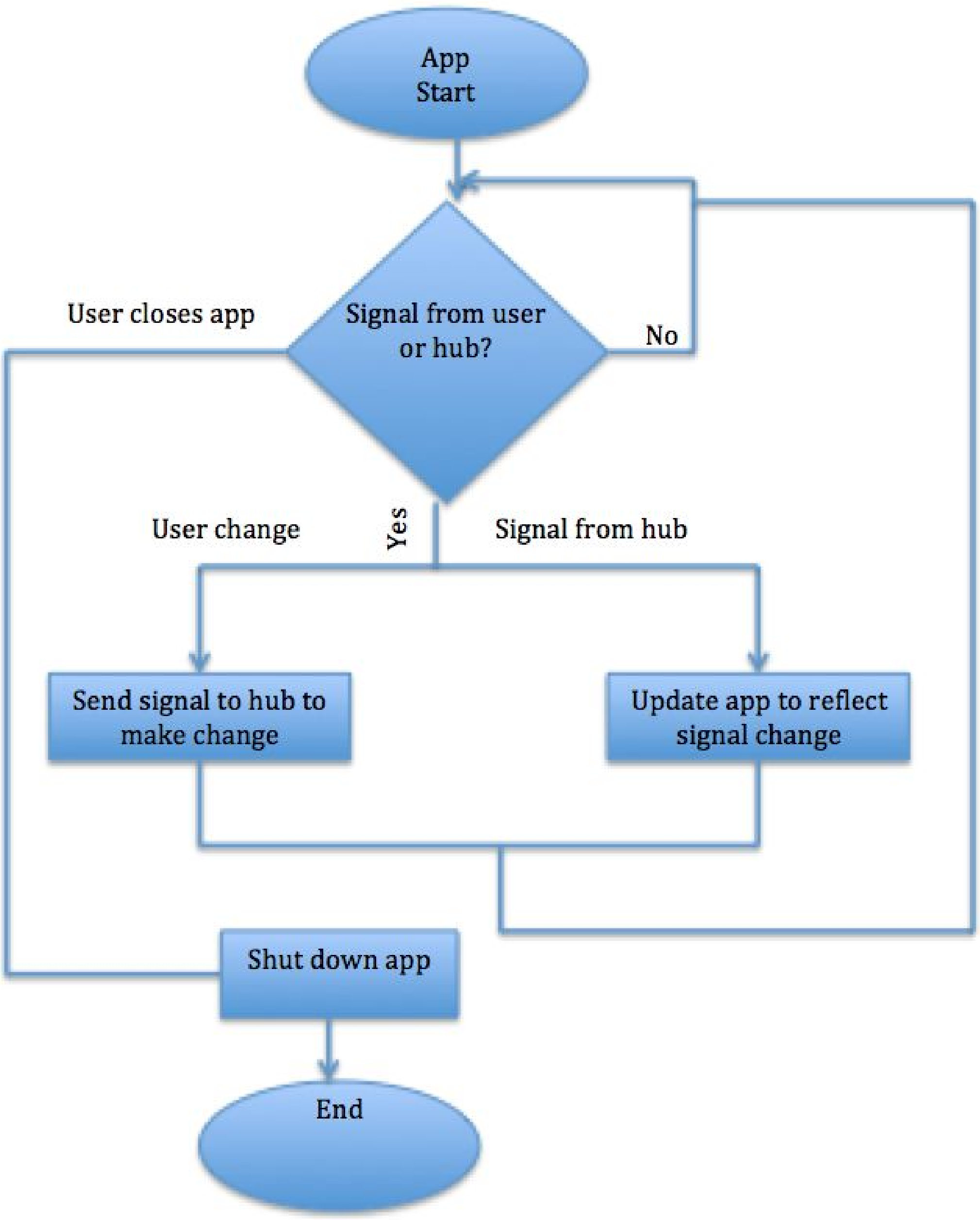

Figure 5. Block diagram for the user interface.

Subsystems: Valve

Software

For the wireless communications protocol, MQTT was chosen. MQTT is ideal because of the abundance of resources (found both in online forums and IDE libraries) and its relative ease of implementation. The MQTT code is enveloped in the general C code for the pylon microcontroller, programmed from the Arduino IDE. The Arduino IDE is employed for convenience; MQTT and microcontroller code are free, readily available, and easily uploaded into a microcontroller.

The microcontroller on the valve is an ESP8266, so chosen because of its Wi-Fi capabilities. The chip itself does not need to perform any heavy computations and is primarily the medium of relaying moisture measurements. A desirable aspect of the ESP8266 is its ability to enter deep sleep. This is also a system requirement, as the valve will go long stretches of time where measurements will not be needed, such as when irrigation is not taking place.

A critical constraint on the microcontroller is non-volatile memory. Whenever the valve activates or wakes from sleep, it will need to know the Wi-Fi credentials of the hub. In general, these will be saved into the ESP8266's EEPROM from the last sync operation. If the valve does not have these details saved, it becomes a Wi-Fi access point and allows the hub to connect and transmit its credentials. These details are written and saved into the EEPROM for future reference and can be erased in the event of another sync. A push button is included on the valve to allow manual initiation of a sync and an LED is included to indicate sync progress.

The valve wirelessly updates the hub with frequency that depends on whether or not the valve is in sleep or awake mode. In both modes, the valve undergoes an amount of sleep before running an active cycle; the sleep and awake modes only dictate the length of that time. In practice, a wake/sleep command from the hub determines how long the valve will sleep before undertaking the next cycle. This means that the valve's activity is regulated on a cycle-by-cycle basis from the hub. Each of these cycles consists of the valve waking up, reading its non-volatile memory, connecting Wi-Fi/MQTT, opening and closing based on hub instructions, and finally listening for commands from the hub before sleeping again.

The details of this process can be found in the Appendix C . Here the full listing of the code is shown with comments highlighting the process flow.

Hardware

Solar Battery Charging Subsystem

One of the major requirements of the Flood IrrEEgation Control System is that the pylons and the valve do not need to be removed during irrigation. A consequence of this requirement is that the pylons and the valve need to be able to run off battery power for a considerable amount of time.

To counter the possible issue of the batteries draining below operable levels during irrigation, each pylon and valve includes a solar charging subsystem. This subsystem is built around the LT3652, a power tracking battery charger chip produced by Linear Technologies. The LT3652 takes the voltage drop across a solar panel and outputs a variable voltage and current. A general schematic for a typical application can be found below in Figure 6.

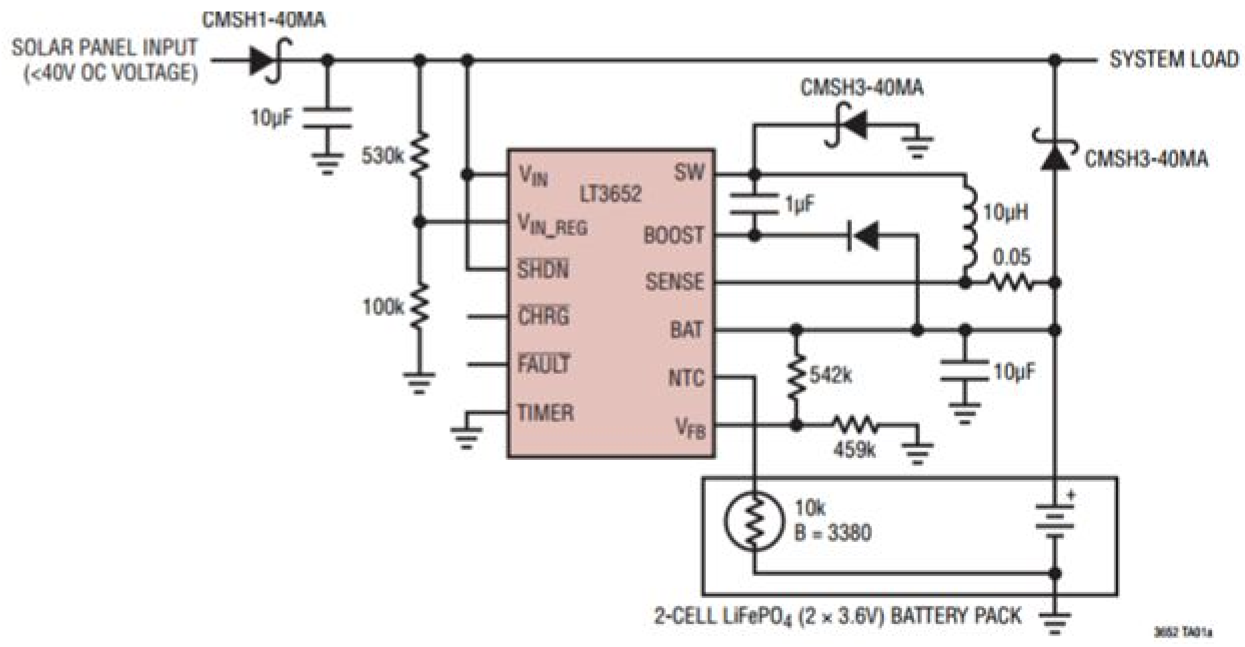

Figure 6: Typical Application Schematic for the LT3652

Some important parameters considered when choosing this device was the minimum input voltage (4.95 V, which was less than the 18 V solar panel used in the design), output current (up to 2 A), and output voltage (up to 14.4 V, which is greater than the valve board's 12 V). To adapt the LT3652 to the specific application required for the Flood Irrigation Control System, three parameters were set: the input regulation voltage, the feedback voltage, and the current output.

One of the options that the LT3652 offers is a minimum supply voltage regulator. If the voltage on the pin is less than 2.7 V, the LT3652 ceases to operate. For the Irrigation System, this value is set at 4 V because this is the minimum voltage required to charge the one cell LiPoly. This is accomplished using a 4.7 kΩ resistor for R_{IN1} and a 10 kΩ resistor for R_{IN2}. These resistors correspond to the 530 kΩ and 100 kΩ resistors on the input side of the LT3652 on the

Typical Application Schematic previously marked as Figure 6. The equation relating the resistor values, the set minimum operating voltage, can be seen below in Figure 7.

$$R_{IN1}/R_{IN2} = (V_{IN(MIN)}/2.7) - 1$$

Figure 7: Equation for setting the Input Regulation Voltage

The LT3652 has the ability of charging batteries with voltage ratings from 3.3 V to 14.4 V. This output charging feedback voltage is set via a voltage divider across the V_{FB} pin on the output side of the LT3652. The equations used to determine the resistor values for this voltage divider come from the datasheet and can be found below in Figure 8. The valve and the pylon board have different voltage requirements, 12 V and 4 V, respectively, so different resistor values were required for the boards. On the valve board, R1 was chosen to be 931 kΩ and R2 was chosen to be 340 kΩ. For the pylon board, R1 was set at 312 kΩ and R2 was set at 1.27 MΩ. These resistors correspond to the 542 kΩ and the 459 kΩ resistors, respectively on the output side of the LT3652, as seen in the aforementioned Figure 6.

$$R1 = (V_{BAT(FLT)} \bullet 2.5 \bullet 10^5)/3.3$$
$$R2 = (R1 \bullet 2.5 \bullet 10^5)/(R1 - (2.5 \bullet 10^5))$$

Figure 8: Equation for setting the Output Feedback Voltage

The last major characteristic set using hardware components was the charging current maximum. This current was set using two .22 Ω resistors in parallel, resulting in a practical resistance of .11 Ω. The output current was set this high as to minimize the time required to charge the batteries. Unfortunately, the output power was too large for the chip to handle, so the resistance was increased to 1.5 Ω. The equation describing the relationship between the resistance and the charging current maximum can be found in Figure 9. This resistor matches the .05 Ω resistor found on the output side of the LT3652, as seen in Figure 6.

$$R_{SENSE} = 0.1/I_{CHG(MAX)}$$

Figure 9: Equation for setting the Maximum

In addition to the resistor values set in the last three steps, a few other minor changes were made to the suggested circuit. The thermistor, used to shut down the LT3652 in extreme temperatures, was omitted due to the concern that a thermistor may get hot under normal operating conditions in the New Mexican sun. Also, the direct connection between the output of the solar and the battery was disconnected in order to force all current, stable or otherwise, through the stabilizing chip.

ESP8266 and Antenna

For both the pylon and valve boards, the base ESP8266EX microchip is utilized with separately mounted components, including a crystal oscillator, flash memory chip, and antenna hookup.

The crystal oscillator chosen for this project is the TSX3223, which was chosen because the oscillator runs at the required 24 MHz and is stocked at Notre Dame. The oscillator is paired with two closely mounted 10 pF capacitors to smooth out the clock. Also included near the ESP8266 is the AT25SF041 flash memory chip, which was chosen because SparkFun utilizes the chip in its ThingDev board. The antenna connection was specially designed for this project. The input impedance of the ESP8266EX was determined to be 50 Ω, so the trace width between the microcontroller and the antenna hookup was 50 Ω. This was determined using the ADS package with help from Dr. Chisum.

In addition to the standard operation requirements, the ESP8266EX on these two boards control and read a variety of different input and outputs. On both boards, GPIO5, set in input mode, is pulled high until an external button is pushed, at which time the pin is pulled low, and the ESP8266 enters sync mode. GPIO4, set in output mode, is set high to power an LED indicating communications with the central hub are functioning. The ADC pin, also marked as “TOUT”, connects to the signal port of the humidity sensor to determine the presence of water. GPIO12, set in output mode, turns the humidity sensor on and off. For the valve board, GPIO13 and GPIO14, both set in output mode, respectively control direction of and power to the linear actuator. The exact method of this control is explained in a following section, and a schematic of the entire system can be found in Appendix A.

Humidity Sensor

The main purpose of this system is to determine the presence of water at a point in the field; this sensor accomplishes that goal. The sensor used is a resistive SparkFun Soil Moisture Sensor. Based on the conductivity between the two probes on the sensor, the sensor outputs an analog signal. This signal is routed through a unity gate buffer, which for this project is a regular operational amplifier, and is passed to the ADC pin of the ESP8266EX. Power to this sensor is controlled using an NPN transistor, controlled by GPIO12 on the ESP8266EX, between VCC and the sensor’s source voltage. A picture of this sensor can be found below in Figure 10.

Figure 10. SparkFun Humidity Sensor

DC to DC (Valve)

For regulating the VCC for the pylon board, a TPS54202H was placed on the board. This chip was chosen as the TPS and can output a 3.3 V signal with up to 20 V input (the pylon board uses a 12 V LiPoly battery), and does not waste much power. The TPS54202H was installed exactly matching the datasheet.

Linear Actuator Control System

The primary purpose of the valve board is to open and shut the floodgate. This goal is accomplished using a system of transistors and relays to control the power and polarity of a fill-in linear actuator. Two NPN transistors control current flow through the coils of a SPDT relay and a SPST relay. These relays were chosen to use the 12 V supply already present as well as the relatively low current draw when the coils are on. The SPST relay connects the actuator to ground with no current through the coils and connects the actuator to power when current is applied to coils. The other relay, the SPDT, switches the polarity of the voltage (when the SPST relay is on) across the monopole linear actuator. The SPST transistor is controlled by GPIO14 from the ESP8266EX, while the SPDT is controlled by GPIO13. A diagram of this system can be found in Appendix A. Finally, the linear actuator used for the pylon board is the ECO-WORTHY 10" (250mm) Stroke Linear Actuator. This actuator was chosen because it can handle a 330 lb load with only 12 V. A picture of this actuator can be found in Figure 11.

Figure 11: ECO-WORTHY 10"(250mm) Stroke Linear Actuator

Subsystems: Pylon

Software

The software for the pylon is nearly identical to that of the valve. The code commands the device to check its EEPROM upon waking and either enter standby mode or jump straight into MQTT depending on whether the identifiers are found there. However, instead of sending an "active" signal to the hub, the pylon reads its moisture sensor and sends the sensor status ("wet" or "dry") to the hub. From there, it erases any EEPROM if a reset or identity reset command was sent and proceeds to sleep. This full code with comments can be found in Appendix B.

