CS462 CAPSTONE PROGRESS REPORT

FEBRUARY 17, 2018- WINTER TERM

DEVELOPING AN INTERNET OF THINGS IRRIGATION VALVE

PREPARED AND ISSUED BY

GROUP 35

CHRISTOPHER CARLSEN
YIZHENG WANG
PETER DORICH

Abstract

This document is a .

Contents

1 Introduction

The Internet of Things Irrigation Valve project is an open-source, proof of concept project that aims to create an internet-connected, irrigation system control suite. The device is being developed with farmers/agriculturists in mind, and is an attempt to design a more sustainable and efficient method of watering specific, targeted plots of land. The current method of irrigation employed by the majority of agriculturists generally relies on either the user physically engaging/disengaging the irrigation valves, or using a simple (”dumb”) timer to do the job for them. This project will deliver an IoT irrigation system with automatic control. The three major goals of this project are completion of a graphical user interface, a wireless hub, and soil moisture valves. Each section of the project will work coherently as system.

2 Valve Relay

Author: Christopher Carlsen

As a quick refresher, the valve-control relay and soil-moisture sensor segment of this project is the main data collection and irrigation valve control device. The first of its two primary jobs is to gather soil-moisture data from the immediate area surrounding the irrigation valve that the device is attached to, which is then sent to a central hub for upload to the internet. The second job is then to receive instructions from the user on the preferred times for the irrigation valve to be engaged for watering.

2.1 Current Status

I’ve decided to discuss my progress on this project by comparison to the requirements established in the requirements document. The first of these is the devices ability to gather soil-moisture data. This goal was the initial focus of this segment of the project, as it is the primary information on which this project is built. Weeks two through four of this term were spent understanding, implementing and trouble-shooting the use of the SDI-12 library, which is the library being used to read data from the Decagon GS3 sensor devices. I had some difficulty implementing this initially, which will be explained in the problems section. After about a week and half of frustration and little progress, I switched hardware to an M0 style Adafruit Feather board. After spending some time refactoring the day that I switched, I was reading good data from the sensor device and was able to record and report it to the screen via a serial-port monitor. The Decagon sensor reports its data in character-array/string format, which (thankfully) made parsing the data into useful values a mostly trivial task. By the end of week four I had good data which flowed naturally into working on the second requirement.

The second requirement for this device was that it can reliably upload its gathered data to a central hub device. The majority of the fourth and fifth weeks, and most of week 6 were spent getting this implemented. Initial progress for this went relatively quick, and Peter and I were able to pass simple character data between ourselves after about a week. Our client had, early in the term, stated that they would like to use a forward-slash delimited string as their method of data packing. I was initially achieving this by manually building a character array and then passing over LoRa to the hub device to be parsed and handled. However, around the end of week four, our client asked that we try and switch to an implementation of a library known as Open Sound Control (OSC) to bundle the data. The RadioHead library that our group is implementing to communicate over LoRa radio only allows the transmission of uint8_t type data, which was making transmission of was making transmission of more robust data structures a bit challenging. Interestingly, another

Capstone team with a project in the OPEnS lab was working on a solution to integrate various devices and libraries and had developed a sample codebase to handle the packing and transmission of OSC bundles over LoRa radio, which we are now basing our networking off of. This led to some issues of its own that I will cover below, and eventually it was decided that a change of hardware to the M0 was necessary for this as well, but we now have a working LoRa communication solution that the client is happy with. This did require some small changes to the code-base but was otherwise a simple change-over that only took a portion of a day to complete.

The third requirement for my device, to programmatically control an irrigation valve, is at a functionally demonstrable level of completion. This was the most straight-forward requirement for my part of the project by far, but comically, was also the most stressful to test for fear that I might somehow blow up the OPEnS lab. The end of week six was spent thinking through the most logical electrical circuit design to run the valve-controller bi-directionally, so that it would both open and close the valve. To implement this two separate relay shields were used to run the valve-controller. While the circuit is rather simple, I unfortunately have very little electronics experience and had to spend some time understanding how relays worked, as well as circuit completion. However, after a long evening in the OPEnS lab snipping wires and such, I had a functional valve-controller which can be controlled programmatically.

2.2 Left To Do

There is one primary remaining piece from the valve-hub communication requirement that I mentioned above. Because of some of the changes that were made to our methodology of data-message transport, the communication between my device and the hub is currently unidirectional, meaning that we are only passing messages from my device to the hub. The next step for this will be to develop a method for handling and storing instructions from the hub device when sent by the client. While the current codebase I have should allow for a (hopefully) simple implementation of this, networking code and timing has proven to be challenging at times, so I expect this to be a large part of the remaining time spent on this project.

Of my four documented requirements, the last requirement is the only one yet remaining to be implemented in some form. This is to develop a fail-safe contingency that guarantees continued, controlled irrigation in the event of a loss of network connection with either the hub device, or the internet at large. The planned solution for this is to keep a base set of watering pattern instructions stored in non-volatile memory of the device, which will update at the request of the user, so that upon loss of network functionality, the device can continue watering at a base level of functionality. With the previous three segments of the valve-control system now in a near-to-complete level of readiness, I will begin implementation of this within the next week or so.

