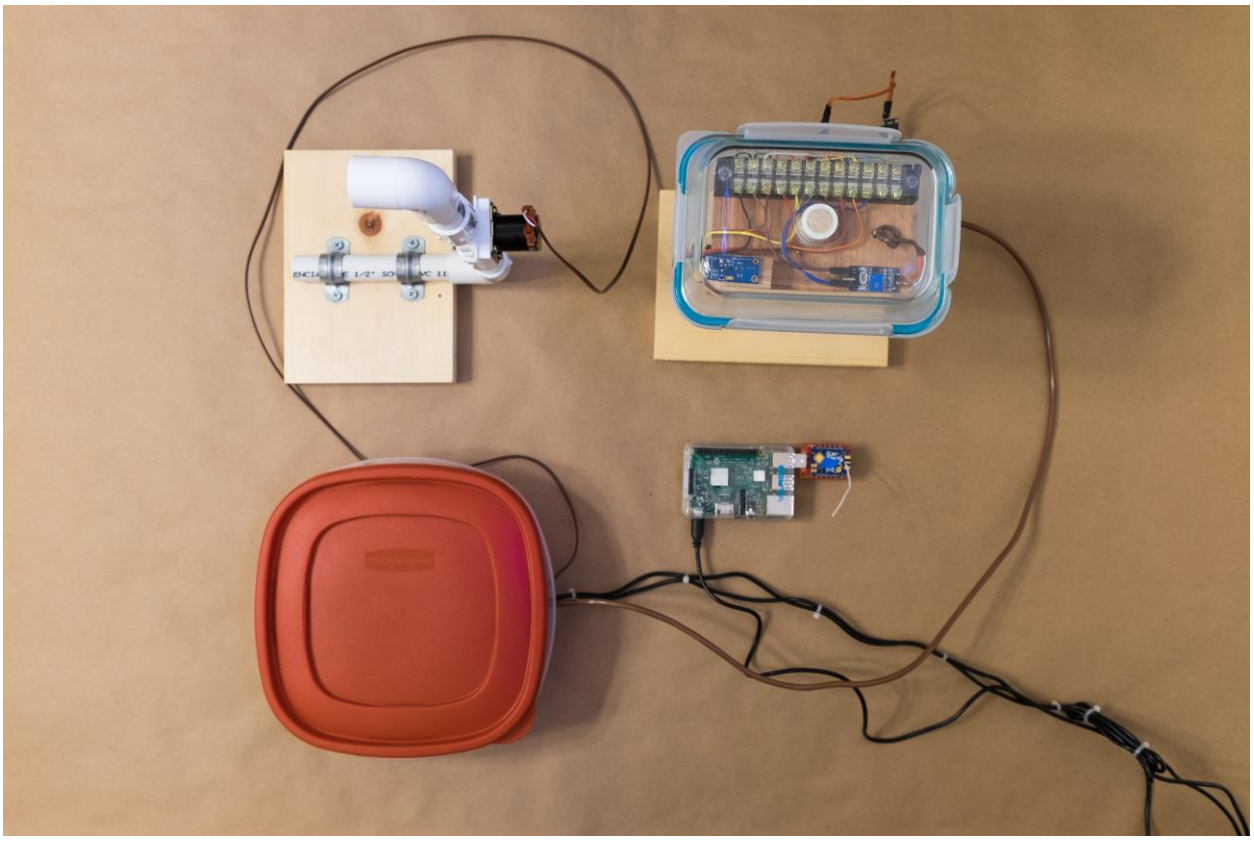
# Project ALIN

Automatic Lawn Irrigation Network



December 2016 - Thomas Dye, John Walter

CSS 427 - Embedded Systems - Dr. Yang Peng

University of Washington

## Introduction

Our project was to create a “smart” irrigation system by using sensor analysis to control water flow to designated zones to only water plants when necessary. The challenge we faced is that different parts of a yard have different watering requirements. We needed to sense these differences and independently schedule watering for each unique locale.

This project was made possible through cooperation with UWB staff member Rafael Machado De L Silva. He has a similar project he wants created that has a high degree of feature overlap. Throughout the project, Rafael operated in the capacity of an advisor and sponsor, supplying us with both advice and the majority of the hardware used.

## System Overview

Basic system:

- A single command station houses all schedule logic and provides a user interface.
- Users can configure the system and do on-demand actions through a web interface, connecting over WiFi.
- The command station talks to some arbitrary number of controllers, remotely located, using XBee radios.
- Each controller gathers sensor data from any number of irrigation zones and reports that to the command station.
- The controller also opens valves to control irrigation when the command station sends an irrigation schedule.

The command station creates or updates an irrigation plan and sends start and stop times to the controller. At the prescribed time, the controller opens and closes valves connected to piping to deliver water to zones.

The command station’s evaluation of a zone’s irrigation schedule can be augmented by user preference and the number (and type) of sensors in use. Our minimum goal for this iteration of the project was to collect soil moisture information as use that for evaluation. Sensor collection for temperature, humidity, and brightness have also been implemented. Online weather data analysis continues to be a goal for a future version of this project.