Hardware

The hardware for the pylon is more or less identical to the valve board, minus the absence of the Linear Actuator Control System and a different TPS. For regulating the VCC for the pylon board, a TPS61201 was placed on the board. This chip was chosen as the TPS is stocked at Notre Dame, outputs a 3.3 V signal with up to 5 V input (the pylon board uses a 3.7 V LiPoly battery), and does not waste much power. The TPS61201 was installed exactly matching the datasheet.

Subsystems: Central Hub

Software

The software requirements for the Central Hub include interfacing with and running the web application, communicating with devices, and processing information. The bulk of the hub software was written in Python, one of the Raspberry Pi's native programming languages. Python was chosen over other potential languages because of ease of implementation, functionality and flexibility, and availability of resources.

The hub is the MQTT broker and the single unifying subsystem; it communicates with the app, pylons, and valve and relays messages between them (per the Publish/Subscribe format). As such, it is currently a Raspberry Pi 3 and is the single most intelligent subsystem.

After the initial setup of the Raspian OS, the Mosquitto MQTT broker and client are installed on the Raspberry Pi in order to host and publish and subscribe using MQTT. The team tested the device's MQTT capabilities using an MQTT simulator, MQTT.fx, and was successfully able to publish and subscribe between multiple devices over the new broker.

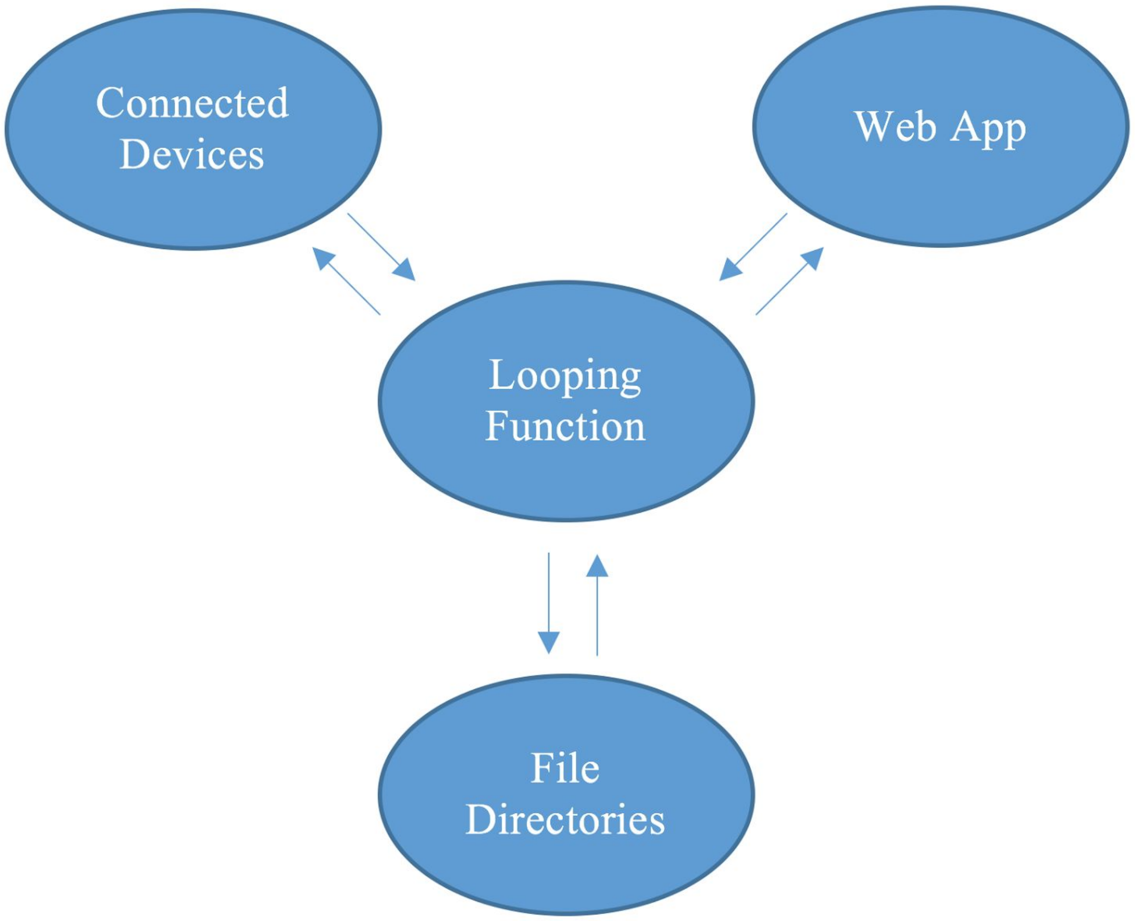

Figure 12: Flowchart of Hub software

In order to create and interface with the web app, the Flask language, a Python subset, was used. Flask acts as a bridge between raw Python code and HTML, essentially triggering Python functions upon visiting an HTML webpage. By implementing Flask, the team was also able to develop the Hub's business code and professional code separately, rather than in large, complex, integrated scripts.

To process the incoming and outgoing information, the team decided to develop a centralized file directory system that would organize and store all of the necessary data. This data could then be relayed directly to and from the web app natively running on the Raspberry Pi and communicated to all connected devices over MQTT. MQTT wireless protocol was incorporated into the hub scripts using the Paho MQTT library for Python. The hub could then publish commands to the MQTT topics that the devices would be listening to and also subscribe to the topics that the devices would be publishing to.

Essentially, the hub holds two lists at all times. The first list contains details on every pylon connected to the hub and includes qualities such as their names, connection statuses, and whether they are wet or dry. The second list contains details on every valve connected to the hub and notes details similar to those of the pylons. The hub then runs in an infinite loop over MQTT, during which time it automatically updates the statuses of all valves and pylons approximately every thirty seconds. This system is entirely autonomous, leaving a reliable system that the farmer does not need to monitor or worry about. That being said, the user can monitor and command the hub at will using the app.

Because the Hub was designed to run constantly on AC power at home, we developed a looping function that would read all device statuses and send commands based on these statuses and other received information. The looping function, as well as the web app server and IP address functions, can be found in Appendix D.

When completed, the Hub subsystem was first tested by running the looping, IP address, and web app server scripts together and simulating MQTT communications with MQTT.fx. Then, further testing was done to incorporate working pylon and valve devices.

Hardware

As mentioned above in the software requirements, a Raspberry Pi 3 was incorporated as the central hub. A 3.5 inch Kuman TFT touchscreen was also included to allow the user to access the web application without the need of a separate wireless device. Finally, we designed and built a 3D printed shell for the Raspberry Pi for protective and aesthetic purposes.

Subsystems: User Interface

Software

The software requirements for the user interface include interfacing with the Hub and with the user. As stated above in the Hub requirements, we implemented the Flask programming language to bridge the gap between the functional Python code and the HTML code of the actual user interface. We decided to develop a web application rather than a typical mobile application so that it could be accessed on any wirelessly-enabled device.

The main way the subsystems interface and communicate with one another is through the MQTT server, which has been described above. In the Raspberry Pi linux system there is a directory which holds text files, each of which corresponds to either a pylon or a valve. Whenever a pylon or valve is synced to the system, the text file is generated. Inside the text file, is the the devices identity, state whether it is asleep or awake, ID reset, reset connected valve or pylon, and whether it is wet or dry if it is a pylon or open or closed if it is a valve. Each time the pylon/valve wakes up it sends a status report via MQTT to the central hub, which is constantly looping and looking for this data. As soon as the hub receives the information it edits the file and saves the new status. Simultaneously, the web interface is constantly checking and reading those text files and updates the status of the device on the website. Similarly, whenever the user alters information for a given device, this central hub changes the information in the text file. Therefore the next time the device wakes from sleep, the alterations to the text file will be transmitted as commands across MQTT, and the status of the device will alter to meet those changes.

System Integration Testing

Description of how the integrated set of subsystems was tested

With the complex array of subsystems comprising this project, testing was expansive and complicated. The main test the project needed to pass involved taking inputs from the pylon, reading from both the hub and the app and demonstrating the ability to open and close the valve from both the hub and the app.

First, the moisture sensor needed to be tested. For this, the sensor and its circuit were connected directly to a computer and the computer's terminal was read. The circuit and code was adjusted until the terminal correctly read "dry" when no water was present and "wet" when water was present.

Next, the solar charger was tested. Unfortunately, this was not as simple as taking out the battery and checking for a proper open circuit voltage (approximately 4.95 V). The LT3652 identified a battery fault when the battery was not attached, causing it to withhold power such that only 0.36 V ran across the battery port. Instead, the status registers of the LT3652 chip were read when the battery was plugged in and in good sunlight. When this was done, the CHRG register was high and the FAULT register was low, signifying normal charging.

To test the motor board, the relay switches first needed to be tested. This was done by simply having them alternate from low to high periodically and reading associated pins to make sure they were switching when directed. Next, the linear actuator was added. Originally, problems were experienced when the linear actuator was applied. Part of this was a result of an AC signal coming from the motor. This was omitted by connecting a capacitor between the linear actuator's leads. However, the system still experienced problems. The small capacitor connected between the leads was not rated for the voltage spike that occurred when the linear actuator switched polarities. This voltage spike was a result of a rapid change in current (V = L*dI/dt). By connecting 150 V capacitors between each lead of the linear actuator and ground, this problem was reduced.

However, the valve board still had problems. Unfortunately, the system only worked when one battery powered the relay switches and linear actuator, while the other powered the ESP8266. To account for this discrepancy, two identical boards were connected by their ground planes, GPIO13 pins, and GPIO14 pins. This solved the problem and allowed the entire system to work as expected.

The final key to the project that needed to be tested was range. The team determined that 200 m of range was necessary for this project to sufficiently account for the size of most farms that use flood irrigation in the southwest United States. Using the PCB antenna built into the SparkFun ThingDev, however, only 150 m of range were obtained. As a result, three 9 dBi antennas were attached to the central hub's router and a 2 dBi external antenna was applied to both the pylon and the valve. As a result, the signal's range rose to 250 m, which exceed the minimum.

Description of how testing demonstrated that the overall system meets the design requirements

The testing done on this system proves the effectiveness of the product in all the design constraints originally set out by the team. Detailed system requirements are meticulously outlined in the section titled “Detailed System Requirements.” The first requirement is a central hub that establishes and maintains two-way communication with the motor, valve, and app and directing those three subsystems appropriately. The tests done demonstrated the hub’s effectiveness in accomplishing these tasks. Additionally, the hub is capable of communicating over a distance of 250 m, which exceeded original expectations and requirements. Furthermore, the pylon meets its design requirements, as shown through testing of its solar charging capabilities, ability to sense water, connect to an MQTT server, and communicate with the central hub. The valve also met its design requirements by opening and closing the linear actuator remotely. A metal valve was not raised or lowered (as originally planned) since doing so in a way that would be applicable to flood irrigation in the real world would exceed the budget constraints placed on this project. Finally, the app is able to make commands to the hub, which can then be relayed to the pylon or valve. Additionally, the status of all pylons and valves can be checked from the app.

User Manual/Installation Instructions

Setup/Installation

By design, the product is very easy to set up. Installation follows three steps: bringing the hub and server online, syncing devices, and placing devices in the field.

First, the hub must be active and the web app up and running. To ensure this, open the Raspberry Pi's Terminal window after booting up to launch the application.

Second, the devices must be synced. It's important to note that only one device may be synced at a time, as all devices use the same wireless pathways for syncing. Both pylon and valve devices follow the same syncing procedure. First, make sure that both the hub and the devices to be synced are on. The hub should be displaying the web app main page and the light on the device should be blinking very slowly (two seconds on two seconds off). Bring the device near the hub (within ten feet); proximity makes the sync work faster. Press the button on the device and hold down for about a second. The led will now start blinking faster (about one second on, one second off). This indicates the the device is initiating a wireless access point to receive information from the hub.

On a computer or phone, connect to the router's Wi-Fi. Then, access the web app by going to the URL http://192.168.0.100:5000. The web app main page should come up - select the sync command. The user will be prompted to input a name for the device and specify whether it is a pylon or a valve; we recommend naming the device something easy to remember and easy to associate with its position or role in the field. Press "enter"; the web app will remain motionless for a few minutes while the sync completes. When it is done, the device LED will blink rapidly before remaining lit in the “on” position. This indicates that the sync was successful and the device was successfully connected and is awaiting instructions. On the web app, the newly named device will show up under the list of devices (depending on whether it’s a pylon or a valve). The device will periodically go dark as it goes into sleep mode after receiving instructions from the hub. Repeat this step for every device you wish to bring onto the system.

Finally, the devices must be set in the field.

For a pylon: unscrew the nuts binding the PVC tube and solar configuration to the metal stake. Disconnect the moisture sensor from the casing by detaching the small white connector. With the metal stake completely detached from the casing, drive it into the soil wherever a position should be monitored; a hammer may be useful here. After loosening the nuts on the moisture sensor carriage, slide it up or down the shaft of the metal stake until the tips of the moisture sensor are about one centimeter off the ground. Holding the casing close to the stake, re-attach the white connector to the moisture sensor; tighten the nuts on the sensor casing. Finally, fit the screws of the casing through the holes on the stake and use the nuts to fasten the casing securely to the stake. With the pylon synced and in the ground, it is ready for operation.

For a valve: Adjust the solar panel so that it’s facing straight up and adjust the antenna so that it’s perpendicular to the ground. This part of installation is very modular and depends on the current floodgate configuration. However, as long as the plate of the floodgate can be securely fastened to the end of the linear actuator (by use of a screw or similar fastener) and the plate is not heavier than 100 lbs, the system will function adequately. Securely fasten the base of the linear actuator (the thicker part) to the frame of the floodgate. Be sure that the valve circuit boards are protected from the elements but all connections are secure.

At this point, all components are set and functional. Open the web app on a phone or computer; all the synced valves and pylons should be on their respective pages. To set the devices to awake mode (for an irrigation cycle), click on the “Sleep/Awake” button and click “Awake” in the next page. Or, to put devices in stasis, select Sleep from the same page.

To change the name of a device, navigate to it in its respective list and click the Manage button. Click “ID Reset”; the user will be given an input to enter the new name of the device; press “enter” after typing in the desired name. The device will now show up as “Syncing” in its list. After a minute or so, the device will be appropriately connected and will show up as its own name.

To pull a device off the network (reset), navigate to it in its respective list, click the Manage button, and click “Reset”. The device can now be synced to the same hub or another hub following the sync instructions as given above.

To open/close a valve, navigate to it in the the Valves list, and click “Manage”. If the valve is open, the user will have the option to close it and vice versa.

To assign a valve to a pylon, navigate to the pylon in the Pylons list and click “Manage.” Click “Assign Valve,” and choose the desired valve from the dropdown menu. The valve name should show up in the Pylons list in line with name of the pylon it’s associated with.

How the user can tell if the product is working

Regarding the devices: the LED indicator should flash quickly, then stay on for anywhere between ten seconds and a minute then going dark to repeat. This indicates that the device is waking up, communicating with the hub, then going to sleep.

Regarding the hub: when looking at the webapp, the status of a pylon should change within one minute of it transitioning between wet and dry. The correct sleep and awake settings should also be indicated in the devices lists. The web app should not be hanging on “Syncing” indicators in place of the device’s name. Similarly, none of the valves should get stuck with an “opening” or “closing” indicator for longer than three minutes. In general, the web app should be responsive to whatever change the user makes within a matter of minutes.

Of course, the user should see the actuator move after sending a close or open command to the valve (matter of minutes).