2.3 Problems & Possible Solutions

As likely should have been expected, I ran into my fair share of problems in developing solutions for this project. The first major issue was a compatibility issue that I originally encountered while trying to read data from the Decagon moisture sensors using the SDI-12 library. While the SDI-12 library is a fairly robust library, it requires use of interrupt pins on the micro-controller board to operate correctly. This would normally be alright, but the 32u4 micro-controller was particularly limited in this area and was having collision issues with another library that was claiming all the pins. Resolving this took over a week, which was unfortunate and frustrating, but is now in a functional state.

The second big issue that slowed down progress on this project was also, as far as I can tell, a hardware limitation of the 32u4 board. I spent a few days working in tandem with another Capstone member who developed the OSC bundle-handling codebase that our project now implements. After almost a week of trying to get the LoRa radio implementation of Open Sound Control working with the 32u4, a new approach was to implement new hardware, which was then up and running within a few hours of work.

Because the two previously mentioned issues appear to be primarily hardware limitations of the 32u4 micro-controller board, my device is now being implemented on the M0 flavor of the Adafruit Feather series of micro-processors.

2.4 Relevant & Interesting Information

I dont have a lot of interesting details to share about this project just yet, as so much of my time up to this point has been spent building the core structures of the device. I do want to elaborate slightly though, on an interesting situation that I had mentioned previously however, which is that I have been working regularly with a student from another Capstone team in the OPEnS lab. His groups' project is to develop a generic library of sample code using various micro-controllers and sensors in, and their various libraries. This was an unexpected and all-together enjoyable experience, and it was super interesting to work with an otherwise unrelated person on solving a mutually beneficial problem for the OPEnS lab.

3 Hub

3.1 Goals and Current Status

Author: Peter Dorich

The second portion of the project, known as the Wireless Hub, is designed to transfer information from the client to the valve, and in reverse. The valve will communicate to the Hub via Long range radio signals. The client will connect to the hub using MQTT with an Ethernet cable. The hub, at the end of the project, must successfully transmit instructions to the valve, and also transmit soil data to a user interface. Currently, the hub is in a state where it is able to transfer data to a user interface from a valve, but cannot yet transfer instructions to the valve. The next 4 weeks of the term will focus on that particular functionality.

3.2 Overview of Term Progress

By December 2017, the wireless hub portion of the project was in a pre-development stage. Small demos were produced that demonstrated certain project requirements for proof of concept purposes. This included a demo that utilized LoRa to turn on an LED on a separate microcontroller.

Since January, actual development has been underway on the hub. The first thing that our team wanted to accomplish during weeks one through three with the hub was getting it connected to the Internet via MQTT. MQTT is a messaging protocol that allows the hub to publish soil data to a graphical user interface for viewing. Since the microcontroller board we chose is not equipped with Wi-Fi, the only way to connect it to the world wide web is through an Ethernet cable. The Feather 32u4 board doesnt have a built in Ethernet port either, so female headers were soldered onto it to allow us to plug in an Adafruit Ethernet FeatherWing. This FeatherWing gave the 32u4 board internet access through an Ethernet cable. To authorize our 32u4 board, we needed to acquire a valid MAC address and a valid IP address for the secure OSU network. These values were supplied and hardcoded into the program. The next step was to connect to

Adafruit.io, which is a service that can display our data. The OPEnS lab has an Adafruit.io subscription, so connecting is as easy as hardcoding a unique account ID number.

To test the setup, we ran a program on the 32u4 microcontroller that published an incrementing value to our Adafruit.io dashboard every few seconds. After opening our Adafruit.io dashboard on a web browser, we witnessed it displaying the value just as it should. This code is the base code for publishing our soil moisture data.

After ensuring that we can publish our data to an online resource, week four was focused on receiving the data. The wireless hub receives soil moisture data from the valves through LoRa. To begin implementing this portion, we tested sending soil data from one 32u4 with a Decagon soil moisture sensor to the hub. Once the hub received a message, the corresponding RSSI value, which is a signal strength indicator for a radio signal, was published to Adafruit.io via MQTT. We could confirm the correctness by witnessing the RSSI value show up on the Adafruit.io dashboard. Our client was even able to view the data remotely on his computer.

The goal for weeks five and six were to incorporate the OPEnS Open Sound Control library in our messaging system. The OSC library was created by OPEnS to pass information to and from devices using a standard string format. The format is as follows: /LOOM/IrrVal/0/VWC f1065353216. The OSC library automatically builds a bundle of these formatted strings using the corresponding data values. LoRa isnt powerful enough to transfer this bundle, however, so OSC breaks it down into separate strings for sending. On the receiving end, OSC once again builds the bundle out of the received strings. Once this messaging protocol is implemented, separate values such as temperature and soil moisture can be published to Adafruit.io for viewing.

Over the course of the next 4 weeks, the hub will need to support a buffer, as well as bidirectional communication. The bidirectional communication will allow the user to turn the valve on or off. The buffer is needed due to the nature of the system. The system is designed to have the valve send soil data every 20 minutes. When it is not sending, the microcontroller is sleeping and saving battery. Because of this fact, the hub needs to store incoming commands in a buffer until the valve wakes up.