## Hardware

Project ALIN is comprised of two embedded devices networked together with XBee 900Mhz radios, a single sensor pod, and an electric water valve, both of which are associated with a zone. The command station is powered by a Raspberry Pi 3 and the controller is Arduino-based.

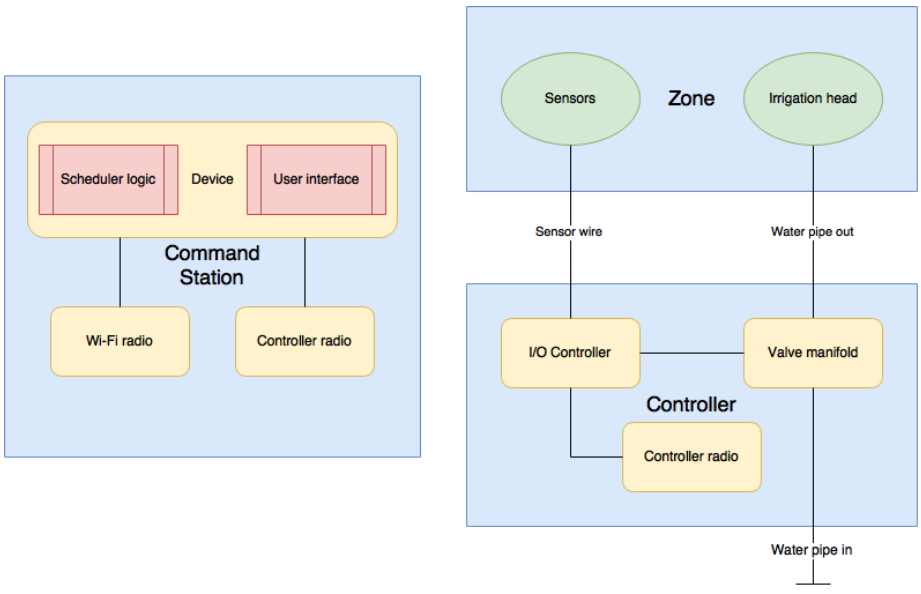



Figure 1: System Layout

The command station is a low power, internet connected embedded device.  It has three responsibilities:

- Receive sensor data from controllers for all zones
- Evaluate sensor data and send out irrigation schedules when necessary
- Provide direct user interaction to configure and control the system

Based off of the scheduling algorithm, irrigation is automatic when conditions warrant it; however, manual control is also possible.