How the user can troubleshoot the product

Likely errors/symptoms:

- Web app giving “Internal Server Error”
- Valve or Pylon not updating in webapp
- Device LED either staying on indefinitely
- Device LED never turning on
- Actuator not moving
- Device getting hung up on “Syncing”
- Valve getting hung up on “opening” or “closing”

Regarding issues having to do with devices: check all connections and reboot by disconnecting and reconnecting the battery.

Regarding issues with the hub: complete reboot.

The team does not anticipate the user having the skills or equipment to replace surface mount parts or interact with the Raspberry Pi and edit files. If a simple reboot is not sufficient to remove errors, the problem is likely beyond the abilities of the user to fix. Please contact customer service or send the product in to be repaired.

To-Market Design Changes

As a result of budget and time limitations, the functional prototype created differs from the product that would be introduced to the market. The greatest change that would need to be made is the valve subsystem. In the prototype, a 12 V linear actuator with the capability of pushing 1500 N and pulling 1000 N was used. The device did not open or close a valve, but instead moved up or down based on whether or not water was sensed by the pylon. In real life, a motor would need to either be retrofitted to the preexisting valve, or a brand new valve would need to be installed. In either case, the motor would need to be much larger and have greater capabilities than the one used for the prototype.

Generally, fields that utilize flood irrigation require the farmer to turn a wheel that closes the valve, stopping the water from flowing. Rather than having a new valve be installed entirely (this would be very expensive), the best way to prepare this system for commercial use would first be to find a motor capable of turning the wheel. Once an acceptable motor has been determined, the system should be fitted with a method to decide when the valve is fully open and fully closed. This would tell the motor when to stop turning the wheel. Finally, hardware would need to be designed to attach the motor to the pipe water exits through.

While the motor and valve subsystem would need to be upgraded in order to be brought to the market, the other aspects of the prototypes are market-ready. The pylon is robust enough to withstand high winds, rains, and should indefinitely run as a result of a solar charging system that is capable of charging the battery as well as a battery that will take a long time to drain as a result of the low level of power the system requires to run effectively. Similarly, the app is ready for download by customers and would effectively provide a way to remotely control the system in a cosmetically pleasing and simple manner. Finally, the central hub is ready for installation into the home of the irrigation in question.

Conclusion

Overall, this project was successful in meeting its design goals. All constraints were met and, aside from some valve fitting to pre-existing floodgates, the product could serve as a prototype for a commercially-used system in the future.

The wireless network system offers a modular solution that can scaled up or down depending on the needs of the farmer. The pylons are robust in physical and software build to handle the New Mexico elements and requirements for performing as designed in the field. The irrigation system also removes the strenuous physical effort required to reach and open a 300 lb floodgate with a wireless linear actuator control system. To cap off the system, the pylons and valves can be monitored and controlled from the comfort of your home. As previously stated, flood irrigation is a daunting process that requires the prolonged attention of the farmer in order to properly care for crops without exposing him to property damage or the wasting resources. Rather than spending 24 to 36 hours every two to three weeks grounded and enduring the demanding irrigation process, farmers will now be able to rely on a reliable, effective, and autonomous irrigation system that takes the human out of the process.

Appendices

A: Circuit Diagrams

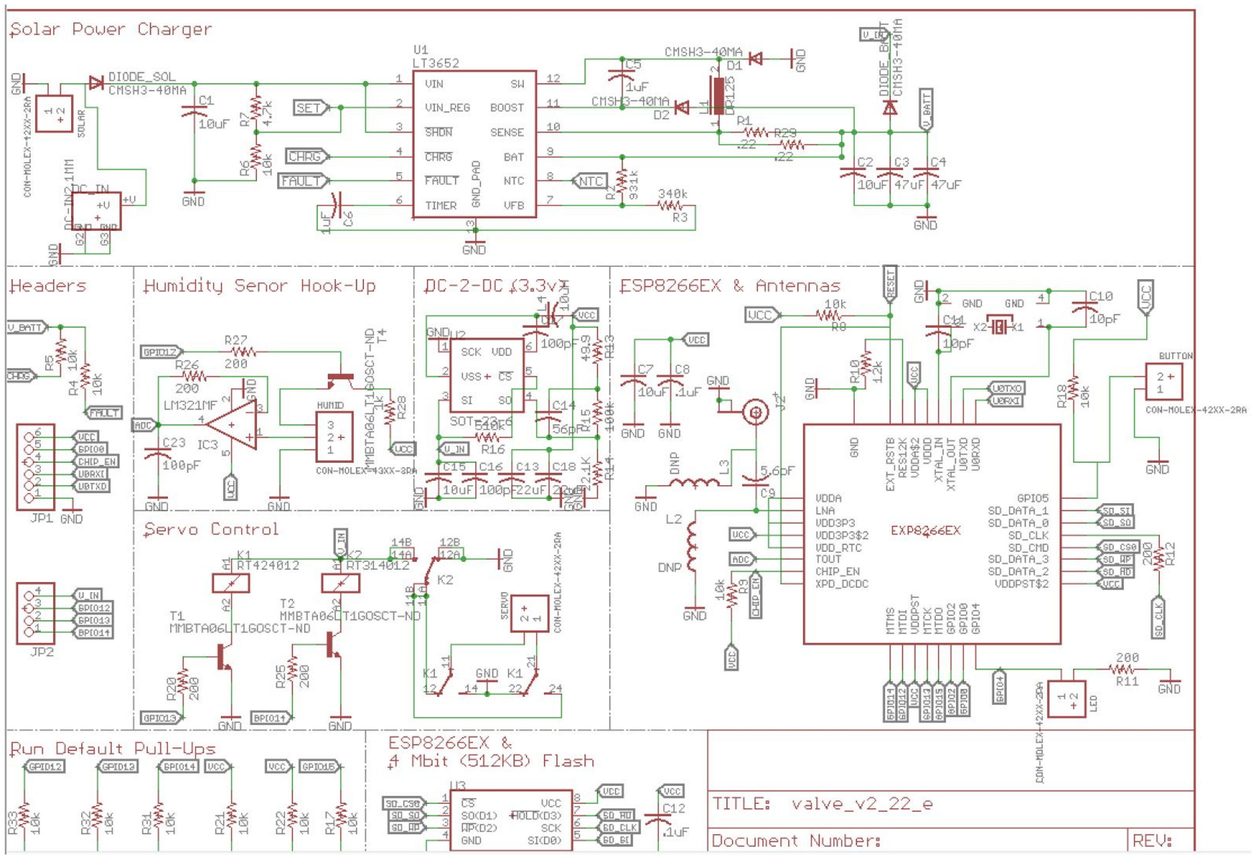

Figure 13. Pylon schematic.

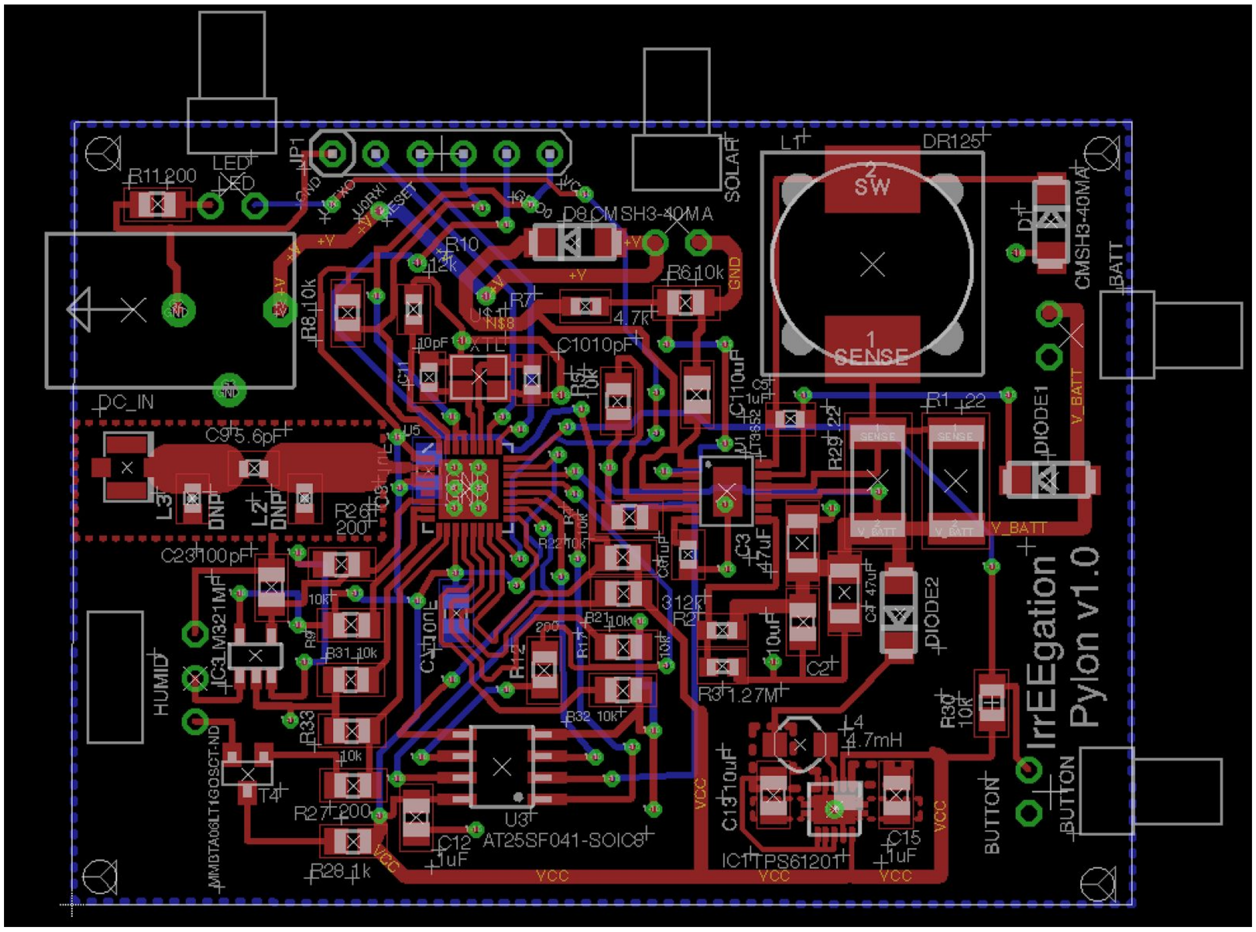

Figure 14. Pylon board.

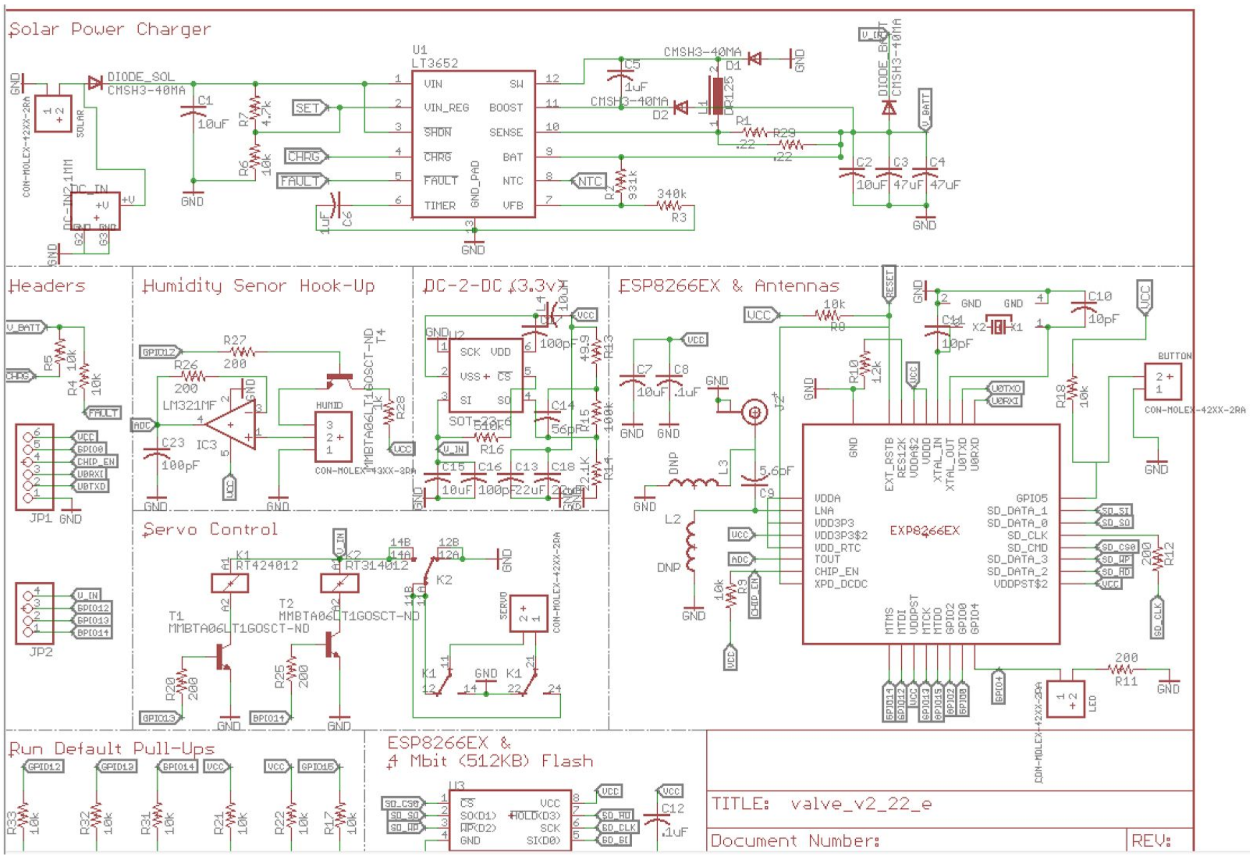

Figure 15. Valve schematic.

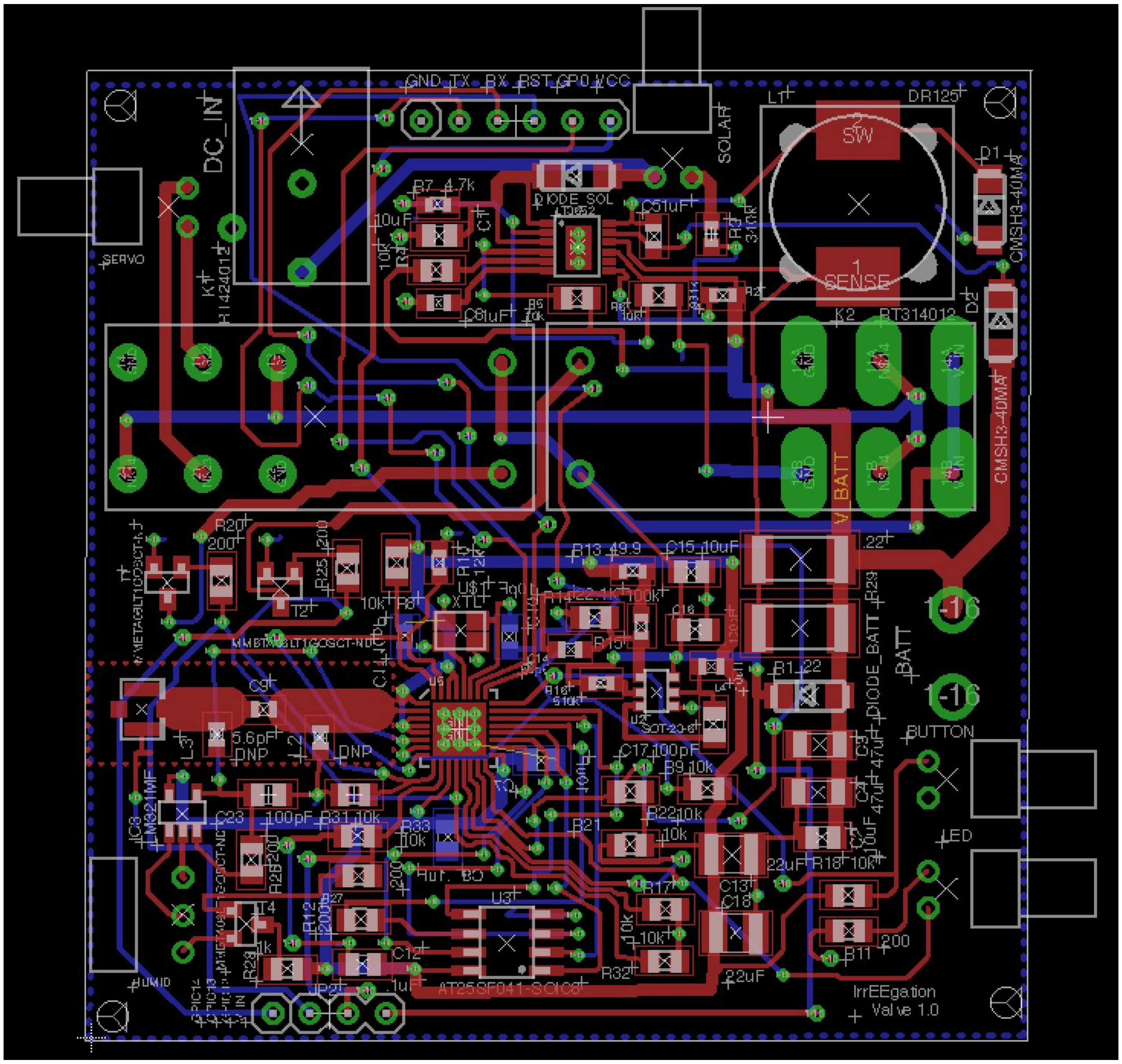

Figure 6. Valve board.