3.3 Problems and Solutions

During OSC library testing, we noticed an issue with our feather 32u4 boards. The microcontrollers would send and recieve multiple messages, but after a minute they would hang and do nothing. After debugging the library and investigating our code, we determined it to be an unknown compatibility issue with the OSC library and the feather 32u4 boards. To get around this issue, we decided to adopt and use the M0 board instead, which will work perfectly in our project. The M0 is also equipped with a LoRa radio as well. The unknown bug will be reviewed by the OPEnS lab for further investigation.

4 Client UI

Author: Yizheng Wang

The third part of the project, this will be the actual interface with which this systems users will interact. Users will use this system to view recorded moisture data from various sensors, as well as use it to make decisions on when and where water is most needed in their field. This system will take user input and send it to desired watering schedules (via the relay hub) to the individual sensor/valve-control devices in the desired watering areas.

4.1 Current Status

I have implement most of the fundamental functionalities of the web applications right now. Its possible to receive data from Adafruit.io, display it to user, and take input from user and send it to Adafruit.io. For each requirement of the project.

1. Receive data from Adafruit.io

This part is implemented by a function in the request.java file. It will pull a request from Adafruit.io by using the AIO-key of account and receive all the data stored in this account. After that, it will go through a loop, the number of loop is the current sensors in use. This loop will go through the returned object and find out each variable contained in the object. Then the variables will assign to the List in the UI.

2. Display data to user

There are three dashboards in the UI frame. Each of the dashboard refers to a type of soil data. They are EC, temperature and VWC. All the sensors will use same dashboards to display data. There will be a tab click event, when user click the tab, UI will set the value for each dashboard according to the selected tab.

3. Receive input from user

Control panel is right under the display panel. User may choose the mode to use for the sensor using a combo box, this box contains three choices, time, VWC and both. Each refers to a type of mode. For VWC mode, there will be two sliders, one of them refers to the start value of VWC mode, while the other one refers to the close value of VWC mode. For Time mode, User may choose the start date and enter the period in an input box. There is no error check for the input right now.

4. Load/save settings in local files When user open the UI. Load function will run first. It will read each line in a specific file called log.txt, each line stores the mode settings for a sensor. They will be assigned to the Listed variables in the UI. Each time user clicks the apply button. Save function will run. It will over write the log.txt file and write mode settings to the file.

5. Send data to Adafruit.io

Similar to the receive part, it will pull a request to the Adafruit.io, this time it will include the name of feed in Adafruit.io and their changed values. They will be add at the last of the pulled url as a string. Then it will check the status code to make sure data has been sent successfully.

6. Update data every 15 minutes

To implement this part, I use the multi-threading. By creating two threads, one of them will run save function and create the frame for UI. The other one will be a while loop, sleep for 15minutes, and then update the data by pulling a request to Adafruit.io and call update data function. Timer object may also work, but I didnt try it.

7. Add/remove a sensor

There is a tab panel on the top of the frame. User can change the sensor data to view by clicking different tab. There will also be two buttons beside the tab panel. Add button and remove button. If users click add button, a new sensor will be added at the last of the tab. If user click the remove button, then the selected sensor will be removed.

4.2 Left To Do

The current UI can only handle the data of one sensor. I need to change the type of the variables. Include soil data and mode settings. All those variables will be a list contains each type of data. Type of variables will not be the only part to be revised. The update function shouldnt update the data only one time, it should go through a loop according to the number of the sensors it will handle.

While I can set the variables name for testing, user wont do that. There should be a way for users to select which variable will point to which feed in the Adafruit.io, or they may need to enter the name of feeds with a specific format. Each Adafruit account have its unique AIO-key, UI should be able to receive the AIO-key of the user account as well.

I havent set up the error check function yet. So I will add it later. If error happens, an error message should show in the frame to notice users.

4.3 Problems & Possible Solutions

Every part of the UI is not difficult. The only problem that impeded my progress is the functionality of add/remove the sensor. Although there are many sample codes online, I just cant add a remove button as a x in a tab to remove the selected tab.

To solve this problem, I add a remove button under the tab panel. Create a click event for it, remove the tab that is current using. Its a little dumb, but it works.

4.4 Relevant & Interesting Information

The UI design is interesting. The function I called most often is setBounds(set the size and position of a component). The second is the doClick(click a button). Refresh button in the frame almost does everything, include pull request, update data, and repaint the frame.

5 Conclusion

While progress for the soil sensor and hub devices started off in a bit of a rough patch, at the time of this writing, solid progress has been made on both devices. The soil moisture sensor is reading data and transmitting it to the hub reliably, and the hub has been able to communicate via ethernet and MQTT to publish the data to an Adafruit.io interface on the internet. The UI is in a functional state, and interacts with Adafruit.io, which will make recording data and sending instructions from the application simple. New features will likely be added to it as we get a bit further into the project as well.

All things consider, the project is moving along nicely and seems to be on course for a beta-level release by the end of the term.