The Implementation of Passive RFID Tags to Sample Volumetric Water Content for an Autonomous Irrigation System

by
Matthew Guo

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Electrical & Computer Engineering and Computer Science
(Honors Scholar)

Presented August 28, 2020
Commencement June 2021

AN ABSTRACT OF THE THESIS OF

Matthew Guo for the degree of Honors Baccalaureate of Science in Electrical & Computer Engineering and Computer Science presented on August 28, 2020. Title: The Implementation of Passive RFID Tags to Sample Volumetric Water Content for an Autonomous Irrigation System.

Abstract approved:_____

Chet Udell

The agricultural industry in the Western United States continues to possess issues of overwatering, accounting for over 80% of consumed water per year in the country. While technologies such as satellite imaging and sensor networks have slowly improved irrigation Best Management Practices to reduce superfluous water consumption, limitations associated with these technological advancements include complex designs, high installation and equipment costs, and intensive maintenance labor. A proof-of-concept autonomous irrigation system installed at Peoria Gardens aims to alleviate these shortcomings through the use of cheap passive RFID tags from SmarTrac, which utilizes RFMicron's environmental sensing technology to sample moisture value of its surroundings. These tags, calibrated using Adafruit Capacitive Soil Sensors, are placed in the soil of pansy crops at Peoria Gardens. A SparkFun Simultaneous RFID Reader is used to read moisture values sensed by the passive RFID tags, which then communicates the readings to an onboard Feather M0 WiFi via UART for microprocessing. The firmware for the Feather M0 contains a state machine that uses the communicated moisture values, converts it to a subjective Likert Scale Peoria Gardens uses for their crops to determine the activation of an onboard power relay that controls the irrigation switch for the irrigation boom.

Key Words: Irrigation, SmarTrac, RFID, SparkFun Simultaneous RFID Reader, Feather M0 WiFi, Peoria Gardens, Likert Scale, Adafruit Capacitive Soil Sensors, Microprocessor

Corresponding e-mail address: guom@oregonstate.edu

The Implementation of Passive RFID Tags to Sample Volumetric Water Content for an Autonomous Irrigation System

by
Matthew Guo

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Electrical & Computer Engineering and Computer Science
(Honors Scholar)

Presented August 28, 2020
Commencement June 2021

Honors Baccalaureate of Science in Electrical & Computer Engineering and Computer Science project of Matthew Guo presented on August 28, 2020.

APPROVED:

_____

Chet Udell, Mentor, representing Department of Biological and Ecological Engineering

_____

Alec Kowalewski, Committee Member, representing College of Agricultural Sciences

_____

Nico Ardans, Committee Member, representing Peoria Gardens

_____

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, Honors College. My signature below authorizes release of my project to any reader upon request.

_____

Matthew Guo, Author

Introduction

The denotation of irrigation is an artificial application of water to crops, simulating that of nature, in order to enhance the crops' growth rates. This concept, and its associating efficiency, continues to be a large research interest in the agriculture industry. Regarded as one of the more prominent causes of plant and crop issues, the issue of overwatering is common among many gardens, farms, and private lawns. In recent years, agriculture overwatering has been a ubiquitous issue, accounting for the use of over 80% of available consumable water in the United States alone [1]. These overwatering practices pushes the United States into second place as the country with the water waste, with 216 trillion gallons wasted per year, right behind China's 362 trillion gallons per year [2]. Since 1998, however, advancements in irrigation Best Management Practices (BMPs) have resulted in overall decreases in total applied irrigation water, declining from approximately 76 million of acre-feet to 72 million of acre-feet of irrigation water, despite about a 7.5% increase in irrigated acreage in the western United States [3].

A large part of increased BMP within the United States is a rapid growth and incorporation of precision technologies to irrigation practices, such as satellite data, sensor networks, data analytics, etc. By aiding commercial gardeners and farmers with greater water precision, these precision technologies allow an increased throughput of crops while simultaneously minimizing input costs. However, these added efficiencies to the irrigation practices do not come without drawbacks, such as cost and financial considerations, in which 25% of crop growers in the United States stated that they could not justify the added installation costs [4], with 46% of farmers expressing disinterest in the willingness to pay for the satellite imaging technology [5], and labor and technology requirements, in which a great degree of technical knowledge is needed to maintain these added technological practices [3].