B: Pylon Code (as seen in Arduino IDE using C syntax):

pylon_10_final.ino

```
//This is the code running on every pylon device. It proceeds as follows:
//-check EEPROM for identifiers, wait for Sync if identifiers not present
//-if Sync performed, save identifiers to EEPROM
```

```
//-connect to WiFi and establish MQTT communications - save new identity if need be
//-read moisture sensor and publish wet/dry status to hub, wait for commands
//-if commanded, erase EEPROM or specifically delete identity
//-sleep for duration of time determined by hub command
//Comments are found throughout the code, demarcating and describing main functions.
//Serial prints are included also - they were essential for debugging and
//offer additional insight as to the form of the program.
//Following the bulk of the code are our callback and access point functions.

//=============Libraries=========
#include <EEPROM.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
//===============================

//===================GLobal Declarations==============
//Identity Declarations
char identity[32];
int identityLength;
boolean identityReset;
boolean identityResetSet = false;
boolean identityDecided = false;
boolean identityCorrectLength = false;

//Sleep/Awake Declarations
boolean wake = true;
boolean wakeSet = false;

//Reset Declarations
boolean reset;
boolean resetSet = false;

//IO Declarations
const int SWITCH = 5;
const int LED = 4;

//Function to Set Up Callback
void callback(const MQTT::Publish& pub);

//Function to set up access point
void setupWiFi();

//==============Setup==================
void setup() {
 Serial.begin(115200);
 EEPROM.begin(512);
 pinMode(SWITCH, INPUT);
 pinMode(LED, OUTPUT);
 pinMode(A0, INPUT);
 pinMode(12, OUTPUT);
 digitalWrite(12, HIGH);
}
//======================================

//========================Code Officially Begins========================
void loop() {
 //------------Main Declarations-----------------------
 //Moisture Measurement Declarations
 int waterPin = A0;
 int waterValue = 0;
 int waterThreshold = 50;
```

```
String waterLevel;

//Character Arrays to Store Identifiers
char ssid[32];
char password[32];
char ip[32];

//Sleep Timer Declarations
int sleepMultiplier = 3;
int sleepTime;

//Publish and Subscribe Declarations
String pylon = "pylon";
String command = "commandpylon";

//Variables for Looping
char i;
int g;

//Tracking Addresses while reading and writing
int addrRead = 0;
int addrWrite = 0;
int firstAddress = 0;

//Checking if EEPROM full or not
char eepromCheck;
char identityCheck;
boolean infoIn;

//Integers recording lengths of Identifiers
int ssidLength;
int passLength;
int ipLength;
int stringLength;

//LED Control
int ledState = LOW;

//Timer Variables
unsigned long currentMillis;
unsigned long previousMillis;
long waitLightInterval = 2000;
long syncTimeout = 240000;
long wifiTimeout = 10000;

//--------------------------------------------------------



//------------------------ Check EEPROM for stored data ------------
eepromCheck = EEPROM.read(firstAddress);
identityCheck = EEPROM.read(96); //this is the first address corresponding to the identity of the pylon
//-------------------------------------------------------------------
delay(1000);



//------------------If EEPROM Empty, no identifiers are stored - initiate Sync------
////if first address null, there are no identifiers present
if (eepromCheck == '\0')
{
```

```
Serial.println("No info, waiting for sync");
//Declarations for Loading Identifiers
boolean passwordIn = false; //true if pylon has received suitable password
boolean ssidIn = false; //true if pylon has received suitable ssid
boolean ipIn = false; //true if pylon has received suitable ip address
boolean identityIn = false; //true if pylon has received suitable identity
boolean entryLength; //only true if submitted message is under 32 characters
char charCatch;
int j;



previousMillis = millis();
//Standby until button hit
while (digitalRead(SWITCH) == HIGH) {
  currentMillis = millis();
  if (currentMillis - previousMillis >= waitLightInterval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
    // set the LED with the ledState of the variable:
    digitalWrite(LED, ledState);
  }
  //Serial.println("waiting for Sync button");
  delay(500);
}



//Start Up Access Point
WiFiServer server(80);
setupWiFi();
server.begin();
Serial.println("button hit");
previousMillis = millis();




//Listen for identifiers from hub
while (!passwordIn || !ssidIn || !ipIn || !identityIn) //all four identifiers must be received
{
  entryLength = true; //assume incoming entry is correct length


  // Check if a client has connected
  WiFiClient client = server.available();
  if (client) {
    Serial.println("client connected");
  }


  // Read the first line of the request
  String req = client.readStringUntil('\r'); //req holds the submission from the hub
```

```
//Remove unnecessary components of submission (chars added that are unneeded)
int trim = req.length() - 9;
req.remove(trim);
Serial.println(req);
client.flush();

//if ssid sent, record as ssid
if (req.indexOf("ssid/") != -1) {
  req.remove(0, 10);
  req.toCharArray(ssid, 32);
  ssidLength = req.length();
  ssidIn = true;
  if (ssidLength > 32) {
    Serial.println("too long");
    entryLength = false;
    ssidIn = false;
  }
  Serial.print("ssidLength is:");
  Serial.println(ssidLength);

  Serial.print("ssid in: ");
  Serial.print(ssid);
}

//if password sent, record as password
else if (req.indexOf("password/") != -1) {
  req.remove(0, 14);
  req.toCharArray(password, 32);
  passLength = req.length();
  passwordIn = true;
  if (passLength > 32) {
    entryLength = false;
    passwordIn = false;
  }
  Serial.print("passLength is:");
  Serial.println(passLength);
  Serial.print("Password in: ");
  Serial.print(password);
}

//if ip address sent, record as ip address
else if (req.indexOf("ip/") != -1) {
  req.remove(0, 8);
  req.toCharArray(ip, 32);
  ipLength = req.length();
  ipIn = true;
  if (ipLength > 32) {
    entryLength = false;
    ipIn = false;
  }
  Serial.print("ipLength is:");
  Serial.println(ipLength);
  Serial.print("IP in: ");
  Serial.print(ip);
}

//if identity sent, record as identity
else if (req.indexOf("identity/") != -1) {
  req.remove(0, 14);
  req.toCharArray(identity, 32);
  identityLength = req.length();
```

```
    identityIn = true;
    if (identityLength > 32) {
      entryLength = false;
      identityIn = false;
    }
    Serial.print("identityLength is:");
    Serial.println(identityLength);
    Serial.print("Identity in: ");
    Serial.print(identity);
  }
  //---------------------------------------------------------------



  //alert the hub if the sent string is too many characters (over 32)
  if (!entryLength) {
    req = "too long";
  }


  //Compose Response
  String s = "HTTP/1.1 200 OK\r\n";
  s += "Content-Type: text/html\r\n\r\n";
  s += "<!DOCTYPE HTML>\r\n<html>\r\n";
  s += "String is";
  s += "<br>";
  s += req;

  // Send the response to the client
  client.print(s);
  delay(1);

  //Timer
  currentMillis = millis();

  if (ledState == LOW) {
    ledState = HIGH;
    //Serial.println("now high");
  } else {
    ledState = LOW;
    //Serial.println("now low");
  }
  digitalWrite(LED, ledState);


  //Pull out of loop and restart device if nobody there
  if (currentMillis - previousMillis >= syncTimeout) {
    previousMillis = currentMillis;
    Serial.println("Going to sleep - nobody talking");
    goto initiateSleep;
  }

}
Serial.println("Identifiers found");
infoIn = true; //identifiers in place; proceed with MQTT
delay(5000);
digitalWrite(LED, LOW);
WiFi.mode(WIFI_OFF);

//-------------------------Record All Credentials-----------------------
//Start with SSID Addresses
```

```
Serial.print(" SSID length is  ");
Serial.println(ssidLength);
delay(500);
addrWrite = 0;
int ssidStart = addrWrite;
Serial.print("starting at address ");
Serial.println(ssidStart);
delay(500);
int ssidEnd = addrWrite + ssidLength;
Serial.print("ending at address ");
Serial.println(addrWrite);


//write ssid to ssid addresses
for (addrWrite; addrWrite <= ssidEnd; addrWrite++) {
  EEPROM.write(addrWrite, ssid[addrWrite - ssidStart]);
  EEPROM.commit();
  Serial.print("Address ");
  Serial.print(addrWrite);
  Serial.print(" has :");
  Serial.println(ssid[addrWrite - ssidStart]);
}

//prepare to write password - calculate length, startpoint, endpoint
Serial.print(" Password length is  ");
Serial.println(passLength);
delay(500);
addrWrite = 32;
int passStart = addrWrite;
Serial.print("starting at address ");
Serial.println(addrWrite);
delay(500);
int passEnd = addrWrite + passLength;
Serial.print("ending at address ");
Serial.println(passEnd);

//write password to password addresses
for (addrWrite; addrWrite <= passEnd; addrWrite++) {
  EEPROM.write(addrWrite, password[addrWrite - passStart]);
  EEPROM.commit();
  Serial.print("Address ");
  Serial.print(addrWrite);
  Serial.print(" has :");
  Serial.println(password[addrWrite - passStart]);
}

//prepare for EEPROM writing - calculate startpoint, endpoint, and length
Serial.print(" ip length is  ");
Serial.println(ipLength);
delay(500);
addrWrite = 64;
int ipStart = addrWrite;
Serial.print("starting at address ");
Serial.println(addrWrite);
delay(500);
int ipEnd = addrWrite + ipLength;
Serial.print("ending at address ");
Serial.println(ipEnd);

//loop through and write to ip address addresses
for (addrWrite; addrWrite <= ipEnd; addrWrite++) {
   EEPROM.write(addrWrite, ip[addrWrite - ipStart]);
   EEPROM.commit();
   Serial.print("Address ");
   Serial.print(addrWrite);
   Serial.print(" has :");
   Serial.println(ip[addrWrite - ipStart]);
  }

  //figure out addresses to store identity
  Serial.print(" identity length is  ");
  Serial.println(identityLength);
  delay(500);
  addrWrite = 96;
  int identityStart = addrWrite;
  Serial.print("starting at address ");
  Serial.println(addrWrite);
  delay(500);
  int identityEnd = addrWrite + identityLength;
  Serial.print("ending at address ");
  Serial.println(identityEnd);

  //store identity
  for (addrWrite; addrWrite <= identityEnd; addrWrite++) {
   EEPROM.write(addrWrite, identity[addrWrite - identityStart]);
   EEPROM.commit();
   Serial.print("Address ");
   Serial.print(addrWrite);
   Serial.print(" has :");
   Serial.println(identity[addrWrite - identityStart]);

  }
 }
  //------------end of access point; following is if EEPROM loaded----
  else if (eepromCheck != '\0') {
   Serial.print("Info Found, Starting Read Loop");
   for (int k = 0; k <= 3; k++) {

    //Assign Correctly to SSID, Password, or IP Address
    switch (k) {

     case 0://SSID

      stringLength = 0;
      //Read Loop
      addrRead = 0; //ssid starts at address 0
      do  {
       Serial.println("Reading SSID");
       i = EEPROM.read(addrRead);
       ssid[stringLength] = EEPROM.read(addrRead);
       Serial.println(i);
       addrRead ++;
       stringLength ++;
      } while (i != '\0' && stringLength <= 32);
      delay(1000);
      Serial.print("SSID is ");
      Serial.println(ssid);
      ssidLength = stringLength - 1;
      break;

     case 1://Password
  stringLength = 0;
  addrRead = 32;
  //Read
  do  {
    Serial.println("Reading Password");
    i = EEPROM.read(addrRead);
    password[stringLength] = EEPROM.read(addrRead);
    Serial.println(i);
    //delay(1000);
    addrRead ++;
    stringLength ++;
  } while (i != '\0' && stringLength <= 32);
  delay(1000);
  Serial.print("Password is ");
  Serial.println(password);
  passLength = stringLength - 1;
  break;

case 2://IP Address

  stringLength = 0;
  addrRead = 64;
  //Read
  do  {
    Serial.println("Reading IP Address");
    i = EEPROM.read(addrRead);
    ip[stringLength] = EEPROM.read(addrRead);
    Serial.println(i);
    //delay(1000);
    addrRead ++;
    stringLength ++;
  } while (i != '\0' && stringLength <= 32);
  delay(1000);
  Serial.print("IP is ");
  Serial.println(ip);
  ipLength = stringLength - 1;
  break;


case 3://Identity

  //if there is no identity (having been erased from an earlier cycle) then don't bother reading it
  if (identityCheck == '\0') {
    break;
  }
  stringLength = 0;
  addrRead = 96;
  //Read Loop
  do  {
    Serial.println("Reading Identity");
    i = EEPROM.read(addrRead);
    identity[stringLength] = EEPROM.read(addrRead);
    Serial.println(i);
    //delay(1000);
    addrRead ++;
    stringLength ++;
  } while (i != '\0' && stringLength <= 32);
  delay(1000);
  Serial.print("Identity is ");
  Serial.println(identity);
  identityLength = stringLength - 1;
```

```
        break;

      }
    }
    infoIn = true; //identifiers in place, proceed with MQTT
  }
  //-----------------------------------------------------------------------------------



  //-------------------------if EEPROM has info - > connect---------------------------
  if (infoIn) {   //first, check if identifiers are in place
    Serial.println("made it to connect part");
    WiFiClient wf_client; // instantiate wifi client
    PubSubClient client(wf_client, ip); // pass to pubsub
    Serial.println();
    Serial.println();
    Serial.println(F("Modified pubsub client basic code using modified pubsub software"));

    // Connect to WIFI of Router.
    Serial.println(); Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    //Set Time Loop
    previousMillis = millis();

    client.set_callback(callback);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
      currentMillis = millis();
      delay(500);
      Serial.print(".");
      if (ledState == LOW) {
        ledState = HIGH;
      } else {
        ledState = LOW;
      }
      digitalWrite(LED, ledState);

      //Pull out of loop and restart device if WiFi not found
      if (currentMillis - previousMillis >= wifiTimeout) {
        previousMillis = currentMillis;
        Serial.println("Going to sleep - no wifi");
        goto initiateSleep;
      }

    }
    Serial.println();
    Serial.println("WiFi connected");
    Serial.println("IP address: "); Serial.println(WiFi.localIP());
    //-------------------------------------------------------------------------------



    //----------------------------check moisture sensor----------------------------
    waterValue = analogRead(waterPin);
    if (waterValue > waterThreshold) {
      waterLevel = "wet";
    }
    else {
```

```
  waterLevel = "dry";
}

//------------------------------------------------------------------------------



//---------------------------------initiating mqtt cycle-------------------------
Serial.println("Starting Loop");
if (WiFi.status() == WL_CONNECTED) {
  if (!client.connected()) {
    if (client.connect("mydevice", "Pylon code", 0 , false, "pylon disconnected")) {
      digitalWrite(LED, HIGH);
      Serial.println("MQTT Connected");
      Serial.println("connected and stuff");


      //store identity in EEPROM
      if (identityCheck == '\0' || eepromCheck == '\0') {
        Serial.println("no identity found");

        //identity sync, if identity not already present
        if (eepromCheck != '\0') {
          Serial.println("identity only thing missing, waiting for hub command");
          identityDecided = false;
          client.subscribe("commandIdentity");
          client.publish("identity", "ready for identity");
          do {
            client.loop();
            delay(2000);
            Serial.print("identity decided is : ");
            Serial.println(identityDecided);
            if (!identityCorrectLength) {
              client.publish("identity", "too long");
            }
          if (!client.connected()) {
            Serial.println("Lost MQTT, jumping out of loop, sleeping and trying again");
            identityDecided = true;

          }
          } while (!identityDecided );
          Serial.print("identity is : ");
          Serial.println(identity);
          client.publish("identity received", "identity"); //added for 2_2
          client.unsubscribe("commandIdentity");
        }


        //figure out addresses to store identity
        Serial.print(" identity length is  ");
        Serial.println(identityLength);
        delay(500);
        addrWrite = 96;
        int identityStart = addrWrite;
        Serial.print("starting at address ");
        Serial.println(addrWrite);
        delay(500);
        int identityEnd = addrWrite + identityLength;
        Serial.print("ending at address ");
        Serial.println(identityEnd);
  //store identity
  for (addrWrite; addrWrite <= identityEnd; addrWrite++) {
    EEPROM.write(addrWrite, identity[addrWrite - identityStart]);
    EEPROM.commit();
    Serial.print("Address ");
    Serial.print(addrWrite);
    Serial.print(" has :");
    Serial.println(identity[addrWrite - identityStart]);
  }
}


//Publish Waterlevel and Subscribe to Hub
pylon.concat(identity); //create unique topic string for pylon to speak to hub
command.concat(identity); // create unique topic string for hub to speak to pylon
client.publish(pylon, waterLevel); //publish the waterlevel to the hub
Serial.println("just published, now listening");
client.subscribe(command); //listen for commands from the hub
//-------------------------------------------------------------------

if (client.connected()) { //poo
  do {
    client.loop();
    delay(2000);
    Serial.println("waiting for responses");
    Serial.print("Wakeset is : ");
    Serial.println(wakeSet);
    Serial.print("Resetset is : ");
    Serial.println(resetSet);
    Serial.print("identityResetSet is : ");
    Serial.println(identityResetSet);
    Serial.print("water at: ");
    Serial.println(analogRead(waterPin));
    if (!client.connected()) {
      Serial.println("Lost MQTT, jumping out of loop, sleeping and trying again");
      reset = false;
      identityReset = false;
      wake = true;
      wakeSet = true;
      resetSet = true;
      identityResetSet = true;

    }
  } while (!wakeSet || !resetSet || !identityResetSet);

  client.loop();
  Serial.print("Wake is : ");
  Serial.print(wake);
  Serial.print("Reset is : ");
  Serial.println(reset);
  Serial.print("Identity Reset is : ");
  Serial.println(identityReset);
  delay(1000);
  Serial.println("about to decide");

  //Calculate Sleep Time from Sleep Commands
  if (wake) {
    Serial.println("test");
    sleepMultiplier = 3;
  }
  else {
```

```
            sleepMultiplier = 20;
           }
           //------------------------------------------------------------------------

          //Note: the EEPROM write code (currently associated with the Access Point code) originally
          //belonged here, the idea being that the device would only save Identifiers after a first,
          //successful sync. However, our designed boards didn't always connect reliably - sometimes
          //they would successfully sync and receive identifiers, but the MQTT or WiFi connections
          //would fall through and the identifiers would be unsaved. So instead, the EEPROM saving code
          //is placed right after the identifiers are received in the sync.

           //-----------------------------Reset and Identity Reset--------------------------
           //---------Reset
           if (reset) {
            for (int i = 0; i < 128; i++)
              EEPROM.write(i, '\0');
            EEPROM.commit();
            client.publish(pylon, "reset");
           }
           else if (identityReset) { //-------Identity Reset
            for (int i = 96; i < 128; i++)
              EEPROM.write(i, '\0');
            EEPROM.commit();
            client.publish(pylon, "reset");
           }
           Serial.println("MQTT Disconnected");
           delay(500);
          }
         }
        }
        Serial.println("No Connection");
        delay(500);
       }
       Serial.println("made it to end");
      }
      //Initiate Deep Sleep - End of Cycle
    initiateSleep://goto takes program here if any connection cycles took too long
      Serial.println("going to sleep now");
      sleepTime = sleepMultiplier * 1000000;
      ESP.deepSleep(sleepTime, WAKE_NO_RFCAL);

     }



     //192.168.4.1 - IP address of Access Point (for debug purposes)




     //===================Function to set up access point ===============
     void setupWiFi() {
      WiFi.mode(WIFI_AP);

      uint8_t mac[WL_MAC_ADDR_LENGTH];
      String AP_NameString = "irrigateSync";

      char AP_NameChar[AP_NameString.length() + 1];
      memset(AP_NameChar, 0, AP_NameString.length() + 1);

      for (int i = 0; i < AP_NameString.length(); i++)
    AP_NameChar[i] = AP_NameString.charAt(i);

  WiFi.softAP(AP_NameChar);
}
//==================================================================

//========================Callback Function========================
void callback(const MQTT::Publish & pub) {
  // handle message arrived
  Serial.print("Message arrived [");
  Serial.print(pub.topic());
  Serial.print("] ");

  Serial.println(pub.payload_string());
  delay(2000);
  Serial.println();
  //check first three characters of message - having to do with sleep cycle
  if (pub.payload_string().substring(0, 3) == "wak") { //wake command
    wake = true;
    wakeSet = true;
    Serial.println("wake mode");
  }
  else if (pub.payload_string().substring(0, 3) == "slp") { //sleep command
    wake = false;
    wakeSet = true;
    Serial.println("sleep mode");
  }
  //check middle three characters of string - having to do with full reset
  if (pub.payload_string().substring(3, 6) == "rst") {//full reset command
    reset = true;
    resetSet = true;
    Serial.println("Reset");
  }
  else if (pub.payload_string().substring(3, 6) == "sta") {//no reset
    reset = false;
    resetSet = true;
    Serial.println("noReset");
  }
  //check last three characters of message - having to do with identity reset
  if (pub.payload_string().substring(6, 9) == "rst") {//identity reset command
    identityReset = true;
    identityResetSet = true;
    Serial.println("identityReset");
  }
  else if (pub.payload_string().substring(6, 9) == "sta") {//maintain identity command
    identityReset = false;
    identityResetSet = true;
    Serial.println("identitySame");
  }
  else {//only other message that could have been sent would be an identity during an identity sync
    pub.payload_string().toCharArray(identity, 32);
    identityLength = pub.payload_string().length();
    if (identityLength > 32) {
      identityDecided = false;
      identityCorrectLength = false;
    }
    else {
      identityDecided = true;
      identityCorrectLength = true;
    }
```

```
      Serial.println("identity is reset");
    }
  }

  //================================================
```