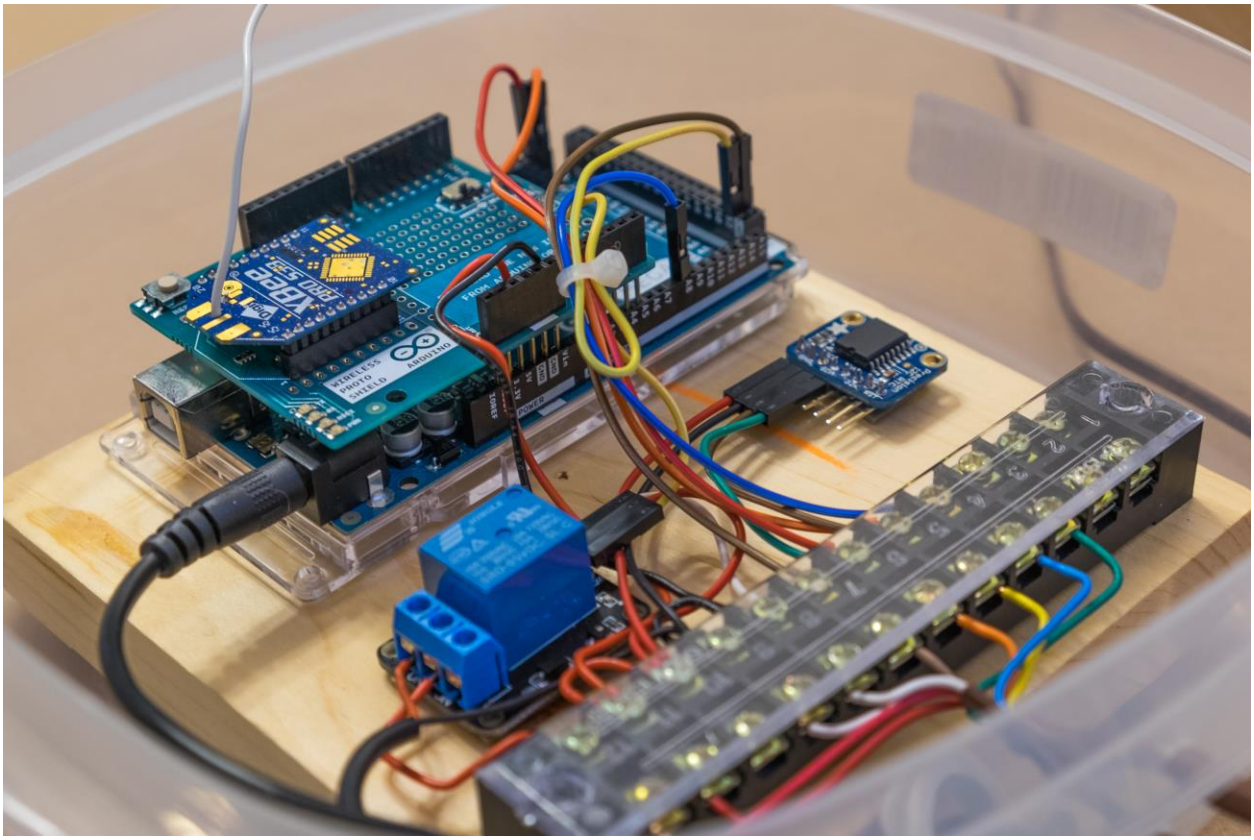


Figure 2:  Command Station

The controller is designed to be wall mounted outdoors.  It has three responsibilities:

- Transmits sensor data both periodically and on demand
- Receives scheduled irrigation events
- And opens, closes, or toggles the valve state for a zone.

Automatic valve control is handled through a real-time clock powered scheduler, but all functionality has manual controls through a web interface provided by the command station.

One controller handles multiple zones where relay controlled 12VDC electric valves handle water flow to the irrigation heads.  Since the user defines how zones are configured, the granularity of control with irrigation is dependent on how fancy you get with your plumbing.

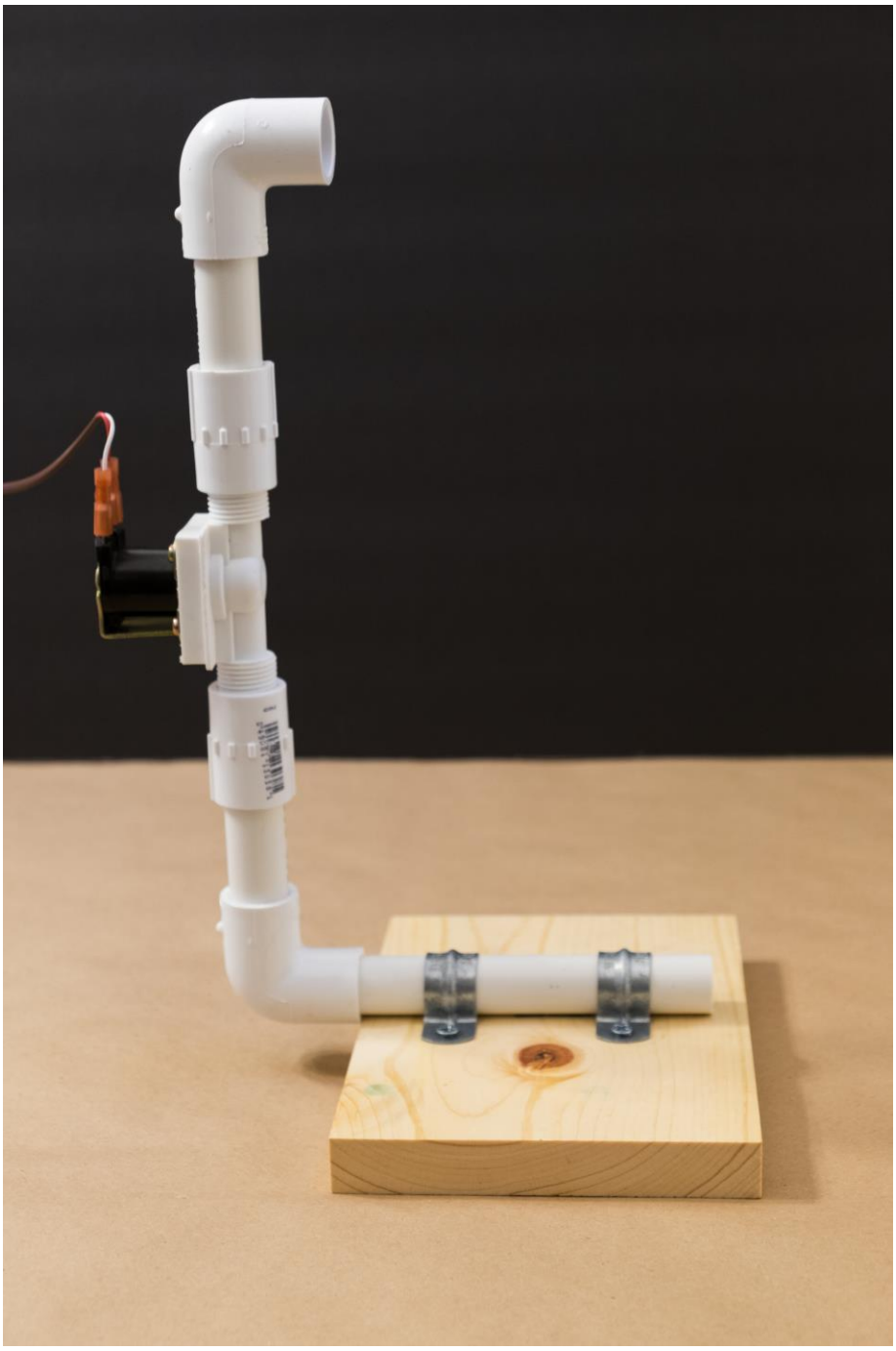


Figure 3: Valve Manifold

Sensor pods are designed to be remotely located inside the zone you're watering. Each contains four sensors, collecting:

- Temperature
- Humidity
- Light
- Moisture

Sensor pods have to be relatively weatherproof, but also need to be ventilated for accurate temperature and humidity readings. A glass lid is used to enclose the unit and provide consistent readings to the light sensor that do not diminish over time through dirt and UV discoloration.