The Oregon State University Openly Published Environmental Sensing Lab (OPEnS Lab) recognizes the environmental and cost benefits of increased BMP and aims to eliminate the two stated drawbacks of precision technologies through inexpensive and simple plug-and-play electrical and mechanical systems. The OPEnS Lab current solution to irrigation overwatering is through the use of SmarTrac Passive RFID Soil Moistures to autonomously control an irrigation system in a commercial irrigation boom, as examined in this thesis.

RFID Technology

Frequency Identification (RFID), is a type of automatic identification and data capture technological method that automatically identifies RFID objects (or "tags") using radio waves, captures digital data stored within each RFID tags, and sends the collected data into a database [6]. Due to its automatic nature, very little human interaction is needed.

Two types of RFID tags exist in the market today: active tags and passive tags. Active RFID tags contain a dedicated microchip on the tag itself that stores the digital data, usually in the form of an Electronically Erasable Programmable Read-Only Memory (EEPROM), an antenna that receives the radio frequency signal from an antenna, and a dedicated onboard battery to provide the necessary power to transmit data to the reader. Active RFID tags are

attractive over barcodes due to the avoidance of an optical scanner to obtain the EEPROM data – in other words, because Active RFID tags utilize RF radio waves for communication instead of an optical scanner, they can transmit data at great lengths (up to 100 feet) and without line-of-sight to the reader [7]. Passive RFID tags also contain a dedicated EEPROM and antenna to receive the radio frequency signal, but benefit from a much smaller, simpler, and cheaper design due to the lack of an onboard battery and its associating battery management system circuit. Instead, the passive RFID tags receives its power directly from the RF radio waves itself. While the lack of an onboard battery results in smaller data transmission distances (approximately 5 feet), it is an attractive technology due to the simplistic and cost-effective design from the added benefit of not needing to swap batteries [8].

Because the RFID tag and RFID reader are usually housed as separate units, the requirement for the RF signals to penetrate through a specific medium before read by the RFID tag can be used to sense different environmental conditions. This is done by allowing the RF signals to manipulate the circuitry on the RFID tag itself [9]. This was seen in a 2012 collaboration between SmarTrac and RFMicron, in which RFMicron's Chameleon technology was utilized in the creation of passive RFID tags [10]. With this Chameleon technology, the passive RFID tags senses its frequency shift and performance loss via differences in its Received Signal Strength Index (RSSI) due to disturbances and dielectric in its environment, and uses an internal variable capacitor to offset the impedance of its antenna, thereby offsetting that performance differential. This adjusted capacitor value is then stored in the RFID EEPROM memory bank, accessible via the RFID reader [11].

SmarTrac Passive RFID Tags as Soil Sensors

SmarTrac and RFMicron released its collaboration passive RFID tag product in 2015, commercialized as the Dogbone Passive RFID Tags, advertising the tags' capabilities to sample moisture content in the environment [12]. Originally intended for the automotive industry to aid in the automotive manufacturing process by detecting leaks during environmental testing, these tags have since been thoroughly examined within the OPEnS Lab as potential soil moisture sensors. SmarTrac advertises 5-bits of resolution, scaling from 0-31, of moisture level, posing 16 times more resolution than a standard Boolean logic of "dry" versus "wet."

In a previous thesis, these SmarTrac Dogbone Passive RFID Tags were calibrated with dedicated Decagon 5TM Soil Moisture Sensors. These Decagon 5TM Soil Moisture Sensors uses the frequency response of the probe submerged in a medium to determine that medium's relative permittivity. The Volumetric Water Content of soil (VWC) is obtained by its relative permittivity (ε_a) via this equation [13]:

$$VWC = 4.3x10^{-6}\ \varepsilon_a^3 - 5.5x10^{-4}\ \varepsilon_a^2 + 2.92x10^{-2}\ \varepsilon_a - 5.3x10^{-2}$$

In the above thesis experiment, 49 distinct trials were conducted with soil moisture levels of 1-31% VWC, measured by an industry-recognized Decagon 5TM Soil Moisture Sensor. With suspected outliers removed from the collected dataset, the SmarTrac Dogbone Passive

RFID Tags found a linear relationship with the Decagon 5TM Soil Moisture Sensor relative permittivity readings, with a calculated R-squared value of 0.72 [14]. Since the relative permittivity is proportional to the VWC of the soil, that statistical relationship from the passive tags applies to the Volumetric Water Content of the surrounding soil. In other words, 72% of the variation of the RFID tag samples used in the experiment is explained by the VWC of the soil, without the control of other external variables. While an R-squared value of 0.72 is not considered a substantial correlation by many statistical standards, it is considered valid in this case because many irrigation monitoring applications generally only need a qualitative scale of 1-5.