C: Valve Code (as seen in the Arduino IDE using C syntax):

valve_8_final.ino

```
//This is the code running on every valve device. It proceeds as follows:
//-check EEPROM for identifiers, wait for Sync if identifiers not present
//-if Sync performed, save identifiers to EEPROM
//-connect to WiFi and establish MQTT communications - save new identity if need be
//-publish an active "ping" to hub, wait for commands
//-if commanded, open or close valve
//-if commanded, erase EEPROM or specifically delete identity
//-sleep for duration of time determined by hub command
//Comments are found throughout the code, demaracating and describing main functions.
//Serial prints are included also - they were essential for debugging and
//offer additional insight as to the form of the program.
//Following the bulk of the code are our callback and access point functions.

//=============Libraries=========
#include <EEPROM.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
//===============================

//==================GLobal Declarations==============
//Declaration for WiFi Access Identity
//const char WiFiAPPSK[] = "sparkfun";

//Identity Declarations
char identity[32];
int identityLength;
boolean identityReset;
boolean identityResetSet = false;
boolean identityDecided = false;
boolean identityCorrectLength = false;

//Sleep/Awake Declarations
boolean wake = true;
boolean wakeSet = false;

//Reset Declarations
boolean reset;
boolean resetSet = false;

//Command Valve
boolean command_close;
boolean valve_set = false;

//IO Declarations
const int SWITCH = 5;
const int LED = 4;

//Function to Set Up Callback
void callback(const MQTT::Publish& pub);

//Function to set up access point
void setupWiFi();

//==============Setup==================
void setup() {
 Serial.begin(115200);
 EEPROM.begin(512);
 pinMode(SWITCH, INPUT);
 pinMode(LED, OUTPUT);
 pinMode(14, OUTPUT); //GPIO14 Swithces actuator on
 pinMode(13, OUTPUT);
}
//======================================

//========================Code Officially Begins========================
void loop() {
 //------------Main Declarations-----------------------
 //Valve Identifiers
 char valve_check;
 boolean valve_closed;
 String valve_status;

 //Character Arrays to Store Identifiers
 char ssid[32];
 char password[32];
 char ip[32];

 //Sleep Timer Declarations
 int sleepMultiplier = 3;
 int sleepTime;

 //Publish and Subscribe Declarations
 String valve = "valve";
 String command = "commandvalve";

 //Variables for Looping
 char i;
 int g;

 //Tracking Addresses while reading and writing
 int addrRead = 0;
 int addrWrite = 0;
 int firstAddress = 0;

 //Checking if EEPROM full or not
 char eepromCheck;
 char identityCheck;
 boolean infoIn;

 //Integers recording lengths of Identifiers
 int ssidLength;
 int passLength;
 int ipLength;
 int stringLength;

 //LED Control
 int ledState = LOW;

 //Timer Variables
 unsigned long currentMillis;
 unsigned long previousMillis;
 long waitLightInterval = 2000;
 long syncTimeout = 240000;
 long wifiTimeout = 10000;

 //-------------------------------------------------------



 //------------------------ Check EEPROM for stored data ------------
 eepromCheck = EEPROM.read(firstAddress);
 identityCheck = EEPROM.read(96); //this is the first address corresponding to the identity of the valve
 //-------------------------------------------------------------------
 delay(1000);



 //------------------If EEPROM Empty, no identifiers are stored - initiate Sync------
 ////if first address null, there are no identifiers present
 if (eepromCheck == '\0')
 {
   Serial.println("No info, waiting for sync");
   //Declarations for Loading Identifiers
   boolean passwordIn = false; //true if valve has received suitable password
   boolean ssidIn = false; //true if valve has received suitable ssid
   boolean ipIn = false; //true if valve has received suitable ip address
   boolean identityIn = false; //true if valve has received suitable identity
   boolean entryLength; //only true if submitted message is under 32 characters
   char charCatch;
   int j;



   previousMillis = millis();
   while (digitalRead(SWITCH) == HIGH) {
     currentMillis = millis();
     if (currentMillis - previousMillis >= waitLightInterval) {
       // save the last time you blinked the LED
       previousMillis = currentMillis;

       // if the LED is off turn it on and vice-versa:
       if (ledState == LOW) {
         ledState = HIGH;
         //Serial.println("now high");
       } else {
         ledState = LOW;
         //Serial.println("now low");
       }
       //Serial.println("loop");

       // set the LED with the ledState of the variable:
       digitalWrite(LED, ledState);
     }
   //Serial.println("waiting for Sync button");
   delay(500);
 }



 //Start Up Access Point
 WiFiServer server(80);
 setupWiFi();
 server.begin();
 Serial.println("button hit");
 previousMillis = millis();




 //Listen for identifiers from hub
 while (!passwordIn || !ssidIn || !ipIn || !identityIn) //all four identifiers must be received
 {
   entryLength = true; //assume incoming entry is correct length


   // Check if a client has connected
   WiFiClient client = server.available();
   if (client) {
     Serial.println("client connected");
   }


   // Read the first line of the request
   String req = client.readStringUntil('\r'); //req holds the submission from the hub

   //Remove unnecessary components of submission (chars added that are unneeded)
   int trim = req.length() - 9;
   req.remove(trim);
   Serial.println(req);
   client.flush();

   //if ssid sent, record as ssid
   if (req.indexOf("ssid/") != -1) {
     req.remove(0, 10);
     req.toCharArray(ssid, 32);
     ssidLength = req.length();
     ssidIn = true;
     if (ssidLength > 32) {
       Serial.println("too long");
       entryLength = false;
       ssidIn = false;
     }
     Serial.print("ssidLength is:");
     Serial.println(ssidLength);

     Serial.print("ssid in: ");
     Serial.print(ssid);
     Serial.println("finish");
   }

   //if password sent, record as password
   else if (req.indexOf("password/") != -1) {
     req.remove(0, 14);
     req.toCharArray(password, 32);
   passLength = req.length();
   passwordIn = true;
   if (passLength > 32) {
     entryLength = false;
     passwordIn = false;
   }
   Serial.print("passLength is:");
   Serial.println(passLength);
   Serial.print("Password in: ");
   Serial.print(password);
   Serial.println("finish");
 }

 //if ip address sent, record as ip address
 else if (req.indexOf("ip/") != -1) {
   req.remove(0, 8);
   req.toCharArray(ip, 32);
   ipLength = req.length();
   ipIn = true;
   if (ipLength > 32) {
     entryLength = false;
     ipIn = false;
   }
   Serial.print("ipLength is:");
   Serial.println(ipLength);
   Serial.print("IP in: ");
   Serial.print(ip);
   Serial.println("finish");
 }

 //if identity sent, record as identity
 else if (req.indexOf("identity/") != -1) {
   req.remove(0, 14);
   req.toCharArray(identity, 32);
   identityLength = req.length();
   identityIn = true;
   if (identityLength > 32) {
     entryLength = false;
     identityIn = false;
   }
   Serial.print("identityLength is:");
   Serial.println(identityLength);
   Serial.print("Identity in: ");
   Serial.print(identity);
   Serial.println("finish");
 }
 //-----------------------------------------------------------------



 //alert the hub if the sent string is too many characters (over 32)
 if (!entryLength) {
   req = "too long";
 }


 //Compose Response
 String s = "HTTP/1.1 200 OK\r\n";
 s += "Content-Type: text/html\r\n\r\n";
 s += "<!DOCTYPE HTML>\r\n<html>\r\n";
 s += "String is";
```

```
  s += "<br>";
  s += req;

  // Send the response to the client
  client.print(s);
  delay(1);

  //Timer
  currentMillis = millis();


  // if the LED is off turn it on and vice-versa:
  if (ledState == LOW) {
    ledState = HIGH;
    //Serial.println("now high");
  } else {
    ledState = LOW;
    //Serial.println("now low");
  }
  digitalWrite(LED, ledState);



  //Pull out of loop and restart device if nobody there
  if (currentMillis - previousMillis >= syncTimeout) {
    previousMillis = currentMillis;
    Serial.println("Going to sleep - nobody talking");
    goto initiateSleep;
  }

}
infoIn = true; //identifiers in place; proceed with MQTT
digitalWrite(LED, LOW);
WiFi.mode(WIFI_OFF);

//--------------------------Record All Credentials------------------------
//Start with SSID Addresses
Serial.print(" SSID length is  ");
Serial.println(ssidLength);
delay(500);
addrWrite = 0;
int ssidStart = addrWrite;
Serial.print("starting at address ");
Serial.println(ssidStart);
delay(500);
int ssidEnd = addrWrite + ssidLength;
Serial.print("ending at address ");
Serial.println(addrWrite);


//write ssid to ssid addresses
for (addrWrite; addrWrite <= ssidEnd; addrWrite++) {
  EEPROM.write(addrWrite, ssid[addrWrite - ssidStart]);
  EEPROM.commit();
  Serial.print("Address ");
  Serial.print(addrWrite);
  Serial.print(" has :");
  Serial.println(ssid[addrWrite - ssidStart]);
}

//prepare to write password - calculate length, startpoint, endpoint
```

```
Serial.print(" Password length is  ");
Serial.println(passLength);
delay(500);
addrWrite = 32;
int passStart = addrWrite;
Serial.print("starting at address ");
Serial.println(addrWrite);
delay(500);
int passEnd = addrWrite + passLength;
Serial.print("ending at address ");
Serial.println(passEnd);

//write password to password addresses
for (addrWrite; addrWrite <= passEnd; addrWrite++) {
  EEPROM.write(addrWrite, password[addrWrite - passStart]);
  EEPROM.commit();
  Serial.print("Address ");
  Serial.print(addrWrite);
  Serial.print(" has :");
  Serial.println(password[addrWrite - passStart]);
}

//prepare for EEPROM writing - calculate startpoint, endpoint, and length
Serial.print(" ip length is  ");
Serial.println(ipLength);
delay(500);
addrWrite = 64;
int ipStart = addrWrite;
Serial.print("starting at address ");
Serial.println(addrWrite);
delay(500);
int ipEnd = addrWrite + ipLength;
Serial.print("ending at address ");
Serial.println(ipEnd);

//loop through and write to ip address addresses
for (addrWrite; addrWrite <= ipEnd; addrWrite++) {
  EEPROM.write(addrWrite, ip[addrWrite - ipStart]);
  EEPROM.commit();
  Serial.print("Address ");
  Serial.print(addrWrite);
  Serial.print(" has :");
  Serial.println(ip[addrWrite - ipStart]);
}

//figure out addresses to store identity
Serial.print(" identity length is  ");
Serial.println(identityLength);
delay(500);
addrWrite = 96;
int identityStart = addrWrite;
Serial.print("starting at address ");
Serial.println(addrWrite);
delay(500);
int identityEnd = addrWrite + identityLength;
Serial.print("ending at address ");
Serial.println(identityEnd);

//store identity
for (addrWrite; addrWrite <= identityEnd; addrWrite++) {
  EEPROM.write(addrWrite, identity[addrWrite - identityStart]);
```

```
    EEPROM.commit();
    Serial.print("Address ");
    Serial.print(addrWrite);
    Serial.print(" has :");
    Serial.println(identity[addrWrite - identityStart]);

  }
}
//-------------end of access point; following is if EEPROM loaded----
else if (eepromCheck != '\0') {
  Serial.print("Info Found, Starting Read Loop");
  for (int k = 0; k <= 3; k++) {

    //Assign Correctly to SSID, Password, or IP Address
    switch (k) {

      case 0://SSID

        stringLength = 0;
        //Read Loop
        addrRead = 0; //ssid starts at address 0
        do  {
          Serial.println("Reading SSID");
          i = EEPROM.read(addrRead);
          ssid[stringLength] = EEPROM.read(addrRead);
          Serial.println(i);
          addrRead ++;
          stringLength ++;
        } while (i != '\0' && stringLength <= 32);
        delay(1000);
        Serial.print("SSID is ");
        Serial.println(ssid);
        ssidLength = stringLength - 1;
        break;

      case 1://Password

        stringLength = 0;
        addrRead = 32;
        //Read
        do  {
          Serial.println("Reading Password");
          i = EEPROM.read(addrRead);
          password[stringLength] = EEPROM.read(addrRead);
          Serial.println(i);
          //delay(1000);
          addrRead ++;
          stringLength ++;
        } while (i != '\0' && stringLength <= 32);
        delay(1000);
        Serial.print("Password is ");
        Serial.println(password);
        passLength = stringLength - 1;
        break;

      case 2://IP Address

        stringLength = 0;
        addrRead = 64;
        //Read
        do  {
          Serial.println("Reading IP Address");
          i = EEPROM.read(addrRead);
          ip[stringLength] = EEPROM.read(addrRead);
          Serial.println(i);
