Oregon State University
College of Engineering

CS463 Capstone Progress Report

May 7, 2018- Spring Term

Developing an Internet of Things Irrigation Valve

Prepared and issued by

Group 35

Christopher Carlsen
Yizheng Wang
Peter Dorich

Abstract

This document is the Midterm Progress Report for the Srping 2018 term of the CS463 Capstone course. Discussed in this document is our group's experience and progress with the project since the last progress report. It describes the general purpose of the project, includes progress up to now and explains any remaining work to be done, as well as any possible problems.

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

For clarity purposes, I have decided to break apart the current status of my segment of the project. I will discuss its status by comparison to the goals outlined in the Requirements Document. The first requirement for this device was the ability to gather soil-moisture data. This was the original focus of my segment of the project. I started off last term spending the first few weeks getting the data reads from the Decagon soil moisture sensor working. This feature uses the SDI12 library as an interface between the sensor and the Feather M0 device. After the first few weeks, this feature was fully implemented and stable. Interestingly enough this term, one of the sensor devices that I was using broke and was sending me bad data. I had at first thought that I had broken something in the code, but after a good couple of hours reading through the code and checking commit logs for any changes that might of broken it, I decided to try a new device... which worked immediately. The implementation of this feature is complete and stable as of last term.

The second primary piece of the valve-controller was to get it into reliable communication with the central hub device. I had imagined that this would initially take the vast majority of our time getting setup, as network (especially wirelessly), can definitely be a difficult feature to implement in a project. After some consideration during the first time, and talking with our client, it was decided that long range (LoRa) radio would be a good fit for our needs due to it's ability to travel distances, and its mostly clear bandwidth. Originally we had planned to just pass our raw data values and have the devices just interpret those as received. However, upon request from the client, the schema for data transform was changed over to include a delimited-string identifier tag for each piece of information. The OpenSoundControl library is used to package this information into bundles before sending. This feature is fully functional as of last term.

Listed as the fourth functional requirement for my segment was the device's ability to store a fail-safe set of data. This was implemented rather late last term, but was pretty simplistic to include. As the Feather M0 device does not

have EEPROM storage like many other micro-controller boards, a library for writing to Flash memory was used as an alternative. Being a non-volatile form of storage, this allows the device to be powered down completely, and still store the most recently received set of good instructions. This functionality was implemented late last term, and ensures that if it loses connectivity to the central hub or powers down, that it would be able to continue expected functionality.

The third listed requirement for the valve-controller states that the device should be able to open and close the attached valve, using a set of conditional parameters passed to it by the hub (from the user). This part of the project has been, quite unexpectedly, the most frustrating part of the project for myself. I spent a about two weeks last term diagramming and thinking about the electrical connections of the attached Feather Relay Shields to the valve-controller that we are using. This part of this requirement was a little challenging for me as I have no electrical engineering experience, but it worked out alright without me blowing up anything or burning down the lab. However, the real challenge has been working out the logic scheme to reliably receive instructions and pass data back and forth between the valve-controllers and the hub. The frustration and issues with this segment of the project stem from two main issues. The first is that the Feather M0, as a micro-controller, is a single-core processor. Because of this, multi-threading is not an option, which made communication difficult, and running multiple jobs difficult. The second issue was that the client originally wanted the device to sleep in between the different segments of work (data-read and send/open valve/close valve). However, as the Feather M0 is one of the newer devices in the micro-controller field, many of the libraries did not yet support it, and more specifically the devices core code did not yet correctly support a true sleep yet either. Because of these issues, almost the entirety of my time spent on Capstone this term, and much of the end of last term, has been spent trying to develop this feature.

After some talk with the client about my efforts trying to get the device to sleep, as well as the issues regarding multi-threading and multiple jobs, it was decided that the sleeping feature would be put on the back burner for now. Along with that, to ease the issue of timing (and a lack of multi-threading ability), it was decide that the current iteration of the valve-controller will take its reads and communicate with the hub on a fixed-increment timer. The requirements document states that it must do this every fifteen minutes, but as of right now it defaults to every 5 minutes. This can be adjusted by the user at any time however. This last requirement for the valve-controller part of the project is, at the time of this writing, is fully functional. That being said, there is a little work to do still on it, as it is currently about 90% feature complete The remaining bit will be described in the following section.