Due to the Passive RFID technology, cheap commercial selling price of the tags, and a found relationship with the Volumetric Water Content of the soil, the SmarTrac Dogbone Passive RFID Tags are a highly attractive soil moisture sensor for the autonomous irrigation system described in this thesis.

Design

Electrical Design

The SmarTrac Dogbone RFID Tag uses ultra-high frequency for RFID communication, which ranges from 902MHz – 928MHz within the United States. As a passive tag, these RFID tags do not require batteries, but rather sources its power from the reader itself. Passive RFID tags itself contain two elements: the antenna, and the EEPROM integrated circuit [8]. When hit by the radio frequency band emitted from a reader, the antenna portion of the RFID tag converts this RF energy to low voltage capable of waking the EEPROM, thus being able to access its data. The result is a cheap and simple RFID system capable of providing unique identification, albeit at the cost of communication range when compared to active RFID tags, which features dedicated onboard batteries.

The system at Peoria Gardens features an ALIEN ALR-8696 UHF RFID Antenna due to its desired IP67 Water and Dust Resistant rating, relatively high 11.0dBic signal gain, and the ability to tune the readability cone through simple power adjustment [15]. This RFID reader synergizes well with the SparkFun Simultaneous RFID Reader containing a ThingMagic Nano M6E RFID Reader Module, which acts as the interface between the RFID antenna and a microprocessing unit for data interpretation. While the SparkFun Simultaneous RFID Reader is not the only RFID Reader on the market, nor the cheapest, it is one of the more attractive options due to its availability and open-source firmware support.

The SparkFun Simultaneous RFID Reader, with its Arduino-shield footprint, was built to interface with an Arduino Uno and readily accepting 5V logic signals produced by the ATMega328P General Purpose Input/Output (GPIO). As industry slowly moves away from AVR architecture to a more capable ARM architecture, OPEnS Lab follows suit, with a recent adoption of the ubiquitous Adafruit Feather M0, which contains the ATMEL SAMD21 microcontroller, for its projects. The system at Peoria Gardens is no different; for feature expandability and promising libraries available for the Adafruit Feather M0, this was

the forefront runner for the microcontroller needed to interpret the RFID data stored in its EEPROM.

Due to the SparkFun Simultaneous RFID Reader's 5V capability and its 3.6V V_{IH}, a proper hardware interface was needed to communicate the SparkFun Simultaneous RFID Reader to the Feather M0 microcontroller, which uses 3.3V communication methods. A BOB-12009 SparkFun Bi-Directional Level Shifter was initially considered for this design for its low BOM cost and ubiquitous nature, but was quickly discarded for other implementations. This particular level shifter utilizes BSS138 NMOS MOSFETs for quick level shifting. With an input capacitance of 27pF, these MOSFETs are a standard method for level shifting slow I2C bus lines. The SparkFun Simultaneous RFID Reader communicates with the system's microcontroller using UART at 115200 baud, thus requires faster logic level shifting for its TX and RX communication. Traditionally, these level shifting approaches are done with digital buffer or inverter integrated circuits (i.e. Schmitt Triggers). For this application, two SN74AHCT125N Logic Level shifters were chosen for its backward compatibility; depending on the voltage applied to its Vcc pins, any incoming digital signal would be level shifted to that voltage level. As such, two of these ICs were necessary for the application: one that translates 3.3V digital logic to 5V, and another that steps voltage down from 5V to 3.3V.