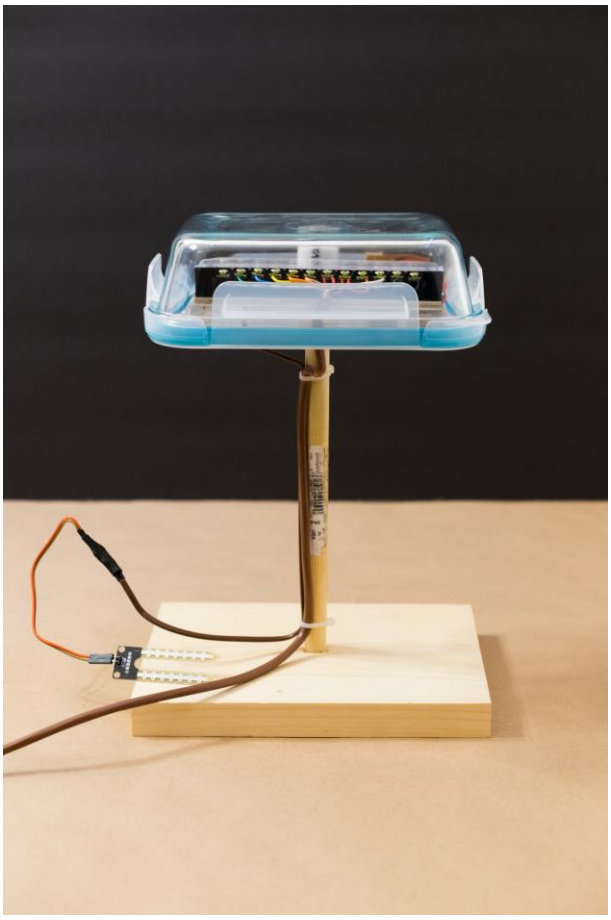


Figure 4:  Sensor Pod - Closed

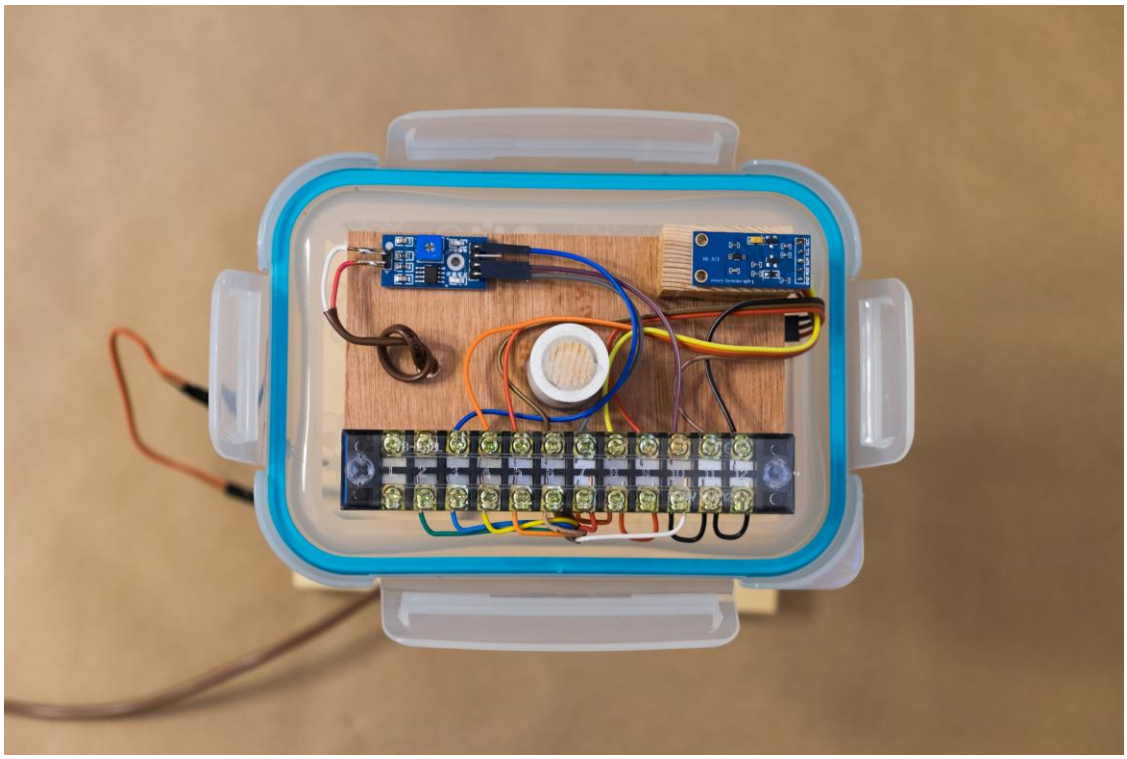


Figure 5:  Sensor Pod - Open

## Scheduling Algorithm

The watering schedule is affected by the following conditions:

- Time of day
- Amount of sunlight being received
- Temperature and humidity
- Soil conditions

These conditions are independently interpreted in the following way:

- Sunny days cause more evaporation than cloudy days
- Hotter days cause much more evaporation than cooler days
- Humid days prevent evaporation, but dry days promote it
- Dry soil is undesirable

The command station will determine if a zone is irrigated depending on combinations of readings taken by pod sensor inputs. Each sensor type has readings that can be distinguished as a '0' or '1' value, where a user-defined threshold separates the two.

| | **0** | **1** |
|---|---|---|
| **Temperature (A)** | Not Hot | Hot |
| **Humidity (B)** | Not Humid | Humid |
| **Light (C)** | Not Sunny | Sunny |
| **Soil Moisture (D)** | Not Moist | Moist |

Table 1: Sensor Threshold Categories

For any combination of the four inputs, there is one decision output: irrigate (1), or do not irrigate (0). To determine the irrigation algorithm, each combination of conditions was evaluated to determine if watering was necessary. From that, the following truth table was formed:

| **A** | **B** | **C** | **D** | **Output X** |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Table 2: Algorithm Truth Table

This combination of all inputs and outputs can be simplified down to the following Boolean equation:

$$X = \sim B \sim D + C \sim D + A \sim D + A \sim BC$$

Thus, the command station will recommend irrigation any time when the following conditions persist:

- Dry air and dry soil, or
- Sunny day and dry soil, or
- Hot air and dry soil, or
- Hot and dry air, and sunny.

As project ALIN is meant to operate as a demonstration system, it's not possible to collect data throughout the day as it normally would for a production system. The final decision to irrigate is revised in the following way:

- Evaluate the average of the last four sensor readings
- Compare those averages against thresholds for each sensor type
- Schedule irrigation if the recommended conditions exist, timed to be one minute in the future for a watering duration of one minute.

## Sensor and Data Specification

The following sensors are in use:

| Sensor | Model | Format | Data | Notes |
|---|---|---|---|---|
| Time | DS3231 | I2C | S:M:H:W:D:M:Y | Real Time Clock |
| Temperature | DHT11 | Digital | Integer Value | Measurements in centigrade scaling |
| Humidity | DHT11 | Digital | Integer Value | Percentage ratio of dew point to temperature |
| Light | BH1750 | I2C | Integer Value | Measurements of intensity in lux |
| Moisture | YL38 | Analog | Integer Value | Resistance between 0 and 1023 |

Table 3: Sensor Overview

Message payloads are delivered inside XBee frames as comma delimited values. We devised a structured communication protocol where the payload begins with a command code and then supplies additional parameters as necessary.

| Code | Code, HEX | Code, DEC | Additional Payload | Notes |
|---|---|---|---|---|
| C_ACK | 0x00 | 0 | | Response sent when a command is received but not recognized. |
| C_SUCCESS | 0x01 | 1 | | Response sent when a command is received, understood, and is executed. |
| C_FAILURE | 0x02 | 2 | | Response sent when a command is received, understood, but does not have valid parameters. |
| C_GET_VALVE_STATE | 0x10 | 16 | Z | Request the valve state for a zone Z. Generates a C_VALVE_DATA response. |
| C_SET_OPEN_VALVE | 0x11 | 17 | Z | Instruct the valve open for a zone Z. Generates a C_SUCCESS response. |
| C_SET_CLOSE_VALVE | 0x12 | 18 | Z | Instruct the valve close for a zone Z. Generates a C_SUCCESS response. |
| C_SET_TOGGLE_VALVE | 0x13 | 19 | Z | Instruct the valve position (open/close) to toggle for a zone Z. Generates a C_SUCCESS response. |
| C_SET_TIME | 0x14 | 20 | S,M,H,W,D,M,Y | Instruct the RTC to update system time. Parameters are S:seconds (0-59), M:minutes (0-59), H:hours (0-23), W:weekday (1-7), D:day (1-31), M:month (1-12), Y:year (0-99). Generates a C_SUCCESS response. |
| C_GET_TIME | 0x15 | 21 | | Request the time from a device. Generates a C_TIME_DATA response. |
| C_GET_ZONE_SENSORS | 0x16 | 22 | Z | Request sensor data for a zone Z. Generates a C_SENSOR_DATA response. |
| C_GET_ALL_SENSORS | 0x17 | 23 | | Request sensor data for all zones. Generates a C_SENSOR_DATA response for each defined zone. |
| C_GET_SCHEDULE | 0x18 | 24 | Z | Request the irrigation schedule for a zone Z. Generates a C_SET_SCHEDULE response. |
| C_SET_SCHEDULE | 0x19 | 25 | Z,H,M,D | Instruct the device to begin irrigation at a specified time and duration for a zone Z. Parameters are H:hour (0-23), M:minute (0-59), D:duration(in minutes). Generates a C_SUCCESS response. |
| C_VALVE_DATA | 0x30 | 48 | Z,S | Data for valve state for a zone Z. State S = 0 is closed. State S = 1 is open. |
| C_TIME_DATA | 0x31 | 49 | S,M,H,W,D,M,Y | Data for the time on a device. Parameters are S:seconds (0-59), M:minutes (0-59), H:hours (0-23), W:weekday (1-7), D:day (1-31), M:month (1-12), Y:year (0-99). |
| C_SENSOR_DATA | 0x32 | 50 | Z,L,T,H,M | Data for sensor readings for a zone Z. Parameters are L:light (lux), T:temp (celcius), H:humidity (percentage), M:moisture (level) |