2.2 Left To Do

While at this point the valve-controller is fully functional, there are still a few features that the client requested that remain to be implemented. The first main goal I mentioned earlier, is to try and add a sleep functionality to the device. The Feather M0 did not have a reliable way to put the device into a sleep state, and then be woken back up. There was a low-power state that, after a very long weekend testing in the lab, was able to be used. This mode seems to slow the CPUs cycle time down dramatically, and also disables many of its GPIO pins and such. The sleep feature was put on the back-burner in favor of finishing the main functionalities of the device that were originally laid out, but I would like to try and implement this feature if possible.

The other feature that I would like to implement is a way to connect the Feather M0 to the solar-panel that is built into the case of the valve-controller. This would allow the device to self-sustain for longer periods of time, or potentially indefinitely. The device requires at most a 6-volt supply, but the solar-panel can put out up 20 volts. Attaching to the solar panel would require a regulator of some kind to keep it from frying the device. I had originally built out a smaller

solar panel that provided up to 6 volts on its own. However, the challenge of mounting this to the external enclosure made it a little infeasible.

2.3 Problems & Possible Solutions

The biggest problem that I faced this term was the implementation of a reliable methodology for sleeping/idling the device, and having it check in at an arbitrary check-in interval. The solution for this was to use a real-time clock (RTC) device to more accurately track time on the device. This device now controls when the logic fires to take a sample, upload it and check for new instructions. Additionally, the use of the RTC enables the ability to (possibly) sleep and then wake the device using the alarms and GPIO interrupt signals.

That being said, at this time, there are no longer any remaining major problems in the valve-controller segment of the project, and everything should hopefully be good for expo!

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 this point in the project, can successfully transmit instructions to the valve, and also transmit soil data to a user interface.

3.2 Overview of Term Progress

By March 2018, the wireless hub portion of the project was in a state where it could transmitt data to the UI from the valve. It couldn't, at that point, send instructions to the valve. Since then, the valve is now capable of sending instructions to the valve, which are then executed. While it does this, it also can keep updating the UI with soil data.

Bidirectional communication was implemented at the end of the last term, and the beginning of spring term. Bidirectional communication now allows the system to transmit instructions to the valve, which then can execute them. This functionality was difficult to implement due to timing issues, but once those were solved we witness successful functionality.

During the last week of the development period we sat down with our client Chet Udell to show him a full demo of our working system. He used the User Interface as we instructed him, and we watched our Irrigation valve open and close based on his inputs. The demo went very well, and was proof of our working system.

One of the last main things to be added was the Open Sound Control Library implementation. 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.

The format of the string contains a lot of information, which is what makes OSC so powerful. For instance, the string above says about six things about the data. ”LOOM” is the name of the overarching series of OPEnS projects. ”IrrVal” is our specific project name. The ”0” is the unique i.d. number for the valve. ”VWC” is the type of data being listed, in this case it is volumetric water content. The ”f” is the type of the data, in this case, float. Finally the number is the value to be converted to float when it reaches the destination.

Since implementing this system, and adding bidirectional communication, the hub has been in a complete state in accordance with the functional requirements of the project.

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

The layout of interface doesn’t change a lot since last term. However, based on users’ feedback. I add menu and some features. This include send notification email to users and change the current using email address. 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.

8. Notice users appropriately

User can change the email address they are using. And if hub or sensors shut down with some issues, the user interface will send an email to user to notice that. To prevent it from sending same warning message many times, the user interface can only send same message once and will be reset when user restart it.

4.2 Left To Do

There aren't many things left to do for the user interface. I think the mainly thing I will focus on will be reformating the code and add comments for user to read it.

4.3 Problems & Possible Solutions

Every thing moved fluently since last term.

5 Conclusion

This term we finally got the project in a completed state. Work is still being done to implement stretch goals and to optimize the system, but at this stage all required functionality is now usable. The next step is to deploy our system for testing at LB farms. In addition, we are fitting all of the valve electronics into the existing P&R valve. To do this, holes will need to be drilled into the metal housing in order for wires and the antenna to leave the device. The valve also has a built-in solar panel that can potentially charge our batteries. The panel is capable of ouputting 20 volts, and the battery only needs 6 to charge. That being said, a regulator must be installed that can control the 20 Volt output. This feature will hopefully be implemented by the end of the term.