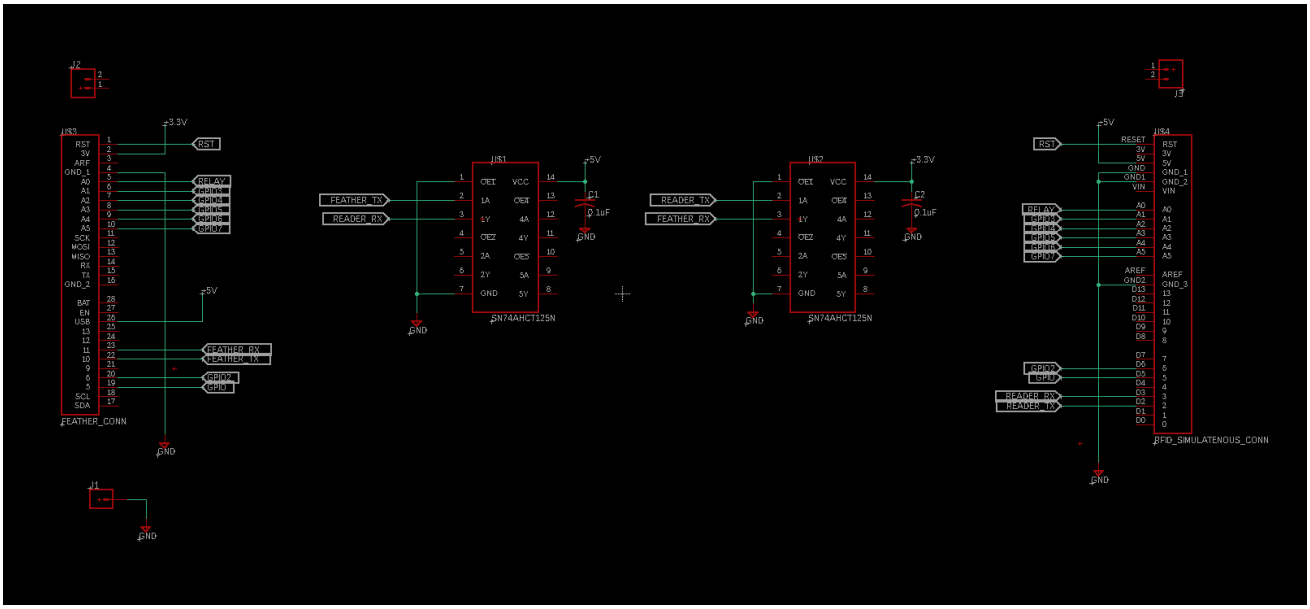


Figure 1. Schematic of the Interface PCB.

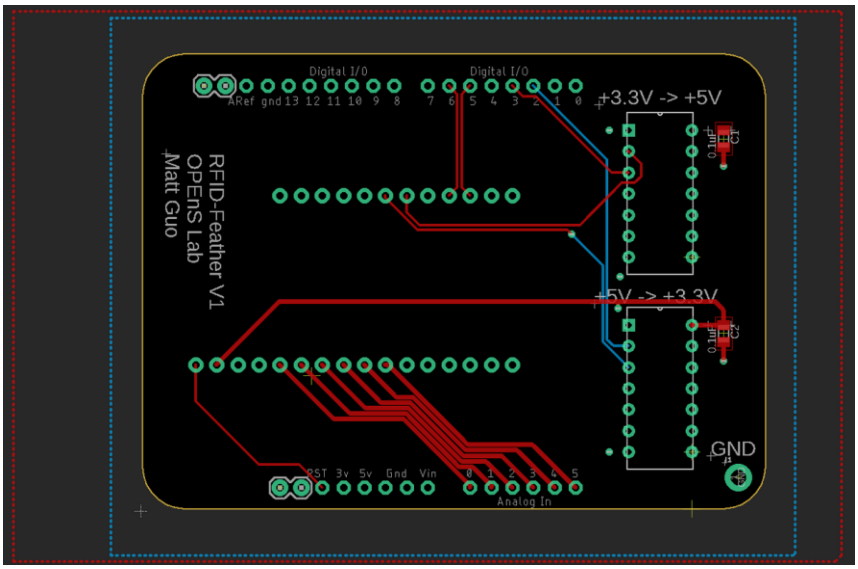


Figure 2. PCB Layout of the Interface PCB.