Table 4: Radio Data Payload Specification

All payload information is transmitted as character arrays as literal values, i.e., the int 1023 is four bytes long, sent as characters ‘1’ ‘0’ ‘2’ ‘3’.  For example, the payload 50,1,138,22,45,1023 describes:

- Sensor data for zone 1
- Brightness is 138 lux, temperature is 22 C, humidity is 45%, and the soil is completely dry.

This system is designed to primarily be autonomous, but threshold and user preferences are configured via a web interface provided by the command station.

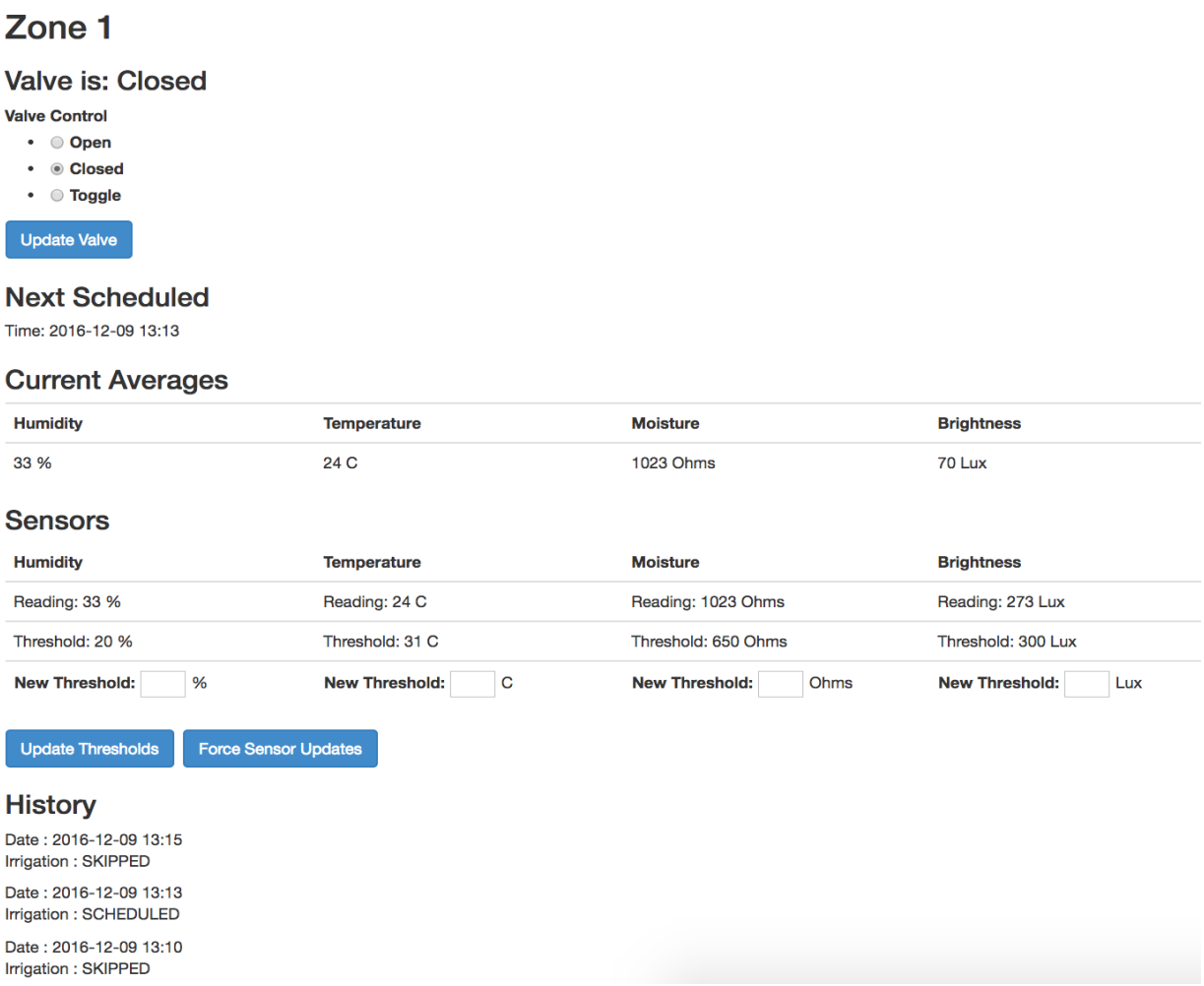



Figure 6:  Command Station Web Interface

The following communication processing is implemented on the command station:

| Command | Trigger | Sends | Expects |
|---|---|---|---|
| C_GET_VALVE_STATE | Sent on 10 second interval | 16,1 | 48,1,0 |
| C_SET_OPEN_VALVE | On demand | 17,1 | 1 |
| C_SET_CLOSE_VALVE | On demand | 18,1 | 1 |
| C_SET_TOGGLE_VALVE | On demand | 19,1 | 1 |
| C_GET_ZONE_SENSORS | On demand | 22,1 | 50,1,273,24,33,1023 |
| C_SET_SCHEDULE | Automatic, by scheduling algorithm | 25,1,13,13,1 | 1 |
| C_SENSOR_DATA | Periodically received from controller | - | 50,1,273,24,33,1023 |

Table 5:  Command Station Radio Communication Sample

Note: Expected data is shown interpreted in Figure 6.

The controller is designed to operate as a headless unit, however non-interactive serial access is available over analog pins 10 and 11 for RX and TX for diagnostic output. Radio communication and controller actions are logged to the serial console. Independent sequence counters exist for sent and received data to aid identification of transmission failures.

The following communication is implemented on the controller:

| Command | Trigger | Receives | Sends |
|---|---|---|---|
| C_ACK | Received unrecognized code | 99 | 0 |
| C_SUCCESS | Received with correct parameters | 19,1 | 1 |
| C_FAILURE | Received with bad parameters | 19,0 | 2 |
| C_GET_VALVE_STATE | Received periodically | 16,1 | 1 |
| C_SET_OPEN_VALVE | Received periodically | 17,1 | 1 |
| C_SET_CLOSE_VALVE | Received periodically | 18,1 | 1 |
| C_SET_TOGGLE_VALVE | Received periodically | 19,1 | 1 |
| C_GET_ZONE_SENSORS | Received periodically | 22,1 | 50,1,273,24,33,1023 |
| C_GET_ALL_SENSORS | Received periodically | 23 | 50,1,273,24,33,1023 |
| C_SET_SCHEDULE | Received periodically | 25,1,13,13,1 | 1 |
| C_SENSOR_DATA | Sent on 30 second interval, or when requested by the command station | 22,1 or 23 | 50,1,273,24,33,1023 |

Table 6: Controller Radio Communication Sample

Controller radio communication sample notes:

- The command code ‘99’ is undefined.
- A C_FAILURE is generated because zone 0 is undefined.
- The C_GET_ALL_SENSORS generates the same response because only one sensor zone is defined. In the event of there being multiple zones, multiple C_SENSOR_DATA transmissions would be sent in rapid succession.

Both the command station and the controller use the retransmission protocol built into the XBee stack. The controller will additionally attempt up to five additional retransmissions (beyond the original transmission) with acknowledgement periods up to five seconds each before giving up. For any retransmission, the last message is resent and the internal send sequence counter is not incremented.

Serial console example:

```
Sent seq: 0, data: 50,1,273,24,33,1023
Error on seq: 0, the remote XBee did not receive our packet.
Sent seq: 0, data: 50,1,273,24,33,1023
Error on seq: 0, the remote XBee did not receive our packet.
Sent seq: 0, data: 50,1,273,24,33,1023
Error on seq: 0, the remote XBee did not receive our packet.
Sent seq: 0, data: 50,1,273,24,33,1023
Error on seq: 0, the remote XBee did not receive our packet.
Sent seq: 0, data: 50,1,273,24,33,1023
Error on seq: 0, the remote XBee did not receive our packet.
Sent seq: 0, data: 50,1,273,24,33,1023
Error on seq: 0, the remote XBee did not receive our packet.
Error on seq: 0, maximum number of retransmissions exceeded.
Received seq: 0, data: 16,1
Sent seq: 1, data: 48,1,0
```

# Exception Report

## Command Station

The web server uses a file-based data sharing scheme with the command station, such that the web server writes its data to files that the command station reads (and only reads) from, and likewise the command station writes its data to files that the web server reads (and only reads) from. This is not a good design for a couple reasons, not the least of which being that the web server cannot be moved to the cloud using this scheme. We went with this design because we did not know how to better achieve our goals, and it wasn't until late into the implementation that we learned that the proper design would be to have the web server alone read and write to a database while providing the command station an interface to access the database through the web server using GET and POST requests. For the purpose of our demonstration, our current file-based design was sufficient. However for ALIN 2.0, we intend to gut the command station and web server of file-based data sharing and implement the above mentioned GET/POST design.

## Controller

The controller logs the occasional bad packet received from the command station, but so far this hasn't resulted in any loss of functionality (the commands still process despite the "ZB_RX_RESPONSE: format not expected" error). I would like to eventually eliminate string processing for building data payloads, as the original implementation I used with character arrays on the stack was causing system crashes for unknown reasons (which were not related to off-by-one errors).