          //delay(1000);
          addrRead ++;
          stringLength ++;
        } while (i != '\0' && stringLength <= 32);
        delay(1000);
        Serial.print("IP is ");
        Serial.println(ip);
        ipLength = stringLength - 1;
        break;


      case 3://Identity

        //if there is no identity (having been erased from an earlier cycle) then don't bother reading it
        if (identityCheck == '\0') {
          break;
        }
        stringLength = 0;
        addrRead = 96;
        //Read Loop
        do  {
          Serial.println("Reading Identity");
          i = EEPROM.read(addrRead);
          identity[stringLength] = EEPROM.read(addrRead);
          Serial.println(i);
          //delay(1000);
          addrRead ++;
          stringLength ++;
        } while (i != '\0' && stringLength <= 32);
        delay(1000);
        Serial.print("Identity is ");
        Serial.println(identity);
        identityLength = stringLength - 1;
        break;

    }
  }
  infoIn = true; //identifiers in place, proceed with MQTT
}
//----------------------------------------------------------------------------------------



//--------------------------if EEPROM has info - > connect----------------------------
if (infoIn) {   //first, check if identifiers are in place
  Serial.println("made it to connect part");
  WiFiClient wf_client; // instantiate wifi client
  PubSubClient client(wf_client, ip); // pass to pubsub
  Serial.println();
  Serial.println();
  Serial.println(F("Modified pubsub client basic code using modified pubsub software"));

  // Connect to WIFI of Router.
  Serial.println(); Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
//Set Time Loop
previousMillis = millis();

client.set_callback(callback);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  currentMillis = millis();
  delay(500);
  Serial.print(".");
  if (ledState == LOW) {
    ledState = HIGH;
  } else {
    ledState = LOW;
  }
  digitalWrite(LED, ledState);

  //Pull out of loop and restart device if WiFi not found
  if (currentMillis - previousMillis >= wifiTimeout) {
    previousMillis = currentMillis;
    Serial.println("Going to sleep - no wifi");
    goto initiateSleep;
  }

}
Serial.println();
Serial.println("WiFi connected");
Serial.println("IP address: "); Serial.println(WiFi.localIP());
//-----------------------------------------------------------------------------



//----------------------------check valve status----------------------------
valve_check = EEPROM.read(125);
Serial.println(valve_check);
if (valve_check == 'o') {
  valve_closed = false;
  valve_status = "open";
}
else {
  valve_closed = true;
  valve_check = 'c';
  valve_status = "closed";
}

EEPROM.write(125, valve_check);
EEPROM.commit();
delay(2000);
Serial.println(EEPROM.read(125));
Serial.print("just wrote status: ");
Serial.println(valve_status);
//-----------------------------------------------------------------------------



//--------------------------------initiating mqtt cycle------------------------
Serial.println("Starting Loop");
if (WiFi.status() == WL_CONNECTED) {
  if (!client.connected()) {
    if (client.connect("mydevice", "Valve Code", 0 , false, "Valve Disconnected")) {
      digitalWrite(LED, HIGH);
      Serial.println("MQTT Connected");
Serial.println("connected and stuff");


//store identity in EEPROM
if (identityCheck == '\0' || eepromCheck == '\0') {
  Serial.println("no identity found");

  //do if identity was not just sent from an identity sync
  if (eepromCheck != '\0') {
    Serial.println("identity only thing missing, waiting for hub command");
    identityDecided = false;
    client.subscribe("commandIdentity");
    client.publish("identity", "ready for identity"); //added for version 2_2
    do {
      client.loop();
      delay(2000);
      Serial.print("identity decided is : ");
      Serial.println(identityDecided);
      if (!identityCorrectLength) {
        client.publish("identity", "too long");
      }
      if (!client.connected()) {
        Serial.println("Lost MQTT, jumping out of loop, sleeping and trying again");
        identityDecided = true;
      }
    } while (!identityDecided );
    Serial.print("identity is : ");
    Serial.println(identity);
    client.publish("identity received", "identity");
    client.unsubscribe("commandIdentity");
  }


  //figure out addresses to store identity
  Serial.print(" identity length is  ");
  Serial.println(identityLength);
  delay(500);
  addrWrite = 96;
  int identityStart = addrWrite;
  Serial.print("starting at address ");
  Serial.println(addrWrite);
  delay(500);
  int identityEnd = addrWrite + identityLength;
  Serial.print("ending at address ");
  Serial.println(identityEnd);

  //store identity
  for (addrWrite; addrWrite <= identityEnd; addrWrite++) {
    EEPROM.write(addrWrite, identity[addrWrite - identityStart]);
    EEPROM.commit();
    Serial.print("Address ");
    Serial.print(addrWrite);
    Serial.print(" has :");
    Serial.println(identity[addrWrite - identityStart]);
  }
}


//Publish Active Signal and Subscribe to Hub
valve.concat(identity); //create unique topic string for valve to speak to hub
command.concat(identity); // create unique topic string for hub to speak to valve
```

```
client.publish(valve, "active"); //publish the valve status to the hub
Serial.print("just published, now listening to :");
Serial.println(command);
client.subscribe(command); //listen for commands from the hub
//--------------------------------------------------------------------

if (client.connected()) { //poo
  do {
    client.loop();
    delay(2000);
    Serial.println("waiting for responses");
    Serial.print("Wakeset is : ");
    Serial.println(wakeSet);
    Serial.print("Resetset is : ");
    Serial.println(resetSet);
    Serial.print("identityResetSet is : ");
    Serial.println(identityResetSet);
    Serial.print("valve_set is : ");
    Serial.println(valve_set);
    Serial.print("valve is: ");
    Serial.println(valve_status);
    if (!client.connected()) {
      Serial.println("Lost MQTT, jumping out of loop, sleeping and trying again");
      reset = false;
      identityReset = false;
      wake = true;
      wakeSet = true;
      resetSet = true;
      identityResetSet = true;
      valve_set=true;
    }
  } while (!wakeSet || !resetSet || !identityResetSet || !valve_set);

  client.loop();
  Serial.print("Wake is : ");
  Serial.print(wake);
  Serial.print("Reset is : ");
  Serial.println(reset);
  Serial.print("Identity Reset is : ");
  Serial.println(identityReset);
  delay(1000);
  Serial.println("about to decide");

  //Calculate Sleep Time from Sleep Commands
  if (wake) {
    Serial.println("test");
    sleepMultiplier = 3;
  }
  else {
    sleepMultiplier = 20;
  }
  //-----------------------------------------------------------------------

  //Note: the EEPROM write code (currently associated with the Access Point code) originally
  //belonged here, the idea being that the device would only save Identifiers after a first,
  //successful sync. However, our designed boards didn't always connect reliably - sometimes
  //they would successfully sync and receive identifiers, but the MQTT or WiFi connections
  //would fall through and the identifiers would be unsaved. So instead, the EEPROM saving code
  //is placed right after the identifiers are received in the sync.

  //-------------------close or open valve----------------------------------
```

```
      if (command_close != valve_closed) {
        client.publish("closedtest", "here");
        if (command_close == true) {
          //Serial.println("closing the valve");
          EEPROM.write(125, 'c');
          EEPROM.commit();
          valve_closed = true;
          valve_status = "closed";
          client.publish(valve, valve_status);
          digitalWrite(13, LOW);// Set GPIO13 to low to set direction to "closing"
          delay(200);
          digitalWrite(14, HIGH);// Set GPIO14 to high to power the actuator
          delay(3000);
          digitalWrite(14, LOW);// Set GPIO14 to low  to stop the actuator
          delay(500);
        }
        else {
          //Serial.println("opening the valve");
          EEPROM.write(125, 'o');
          EEPROM.commit();
          valve_closed = false;
          valve_status = "open";
          client.publish(valve, valve_status);
          digitalWrite(13, HIGH);// Set GPIO13 to high to set direction to "opening"
          delay(200);
          digitalWrite(14, HIGH);// Set GPIO14 to high to power the actuator
          delay(3000);
          digitalWrite(14, LOW);// Set GPIO14 to low  to stop the actuator
          delay(500);


        }


        delay(1);
        Serial.println("just published");

      }

      //-----------------------------------------------------------------------

      //-----------------------------Reset and Identity Reset--------------------------
      //---------Reset
      if (reset) {
        for (int i = 0; i < 128; i++)
          EEPROM.write(i, '\0');
        EEPROM.commit();
        client.publish(valve, "reset");
      }
      else if (identityReset) { //-------Identity Reset
        for (int i = 96; i < 128; i++)
          EEPROM.write(i, '\0');
        EEPROM.commit();
        client.publish(valve, "reset");
      }
      Serial.println("MQTT Disconnected");
      delay(500);
    }
  }
}
Serial.println("No Connection");
```

```
      delay(500);
    }
    Serial.println("made it to end");
  }
  //Initiate Deep Sleep - End of Cycle
initiateSleep://if any connection cycles took too long
  Serial.println("going to sleep now");
  sleepTime = sleepMultiplier * 1000000;
  ESP.deepSleep(sleepTime, WAKE_NO_RFCAL);

}



//192.168.4.1 is IP address of Access Point (for debug purposes)




//===================Function to set up access point ==============
void setupWiFi() {
  WiFi.mode(WIFI_AP);

  uint8_t mac[WL_MAC_ADDR_LENGTH];
  String AP_NameString = "irrigateSync";

  char AP_NameChar[AP_NameString.length() + 1];
  memset(AP_NameChar, 0, AP_NameString.length() + 1);

  for (int i = 0; i < AP_NameString.length(); i++)
    AP_NameChar[i] = AP_NameString.charAt(i);

  WiFi.softAP(AP_NameChar);
}
//=================================================================

//========================Callback Function=======================
void callback(const MQTT::Publish & pub) {
  // handle message arrived
  Serial.print("Message arrived [");
  Serial.print(pub.topic());
  Serial.print("] ");

  Serial.println(pub.payload_string());
  delay(2000);
  Serial.println();
  //check first three characters of message - having to do with sleep cycle
  if (pub.payload_string().substring(0, 3) == "wak") {//wake command
    wake = true;
    wakeSet = true;
    Serial.println("wake mode");
  }
  else if (pub.payload_string().substring(0, 3) == "slp") {//sleep command
    wake = false;
    wakeSet = true;
    Serial.println("sleep mode");
  }
  //check lower-middle three characters of message - having to do with full reset
  if (pub.payload_string().substring(3, 6) == "rst") {//initiate full reset
    reset = true;
    resetSet = true;
    Serial.println("Resetting Identifiers");
  }
  else if (pub.payload_string().substring(3, 6) == "sta") {//maintain all identifiers
    reset = false;
    resetSet = true;
    Serial.println("Keeping Identifiers");
  }
  //check upper-middle three characters of string - having to do with identity reset
  if (pub.payload_string().substring(6, 9) == "rst") {//command to reset identity
    identityReset = true;
    identityResetSet = true;
    Serial.println("Resetting Name");
  }
  else if (pub.payload_string().substring(6, 9) == "sta") {//command to maintain identity
    identityReset = false;
    identityResetSet = true;
    Serial.println("Keeping Name");
  }
  //check last three characters of message - having to do with commands to open or close valve
  if (pub.payload_string().substring(9, 12) == "cls") {
    command_close = true;
    valve_set = true;
    Serial.println("Want Valve Closed");
  }
  else if (pub.payload_string().substring(9, 12) == "opn") {
    command_close = false;
    valve_set = true;
    Serial.println("Want Valve Open");
  }
  else {//the only other message that could have been sent would be an identity during an identity syn
    pub.payload_string().toCharArray(identity, 32);
    identityLength = pub.payload_string().length();
    if (identityLength > 32) {
      identityDecided = false;
      identityCorrectLength = false;
    }
    else {
      identityDecided = true;
      identityCorrectLength = true;
    }

    Serial.println("identity is reset");
  }
}

//=================================================
```