The intent for the interface PCB is to offer connections for both the SparkFun Simultaneous RFID Reader and Feather M0 for a quick “plug-and-play” solution. The top layer of this two layer PCB contains the 5V voltage rail extracted from the USB pin of the Feather M0, used to power the Nano M6E RFID Reader on the SparkFun board, which then feeds its power to the system antenna. The Feather M0 contains an onboard linear voltage regulator that steps down the 5V to 3.3V to power the ATMEL SAMD21 microcontroller. Due to this, all onboard electronics receives its power from a single 5V power source supplied to the USB pin of the microcontroller.

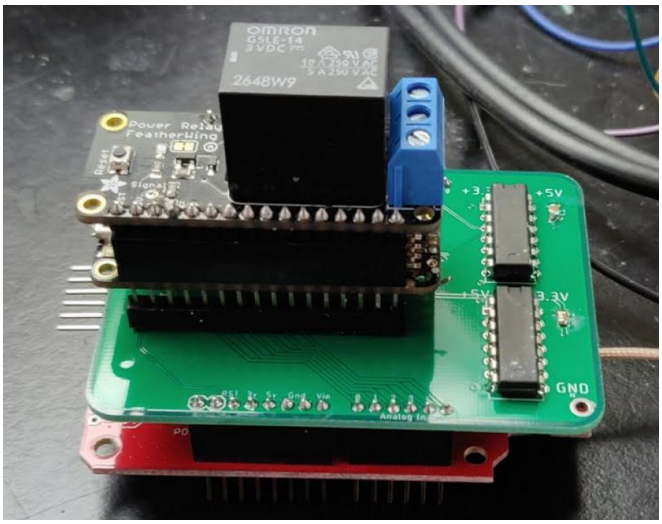


Figure 3. Electronics Sandwich Stack.

The autonomous irrigation system is controlled by the Feather M0 microcontroller. This microcontroller connects to an Adafruit Power Relay Breakout Board, which acts as a digital switch, to control the irrigation spray of the system. Before, the irrigation spray is initially manually controlled with a MTE-106E TriState Latching Switch that has three discreet

positions: OFF, forward spray, and forward and backward spray. This new, autonomous functionality makes use of the forward spray functionality rather than both the forward and backward spray, as it was believed that the latter state would be redundant for the proof-of-concept implementation; with the forward spray implementation, the irrigation could still turn on for the entirety of the irrigation boom path. The new, autonomous functionality identifies which row of crops need to be watered beforehand, manipulate the spray quickly using the forward spray functionality, and then shut itself off. This is done by connecting the Adafruit Power Relay Breakout Board directly onto the MTE-106E hardware switch. Through this connection, either one of the two ways can be used to turn on the irrigation spray (the physical switch or the digital switch); in a digital logic sense, an OR gate is created between the two – either the Power Relay ***or*** the switch can turn on the watering system.

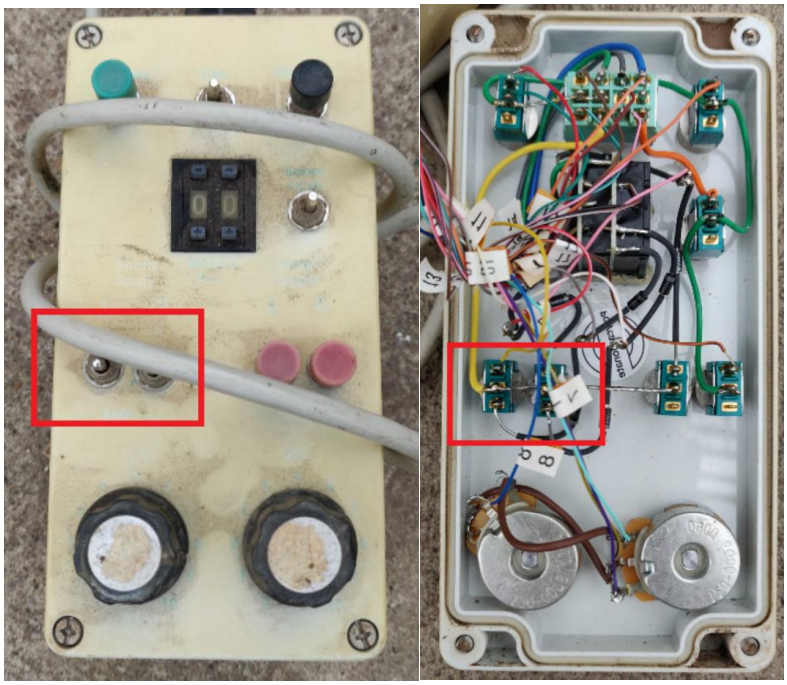


Figure 4. The Peoria Gardens Irrigation Controller, highlighting the Irrigation Switch.