One error that presented itself during testing before presentation is that the discharging solenoid in the valve was causing sporadic controller resets through excessive electrical noise that was being generated. This was also apparent on the monitor connected to the computer which would distort. We might have damaged it through overheating as this was not always a problem, but I would like to reinvestigate sharing a common ground for all components, or at least, electrically isolating it and preventing feedback to the controller.

# Team Assignments and Responsibilities

In the beginning of the project, both members worked together at drafting the design and researching the different options we had to achieve our goal implementing this system. We worked together when we were trying to establish communication between two devices over an XBee radio connection. After the point of establishing connectivity, however, we both split off in separate directions. Thomas was solely responsible for putting together the hardware and housing involving the sensors and the Arduino-based controller side of the system, while John was responsible for the Raspberry Pi based command station and web server. For the most part, these two roles were solely completed by each respective team member. There was a point in the middle of the project where we worked together again to devise the scheduling algorithm, and again nearing the end of the project when we integrated our parts together and performed testing and validation. Distribution of work was varied, but equal.

# Appendix

## XBee Radio Configuration

900 MHz Series 2 Radio 1: (blue tape, command station)

- Product Family: XBP9B-DM
- Function set: XBee PRO 900HP 200K
- Firmware: 071
- Network ID/PAN: 1337
- Standard Router
- Serial High: 0013A200
- Serial Low: 40E3CD0F
- API Mode 1 (without escapes)

900 MHz Series 2 Radio 2: (controller)

- Product Family: XBP9B-DM
- Function set: XBee PRO 900HP 200K
- Firmware: 071
- Network ID/PAN: 1337
- Non-Routing Module
- Serial High: 0013A200
- Serial Low: 40E3CD1E
- API Mode 2 (with escapes)

Note: The different API modes are required as the Python XBee library only works with mode 1, and the Arduino XBee library only works with mode 2. Escaping seems to only be important with regard to the library in use and the mode mismatch does not inhibit communication between devices.

## Parts List

| Source | Item | Quantity | Unit Price | Total Price | Purpose |
|---|---|---|---|---|---|
| School | Arduino Mega 2560 | 1 | $45.95 | $45.95 | Controller logic board |
| | Wireless Shield | 1 | $28.30 | $28.30 | Controller XBee interface |
| | XBee Explorer | 1 | $24.95 | $24.95 | Command station XBee interface |
| | XBee-Pro 900 S3B | 2 | $59.95 | $119.90 | XBee radios |
| | Raspberry Pi 3 | 1 | $39.95 | $39.95 | Command station logic board |
| | Raspberry Pi Case | 1 | $5.00 | $5.00 | Command station enclosure |
| | Raspberry Pi Power | 1 | $7.95 | $7.95 | Command station power source |
| | 16gb Card | 1 | $14.95 | $14.95 | Command station storage device |
| | Sensor Modules | 1 | $21.00 | $21.00 | Controller sensors |
| | Water Valve | 1 | $6.95 | $6.95 | Controller valve manifold |
| | Total | | | $314.90 | |
| Home Depot | 1x6x6 Wood Board | 1 | $5.52 | $5.52 | Controller base construction material |
| | 1/2" 2' PVC Pipe | 1 | $1.22 | $1.22 | Sensor tower construction material |
| | 1/2" Wood Dowel | 1 | $1.77 | $1.77 | Sensor tower construction material |
| | 2 Conductor Wire | 6 | $0.21 | $1.26 | General electrical |
| | 7 Conductor Wire | 10 | $0.73 | $7.30 | General electrical |
| | #8 1/2" Wood Screws | 1 | $0.98 | $0.98 | General construction material |
| | 1/2" F 90 Deg Pipe | 2 | $0.28 | $0.56 | Plumbing |
| | 1/2" F Adapter | 2 | $0.53 | $1.06 | Plumbing |
| | 22-18AWG .25 Spade | 1 | $2.99 | $2.99 | General electrical |
| | 1/2" Pipe Straps | 1 | $2.14 | $2.14 | Plumbing |
| | 3/32" Heat Shrink | 1 | $1.99 | $1.99 | General electrical |
| | Tax | | | $2.55 | |
| | Total | | | $29.34 | |
| Fred Meyer | Rubbermaid | 1 | $3.24 | $3.24 | Controller enclosure |
| | Food storage | 1 | $6.99 | $6.99 | Sensor tower glass enclosure |
| | Tax | | | $1.00 | |
| | Total | | | $11.23 | |
| Amazon | 120pcs Jumper Wire | 1 | $8.68 | $8.68 | General electrical |
| | 5pc Terminal Block | 1 | $7.99 | $7.99 | General electrical |
| | 12VDC Power Supply | 1 | $6.99 | $6.99 | General electrical |
| | Total | | | $23.66 | |
| Adafruit | DS3231 RTC | 1 | $13.95 | $13.95 | Controller sensor |
| | Shipping | | | $9.65 | |
| | Total | | | $23.60 | |
| **Project** | **Grand total** | | | **$402.73** | |