D: Hub Code (Python and Server Programs)

Looping Code - Background Code for Hub (Python): send_all_from_files_11.py

```
#This code is the looping function which loops through all pylons and valves and
#formulates and sends commands based on the content of the files.
#It also writes to the files based on MQTT responses from the devices in real time.
import paho.mqtt.client as mqtt
import time
import os
from pathlib import Path

# following only used for MQTT debug
def on_connect(client, userdata, flags, rc):
    if rc==0:
        client.connected_flag=True
    else:
        client.connected_flag=False
    print("connected. flag func is: ", client.connected_flag)

def write_to_directories(device_type, device_name, message_thing):
    with open("/home/pi/IrrEEgation/"+device_type+"s/"+device_name, "r") as g:
        state = g.readlines()
    g.close
    print(message_thing)
    if message_thing == "reset":
        os.remove("/home/pi/IrrEEgation/"+device_type+"s/"+device_name) #remove the file from directory
        client1.unsubscribe("command"+device_type+device_name) #unsubscribe from removed file
    else:
        state[0]=device_name+"\n" #re-affirms identity, indicative of successful sync
        if message_thing!="active": #distinguish pylon response from valve response, assign wet or dry
            state[4]=message_thing+'\n'
            print(state[4])
        with open("/home/pi/IrrEEgation/"+device_type+"s/"+device_name, "w") as g:
                g.writelines(state)
        g.close

    if message_thing=="wet":
        #tell corresponding valve to close
        action_valve=state[5]
        print("looking for ", action_valve)
        check_file=Path("/home/pi/IrrEEgation/valves/"+action_valve)
        if check_file.is_file():
            print("valve found: ", action_valve)
            with open("/home/pi/IrrEEgation/valves/"+action_valve, "r") as g:
                valve_state = g.readlines()
            g.close
            valve_state[4]="closing\n" #closing until valve indicates that it successfully closed
            with open("/home/pi/IrrEEgation/valves/"+action_valve, "w") as g:
                g.writelines(valve_state)
            g.close




def on_message(client1, userdata, message):
    #turn messages into distilled, useful pieces
    message_thing = str(message.payload)[2:-1]
    topic_thing = str(message.topic)
    device_type=topic_thing[0:5]
    device_name = str(message.topic)[5:]
    write_to_directories(device_type, device_name, message_thing)

#define paths for writing and reading
pylon_path= "/home/pi/IrrEEgation/pylons/"
valve_path="/home/pi/IrrEEgation/valves/"
broker_address="192.168.111"
client1 = mqtt.Client("P1")
client1.on_connect= on_connect
client1.on_message=on_message
```

```
#loop through all pylons and valves
while True:
  client1.connect(broker_address)
  client1.loop_start()
  time.sleep(1)
  print("connected. flag first is: ", client1.connected_flag)
  while client1.connected_flag==True:
    for filename in os.listdir(pylon_path):
      with open(pylon_path+filename, "r") as f:
        state = f.readlines()

      if state[0]=="syncing\n": #if device not fully integrated into system yet, used for identity sync
        client1.publish("commandIdentity", filename)
        #client1.subscribe("pylon"+filename)
      command = "commandpylon"+filename
      listen = "pylon"+filename
      print(command)
      #print(listen)
      if "noReset" in state[1]:
        print("we shall not reset")
        reset_string="sta"
      else:
        print("we shall reset")
        reset_string="rst"

      if "identitySame" in state[2]:
        print("we shall keep the identity the same")
        identity_string = "sta"
      else:
        print("we shall forget our identity")
        identity_string = "rst"

      if "awake" in state[3]:
        print("we shall stay awake")
        sleep_string = "wak"
      else:
        print("we shall sleep")
        sleep_string = "slp"
      command_string=sleep_string+reset_string+identity_string #concatenate to form the full command string
      client1.publish(command,command_string)
      client1.subscribe(listen)


      f.close()

    for filename in os.listdir(valve_path): #repeat for valves
      with open(valve_path+filename, "r") as f:
        state = f.readlines()

      if state[0]=="syncing\n":
        client1.publish("commandIdentity", filename)
      command = "commandvalve"+filename
      listen = "valve"+filename
      print(command)
      #print(listen)
      if "noReset" in state[1]:
        print("we shall not reset")
        reset_string="sta"
      else:
        print("we shall reset")
```

```
                reset_string="rst"

            if "identitySame" in state[2]:
                print("we shall keep the identity the same")
                identity_string = "sta"
            else:
                print("we shall forget our identity")
                identity_string = "rst"

            if "awake" in state[3]:
                print("we shall stay awake")
                sleep_string = "wak"
            else:
                print("we shall sleep")
                sleep_string = "slp"

            if "close" in state[4]:
                print("we shall close")
                valve_string = "cls"
            elif "closing" in state[4]:
                print("we shall keep closing")
                valve_string = "cls"
            elif "open" in state[4]:
                print("we shall open")
                valve_string = "opn"
            else:
                print("we shall keep opening")
                valve_string = "opn"
            command_string=sleep_string+reset_string+identity_string+valve_string
            client1.publish(command,command_string)
            client1.subscribe(listen)

            f.close()

        time.sleep(10)
    client1.loop_stop()
    print("not connected")
    time.sleep(5)
```

E: Write Function on Hub - Connection from Server to Directories (Python):

write_function3.py

```
#This code is the function that the Flask server calls to write to the pylon and valve directories.
#Everything the user inputs is traced through this function.
#Files are read line by line as a list, and altered/re-written in the same format.
#Note, valve_state is only the generic name for any list, used for reading and writing. It doesn
#not refer specifically just to valves.
import urllib
import requests
import time
import subprocess
import os

def command_write_to_directories(device_type, device_name, action, action_string):
    path="/home/pi/IrrEEgation/"+device_type+"s/"+device_name
    new_name_path="/home/pi/IrrEEgation/"+device_type+"s/"+action_string
    preset=["name", "noReset", "identitySame", "awake", "dry", "/n"]
```

```
if action =="close":# close valve
    with open(path, "r") as g:
        valve_state = g.readlines()
    g.close
    valve_state[4]="closing\n"
    with open(path, "w") as g:
        g.writelines(valve_state)
    g.close
elif action =="open": #open valve
    with open(path, "r") as g:
        valve_state = g.readlines()
    g.close
    valve_state[4]="opening\n"
    with open(path, "w") as g:
        g.writelines(valve_state)
    g.close
elif action =="reset": #reset the memory of either device
    with open(path, "r") as g:
        valve_state = g.readlines()
    g.close
    valve_state[1]="Reset\n"
    with open(path, "w") as g:
        g.writelines(valve_state)
    g.close
elif action =="identityReset": #reset the identity only of either device
    with open(path, "r") as g:
        valve_state = g.readlines()
    g.close
    valve_state[2]="identityReset\n"
    with open(path, "w") as g:
        g.writelines(valve_state)
    g.close
    valve_state[2]="identitySame\n"
    valve_state[0]="syncing\n"
    with open(new_name_path, "w") as g:
        g.writelines(valve_state)
    g.close
elif action =="sync": #bring a new device (pylon or valve) onto the system
    print('syncing')
    time.sleep(15)#wait for access point to come online
    subprocess.Popen(["sudo", "wifi", "connect", "--ad-hoc", "irrigateSync"]) # force hub to connect to access point
    time.sleep(35)#wait for connection to access point to stabilize
    #send all identifiers to access point
    urllib.request.urlopen("http://192.168.4.1/ssid/dlink-674E")
    print('ssid')
    time.sleep(1)
    urllib.request.urlopen("http://192.168.4.1/identity/"+device_name)
    print('device')
    time.sleep(1)
    urllib.request.urlopen("http://192.168.4.1/ip/192.168.111")
    print('ip')
    time.sleep(1)
    print('password sending')
    urllib.request.urlopen("http://192.168.4.1/password/sgddr00595")
    print('password sent')
    time.sleep(1)
    #write new file for synced device
    preset[0]=device_name+"\n"
    preset[1]="noReset\n"
    preset[2]="identitySame\n"
    preset[3]="awake\n"
```

```
        if device_type=="pylon":
            preset[4] ="dry\n"
            preset[5]="\n"
        else:
            preset[4]="closed\n"
        with open(path, "w") as g:
            g.writelines(preset)
        g.close
    elif action =="assign_valve": #if valve synced, add final line for open/close command
        with open(path, "r") as g:
            valve_state = g.readlines()
        g.close
        valve_state[5]=action_string
        with open(path, "w") as g:
            g.writelines(valve_state)
        g.close
    elif action =="awake": #send all devices to sleep
        with open(path, "r") as g:
            valve_state = g.readlines()
        g.close
        valve_state[3]= "awake\n"
        with open(path, "w") as g:
            g.writelines(valve_state)
        g.close
    elif action =="asleep": #revert all devices to awake mode
        with open(path, "r") as g:
            valve_state = g.readlines()
        g.close
        valve_state[3]= "asleep\n"
        with open(path, "w") as g:
            g.writelines(valve_state)
        g.close
```

F: Set IP Address Function on Hub - (Python):

set_ip_func.py

```
import subprocess

subprocess.Popen(["sudo", "ifconfig", "eth0", "down"])
subprocess.Popen(["sudo", "ifconfig", "eth0", "192.168.111"])
subprocess.Popen(["sudo", "ifconfig", "eth0", "up"])
```

G: Server Function on Hub - Link between Python functionality and HTML (Python-Flask):

irrEEgation_v2.py

```
##================================IrrEEgation Application
Code======================================##
## Notre Dame Senior Design 2017
## Team IrrEEgation
## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
##    This code was developed to serve as the user interface for the Autonomous Flood Irrigation System.
##    It incorporates Python, Flask, and HTML to create a simple web application that interacts with
##    the user and the functional code running MQTT, dynamic file read/write, etc.
##==========================================================================================
======##
```

```
##====================================Code Starts
Here==============================================##

## Import libraries
from flask import Flask, render_template, request
from write_function3 import command_write_to_directories
from dataRefresh2 import pylonDataRefresh, valveDataRefresh
from read_function import read_from_directories
import paho.mqtt.client as mqtt
import time
import os

## Initialize Flask application
app = Flask(__name__)

## Initialize global arrays with current files for pylons and valves in
## /home/pi/IrrEEgation/pylons
## /home/pi/IrrEEgation/valves
[pylonData, pylonQuantity] = pylonDataRefresh()
[valveData, valveQuantity] = valveDataRefresh()

## Home Page
@app.route('/', methods=['GET', 'POST'])
def index():
  return render_template("index.html")

## Pylon Main Page
@app.route('/pylon_home', methods=['GET', 'POST'])
def pylon_home():
  global pylonData, pylonQuantity
  [pylonData, pylonQuantity] = pylonDataRefresh() ## Refresh data
  return render_template("pylon_home.html", pylonData=pylonData, pylonQuantity=pylonQuantity) ## pass data to HTML and
render page

## Valve Main Page
@app.route('/valve_home', methods=['GET', 'POST'])
def valve_home():
  global valveData, valveQuantity
  [valveData, valveQuantity] = valveDataRefresh() ## Refresh data
  return render_template("valve_home.html", valveData=valveData, valveQuantity=valveQuantity) ## pass data to HTML and
render page

## Sleep Main Page
@app.route('/set_sleep', methods=['GET', 'POST'])
def set_sleep():
  return render_template("set_sleep.html")

## Wake Devices Page
@app.route('/set_sleep/awake', methods=['GET', 'POST'])
def awake():
  global pylonData, pylonQuantity, valveData, valveQuantity
  for pylon in range(pylonQuantity):
    command_write_to_directories("pylon", pylonData[pylon][0].rstrip(), "awake", "awake") ## write "awake" to pylon files
  [pylonData, pylonQuantity] = pylonDataRefresh() ## Refresh data

  for valve in range(valveQuantity):
    command_write_to_directories("valve", valveData[valve][0].rstrip(), "awake", "awake") ## write "awake" to valve files
  [valveData, valveQuantity] = valveDataRefresh() ## Refresh data
  return render_template("awake.html")

## Sleep Devices Page
```

```
@app.route('/set_sleep/asleep', methods=['GET', 'POST'])
def sleep():
    global pylonData, pylonQuantity, valveData, valveQuantity
    for pylon in range(pylonQuantity):
        command_write_to_directories("pylon", pylonData[pylon][0].rstrip(), "asleep", "asleep") ## write "asleep" to pylon files
    [pylonData, pylonQuantity] = pylonDataRefresh() ## Refresh data

    for valve in range(valveQuantity):
        command_write_to_directories("valve", valveData[valve][0].rstrip(), "asleep", "asleep") ## write "asleep" to valve files
    [valveData, valveQuantity] = valveDataRefresh() ## Refresh data
    return render_template("asleep.html")

## Sync Main Page
@app.route('/sync', methods=['GET', 'POST'])
def sync():
    return render_template("sync.html")

## Sync Confirmation Page
@app.route('/sync/new', methods=['GET', 'POST'])
def sync_new():
    device_name = request.form['new_id'] ## Retrieve data from form
    device_type = request.form['Type']
    command_write_to_directories(device_type, device_name, "sync", "sync") ## Write new file to device directory
    return render_template("sync_new.html", device_name=device_name, device_type=device_type)

## Pylon Management Page
@app.route('/manage/<pylon>', methods=['GET', 'POST'])
def manage(pylon=None):
    return render_template("manage.html", pylon=pylon) ## Pass pylon name to management page and render page

## Valve Management Page
@app.route('/manage/valve/<valve>', methods=['GET', 'POST'])
def manage_valve(valve=None):
    valve_state = read_from_directories("valve", valve) ## Read valve open/close state
    if valve_state[4] == "open\n":
        valve_string = "Close"
    elif valve_state[4] == "closed\n":
        valve_string = "Open"
    elif valve_state[4] == "opening\n":
        valve_string = "Opening"
    else:
        valve_string = "Closing"
    return render_template("manage_valve.html", valve=valve, valve_string=valve_string) ## Pass valve state to management
page and render page

## Pylon General Reset Page
@app.route('/reset/<pylon>', methods=['GET', 'POST'])
def reset(pylon=None):
    command_write_to_directories("pylon", pylon, "reset", "Reset") ## Write reset to specified pylon file
    return render_template("reset.html", pylon=pylon)

## Valve General Reset Page
@app.route('/reset/valve/<valve>', methods=['GET', 'POST'])
def reset_valve(valve=None):
    command_write_to_directories("valve", valve, "reset", "Reset") ## Write reset to specified valve file
    return render_template("reset_valve.html", valve=valve)

## Pylon ID Reset Page
@app.route('/pylon_id_reset/<pylon>', methods=['GET', 'POST'])
def pylon_id_reset(pylon=None):
    return render_template("pylon_id_reset.html", selected='pylon_id_reset', pylon=pylon)
```

```
## Pylon ID Confirmation Page
@app.route('/pylon_id_reset/new/<pylon>', methods=['GET', 'POST'])
def pylon_id_new(pylon=None):
    if request.method == 'POST': ## Write new pylon name to file
        command_write_to_directories("pylon", pylon, "identityReset", request.form['new_id'])
        newpylon = request.form['new_id']
        global pylonData, pylonQuantity
        [pylonData, pylonQuantity] = pylonDataRefresh()
    return render_template("pylon_id_new.html", pylon=pylon, newpylon=newpylon) ## Pass new pylon name and render page

## Valve ID Reset Page
@app.route('/id_reset/valve/<valve>', methods=['GET', 'POST'])
def valve_id_reset(valve=None):
    return render_template("valve_id_reset.html", selected='valve_id_reset', valve=valve)

## Valve ID Confirmation Page
@app.route('/id_reset/valve/new/<valve>', methods=['GET', 'POST'])
def valve_id_new(valve=None):
    if request.method == 'POST': ## Write new valve name to file
        command_write_to_directories("valve", valve, "identityReset", request.form['new_id'])
        newvalve = request.form['new_id']
        global valveData, valveQuantity
        [valveData, valveQuantity] = valveDataRefresh()
    return render_template("valve_id_new.html", valve=valve, newvalve=newvalve) ## Pass new valve name and render page

## Pylon-Valve Assignment Page
@app.route('/change_valve/<pylon>', methods=['GET', 'POST'])
def change_valve(pylon=None):
    return render_template("change_valve.html", pylon=pylon, valveData=valveData, valveQuantity=valveQuantity)

## Pylon-Valve Confirmation Page
@app.route('/change_valve/new/<pylon>', methods=['GET', 'POST'])
def change_valve_new(pylon=None):
    if request.method == "POST": ## Write new valve to pylon file
        command_write_to_directories("pylon", pylon, "assign_valve", request.form['Valve'])
        changevalve = request.form['Valve']
        global pylonData, pylonQuantity
        [pylonData, pylonQuantity] = pylonDataRefresh()
    return render_template("change_valve_new.html", pylon=pylon, changevalve=changevalve) ## Pass new valve assignment
and render page

## Valve Toggle Control Page
@app.route('/toggle/<valve>', methods=['GET', 'POST'])
def toggle(valve=None):
    if request.method == "POST":
        valve_state = read_from_directories("valve", valve) ## Read valve open/close status
        if valve_state[4] == "open\n":
            command_write_to_directories("valve", valve, "close", "close") ## Write new status to valve file
            valve_string = "is closing"
        elif valve_state[4] == "closed\n":
            command_write_to_directories("valve", valve, "open", "open")
            valve_string = "is opening"
        elif valve_state[4] == "opening\n":
            valve_string = "is opening"
        else:
            valve_string = "is closing"
        global valveData, valveQuantity
        [valveData, valveQuantity] = valveDataRefresh()

    return render_template("toggle.html", valve=valve, valve_string=valve_string)
```

```
## Launch Application on server 192.168.0.100
if __name__ == "__main__":
    app.run(host='192.168.0.100')
```