Mechanical Design

Each passive RFID tag is enclosed in 3D printed ABS plastic. It was observed in testing that normal moisture levels could overwhelm the sensitivity of the tags. By 3D printing tag coverings, we can simultaneously protect tags from wear and tear, and control the thickness to optimize tag sensitivity to this particular application. To ensure near water-proof measures for the RFID tags, the 3D printing procedure for the RFID case offers a rather idiosyncratic instruction set: the 3D printer was paused after half of its printing duration to apply the RFID tag sticker to the base of the 3D printed ABS plastic, before resuming the 3D print to finish the encasement. While this procedure offers more meticulous care and planning, it was preferred over the initial experimented method of adhering two separate pieces of 3D printed ABS plastic, the bottom and the top, that sandwiches the SmarTrac RFID tag. 3D printing the casing as a singular unit offered consistent water-proofing, which was heavily desired for this application.

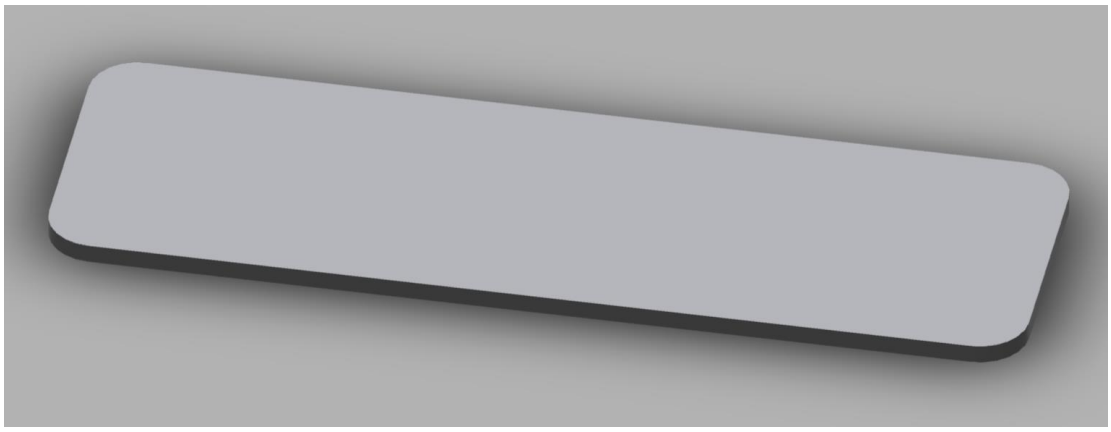


Figure 5. CAD model of the 3D Printed RFID case.

Figure 6. SmarTrac RFID Tag encased in the 3D Printed case.

For consistent readings of the passive RFID tags by the ALIEN ALR-8696 UHF RFID Antenna, the antenna was mounted to the irrigation spray boom itself. The antenna mount designed utilizes the four M3 screws on the back of the antenna to mount a 3D printed ABS plastic plate containing a rotating shaft holder for antenna angle adjustability. A metal bar is then fed into that shaft holder to lower the antenna towards the crops and thus, closer to the passive RFID tags in the soil.

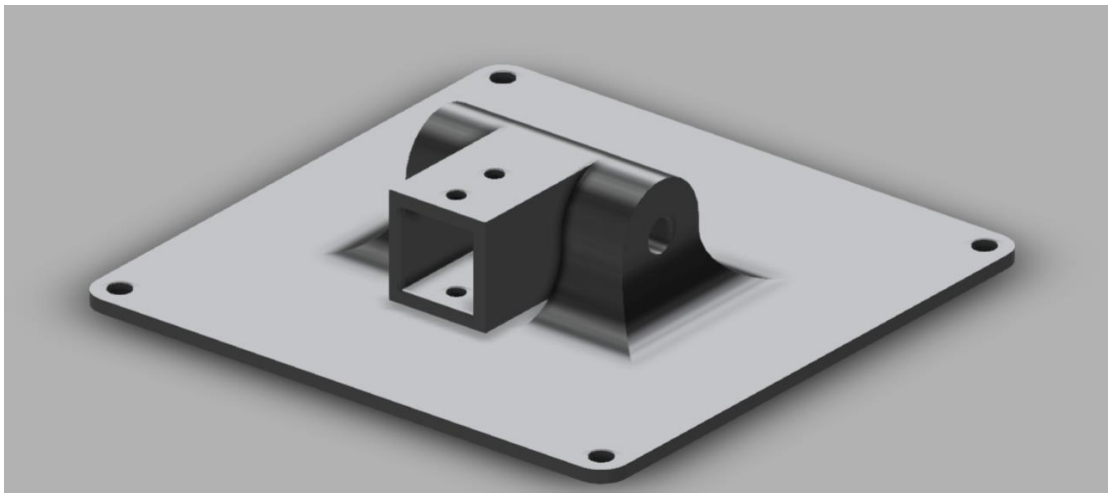


Figure 7. The antenna mount for the ALIEN ALR-8696 RFID Antenna.

The metal bar is then fed to another 3D printed grip that attaches itself onto the 2” diameter metal rod of the irrigation boom, where the water is sprayed from. To avoid a direct connection between metal and ABS plastic, which can potentially wear the 3D printed bracket over time, rubber tape was added between the plastic and metal connection, thus increasing friction. The bracket then clamps down to the metal rod through four hex screws.