H: Data Refresh Function on Hub - (Python):

dataRefresh2.py

```
##=================================IrrEEgation Application
Code=====================================##
## Notre Dame Senior Design 2017
## Team IrrEEgation
## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
##    This code was developed to serve as a refresh function to update the wep application's
##    information.
##==============================================================================================
======##

##=====================================Code Starts
Here=============================================##

## Import libraries
import os

## Pylon Data Refresh function
def pylonDataRefresh():

    ## Import pylon data from file
    pylon_path = "/home/pi/IrrEEgation/pylons/" # Pi directory path
    file_list = os.listdir(pylon_path)

    ## Initialize variables and arrays
    count = 0
    pylonQuantity = len(file_list)
    pylonData = [[0 for x in range(6)] for y in range(pylonQuantity)]

    ## Read in from file
    for filename in file_list:
        with open(pylon_path+filename, "r") as f:
            state = f.readlines()

            ## Set data read to pylon data array
            pylonData[count][:] = [state[0], state[1], state[2], state[3], state[4], state[5]]
            count += 1
            f.close()

    return (pylonData, pylonQuantity) ## Return pylon data and number of pylons

## Valve Data Refresh function
def valveDataRefresh():
    ## Import valve data from file
    valve_path = "/home/pi/IrrEEgation/valves/"    # Pi directory path
    file_list = os.listdir(valve_path)

    ## Initialize variables and arrays
    count = 0
    valveQuantity = len(file_list)
    valveData = [[0 for x in range (5)] for y in range(valveQuantity)]
    ## Read in from file
    for filename in file_list:
        with open(valve_path+filename, "r") as f:
            state = f.readlines()

            ## Set data read to valve data array
            valveData[count][:] = [state[0], state[1], state[2], state[3], state[4]]
            count += 1
            f.close()

    return (valveData, valveQuantity) ## Return valve data and number of valves
```

I: Read Function on Hub - (Python):

read_function.py

```
##=================================IrrEEgation Application
Code=====================================##
## Notre Dame Senior Design 2017
## Team IrrEEgation
## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
##    This code was developed to read data from pylon and valve files.
##=========================================================================================
======##

##=====================================Code Starts
Here=============================================##

## Import libraries
import os

## Read function
def read_from_directories(device_type, device_name):
    path="/home/pi/IrrEEgation/"+device_type+"s/"+device_name ## Pi directory path
    with open(path, "r") as g:
        ## Read in from file
        data = g.readlines()
    g.close
    print(data)
    return data ## Return data read
```

J: Web Application Pages - (HTML):

```
<!--
 ##=================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "all devices asleep" confirmation page.

##=========================================================================================
======##
 -->

<!doctype html>
```

```
<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>All devices are asleep.</h1>

<!--
 ##===================================IrrEEgation Application
Code======================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "all devices awake" confirmation page.

##============================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>All devices are awake.</h1>

<!--
 ##===================================IrrEEgation Application
Code======================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "assign new valve" confirmation page.

##============================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>{{changevalve}} is now the valve for {{pylon}}.</h1>

<!--
 ##===================================IrrEEgation Application
Code======================================##
```

```
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "assign valve" user input page. It provides the user
 ##    with the available valves in a drop-down list.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>Change valve for {{pylon}}.</h1>

<form method="POST" action="/change_valve/new/{{pylon}}">
    <table class="t2">
        <tr>
            <td>
                <select name="Valve">
                    {% for valve in range(valveQuantity) %}
                    <option>{{valveData[valve][0]}}</option>
                    {% endfor %}
                </select>
            </td>
            <td><input type="submit" value="OK" /></td>
        </tr>
    </table>
</form>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "new ID" confirmation page.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>{{pylon}} has been reset to {{newpylon}}</h1>
```

```
<!--
 ##================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "ID reset" user input page. It provides the user with
 ##    a text input option.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>ID reset for {{pylon}}</h1>

<form method="POST" action="/id_reset/new/{{pylon}}">
    <table class="t2">
        <tr>
            <td><input type="text" name="new_id" /></td>
            <td><input type="submit" value="OK" /></td>
        </tr>
    </table>
</form>

<!doctype html>

<html>

    <head>
        <link rel="stylesheet" href='/static/style.css' />
    </head>

    <h1>IrrEEgation</h1>

    <table class="t2">
        <tr><td>
            <form method="GET" action="/pylon_home">
                <input type="submit" value="Pylon" class="button" />
            </form>
            </tr></td>
        <tr><td>
            <form method="GET" action="/valve_home">
                <input type="submit" value="Valve" class="button" />
            </form>
            </tr></td>
        <tr><td>
            <form method="GET" action="/set_sleep">
                <input type="submit" value="Sleep" class="button" />
            </form>
            </tr></td>
        <tr><td>
```

```
            <form method="GET" action="/sync">
                <input type="submit" value="Sync" class="button" />
            </form>
            </tr></td>
    </table>


</html>

<!--
 ##================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "valve management" user input page. It allows the
 ##    user to manage the "reset" status, ID, and open/close status of the valve.

##=============================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>Manage {{valve}}</h1>

<table class="t2">
    <tr><td>
        <form method="GET" action="/reset/valve/{{valve}}">
            <input type="submit" value="Reset" class="button" />
        </form>
    </td></tr>
    <tr><td>
        <form method="GET" action="/id_reset/valve/{{valve}}">
            <input type="submit" value="ID Reset" class="button" />
        </form>
    </td></tr>
    <tr><td>
        <form method="POST" action="/toggle/{{valve}}">
            {% if valve_string == "Open" %}
                <input type="submit" value="{{valve_string}}" class="button" />
            {% endif %}
            {% if valve_string == "Close" %}
                <input type="submit" value="{{valve_string}}" class="button" />
            {% endif %}
            {% if valve_string == "Opening" %}
                <input type="submit" disabled value="{{valve_string}}" class="button_dis" />
            {% endif %}
            {% if valve_string == "Closing" %}
                <input type="submit" disabled value="{{valve_string}}" class="button_dis" />
            {% endif %}
        </form>
    </td></tr>
```

```
</table>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "pylon management" user input page. It allows the
 ##    user to manage the "reset" status, ID, and assigned valve of the pylon.

##==================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>Manage {{pylon}}</h1>

<table class="t2">
    <tr><td>
        <form method="GET" action="/reset/{{pylon}}">
            <input type="submit" value="Reset" class="button" />
        </form>
    </td></tr>
    <tr><td>
        <form method="GET" action="/pylon_id_reset/{{pylon}}">
            <input type="submit" value="ID Reset" class="button" />
        </form>
    </td></tr>
    <tr><td>
        <form method="GET" action="/change_valve/{{pylon}}">
            <input type="submit" value="Valve" class="button" />
        </form>
    </td></tr>
</table>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the pylon main page. Here, the user can monitor all
 ##    connected pylons, their sleep statuses, their water statuses, and their assigned valves.
 ##    The user is also able to navigate to the management pages for each of the pylons listed.

##==================================================================================
======##
 -->

<!doctype html>
```

```
<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>Pylons</h1>

<table>
   <tr><th>Name</th><th>Status</th><th>Sensor</th><th>Valve</th></tr>
   {% for pylon in range(pylonQuantity) %}

<tr><td>{{pylonData[pylon][0]}}</td><td>{{pylonData[pylon][3]}}</td><td>{{pylonData[pylon][4]}}</td><td>{{pylonData[
pylon][5]}}</td>
       <td><form method="GET" action="/manage/{{pylonData[pylon][0]}}">
           <input type="submit" value="Manage" class="button" />
       </form>
       </td>
   </tr>
   {% endfor %}
</table>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "new pylon ID" confirmation page.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>{{pylon}} has been reset to {{newpylon}}</h1>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "new pylon ID" user input page. The user is provided
 ##    with a text input option to rename the pylon.

##=====================================================================================
======##
 -->
```

```
<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>ID reset for {{pylon}}</h1>

<form method="POST" action="/pylon_id_reset/new/{{pylon}}">
   <table class="t2">
      <tr>
         <td><input type="text" name="new_id" /></td>
         <td><input type="submit" value="OK" /></td>
      </tr>
   </table>
</form>

<!--
 ##=================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "valve reset" confirmation page.

##============================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>{{valve}} was successfully reset.</h1>

<!--
 ##=================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "pylon reset" confirmation page.

##============================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
```

```
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>{{pylon}} was successfully reset.</h1>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the sleep main page. Here the user is able to set all
 ##    devices to "awake" or "asleep" mode.

##==================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>Sleep Settings</h1>

<table class="t2">
   <tr><td>
      <form method="GET" action="/set_sleep/awake">
         <input type="submit" value="Wake" class="button" />
      </form>
   </td></tr>
   <tr><td>
      <form method="GET" action="/set_sleep/asleep">
         <input type="submit" value="Sleep" class="button" />
      </form>
   </td></tr>
</table>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "sync complete" confirmation page.

##==================================================================================
======##
 -->

<!doctype html>

<head>
```

```
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>{{device_name}} has been added to {{device_type}}s.</h1>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "user confirmation" of sync. Here the user is able to
 ##    verify that she wants to sync a new device.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>Confirm {{device_name}} as a new {{device_type}}?</h1>

<table class="t2">
    <tr><td>
        <form method="GET" action="/">
            <input type="submit" value="Yes" class="button" />
        </form>
    </td></tr>
</table>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the sync main page. Here the user is able to input a name
 ##    and type of new device.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>
```

```
<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>Sync new device.</h1>

<form method="POST" action="/sync/new">
   <table class="t2">
      <tr>
         <td><input type="text" name="new_id" /></td>
         <td><input type="submit" value="OK" /></td>
      </tr>
      <tr>
         <td>
            <select name="Type">
               <option>pylon</option>
               <option>valve</option>
            </select>
         </td>
      </tr>
   </table>
</form>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "toggle" confirmation page. The user will be notified
 ##    of the valve's performance of the selected operation.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>{{valve}} {{valve_string}}</h1>

<!--
 ##================================IrrEEgation Application
Code====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the valve main page. Here the user is able to monitor
 ##    all connected valves, their sleep statuses, and their gate statuses. The user is also able to
 ##    navigate to the management pages for each valve listed.

##=====================================================================================
======##
 -->
```

```
<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>Valves</h1>

<table>
   <tr><th>Name</th><th>Status</th><th>Gate</th></tr>
   {% for valve in range(valveQuantity) %}
   <tr><td>{{valveData[valve][0]}}</td><td>{{valveData[valve][3]}}</td><td>{{valveData[valve][4]}}</td>
      <td><form method="GET" action="/manage/valve/{{valveData[valve][0]}}">
         <input type="submit" value="Manage" class="button" />
      </form>
      </td>
   </tr>
   {% endfor %}
</table>

<!--
 ##================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "new valve ID" confirmation page.

##=====================================================================================
======##
 -->

<!doctype html>

<head>
   <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
   <input type="submit" value="H" class="button_home" />
</form>

<h1>{{valve}} has been reset to {{newvalve}}</h1>

<!--
 ##================================IrrEEgation Application
Code=====================================##
 ## Notre Dame Senior Design 2017
 ## Team IrrEEgation
 ## Kenneth Harkenrider, Nico Garcia, Trenton Kuta, Tyler Dale, Ellen Halverson
 ##    This code was developed to serve as the "new valve ID" user input page. Here the user is able
 ##    to input a new name for the valve.

##=====================================================================================
======##
 -->
```

```
<!doctype html>

<head>
    <link rel="stylesheet" href='/static/style.css' />
</head>

<form method="GET" action="/">
    <input type="submit" value="H" class="button_home" />
</form>

<h1>ID reset for {{valve}}</h1>

<form method="POST" action="/id_reset/valve/new/{{valve}}">
    <table class="t2">
        <tr>
            <td><input type="text" name="new_id" /></td>
            <td><input type="submit" value="OK" /></td>
        </tr>
    </table>
</form>
```

Style.css

```
h1 {
    color: ghostwhite;
    background: forestgreen;
    font-family: "Helvetica";
    text-align: center;
    margin: auto;
    width: 75%;
}

body {
    color: white;
    background: forestgreen;
}

.button {
    background-color: lightgreen;
    color: darkgreen;
    margin: auto;
    width: 150px;
    text-align: center;
    padding: 15px 30px;
    border: 3px solid forestgreen;
    font-family: "Helvetica";
    font-size: 20px;
}

.button:hover {
    color: forestgreen;
    background-color: floralwhite;
}

.button_dis {
    background-color: gray;
    color: darkgray;
    margin: auto;
    text-align: center;
    width: 150px;
    padding: 15px 30px;
    border: 3px solid forestgreen;
    font-family: "Helvetica";
    font-size: 20px;
}

.button_home {
    background-color: lightgreen;
    color: darkgreen;
    margin: auto;
    text-align: center;
    padding: 10px 13px;
    border: 1px solid forestgreen;
    font-family: "Helvetica";
    font-size: 30px;
}

table {
    border-collapse: collapse;
    width: 40%;
    margin: auto;
    font-family: "Helvetica";
}

th, td {
    text-align: left;
    padding: 4px;
}

th {
    background-color: darkgreen;
    color: white;
}

.t2 {
    border-collapse: collapse;
    width: 10%;
    margin: auto;
}
```