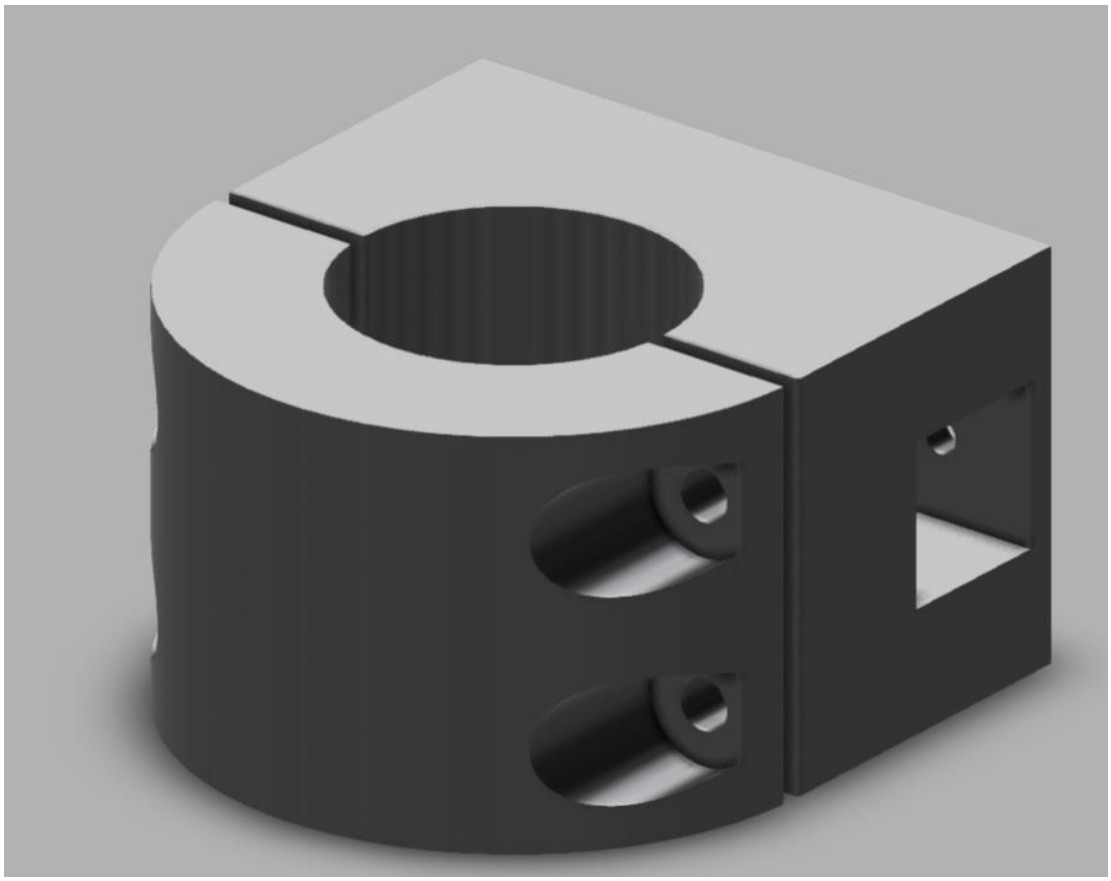


Figure 8. The irrigation boom 3D Printed mount.

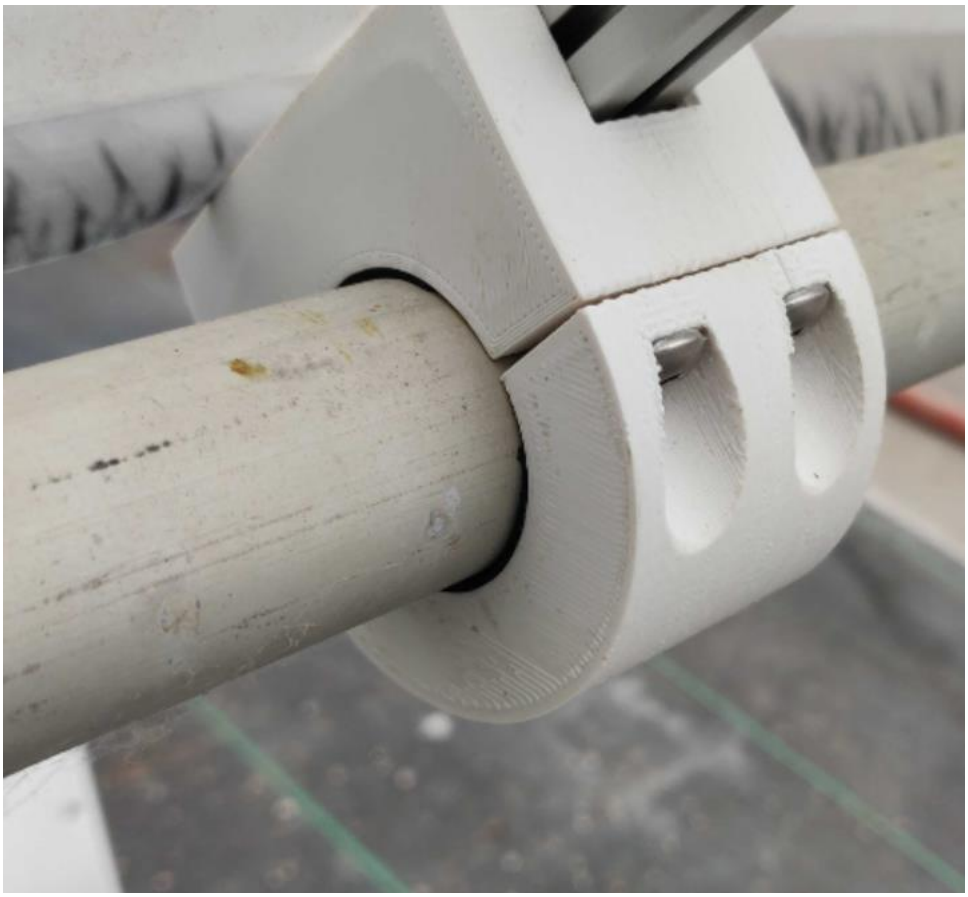


Figure 9. The irrigation boom 3D Printed mount implementation with rubber tape.

The electronic sandwich stack is protected in a water-resistant pelican case, with holes drilled on the top of the case to feed in the necessary wires for power and antenna coax cable. This is stored on the side of the irrigation rail to be near the irrigation boom controller for the onboard Adafruit relay. A momentary button is attached to the reset pin of the Feather M0, which shorts the pin to ground when pressed. This provides a quick, rudimentary way of giving a system reset for debugging purposes. Similarly for debugging, a multi-color LED is placed inside the Pelican case, with the LEDs illuminating through a thin layer of ABS plastic indicating current moisture reading status of dry, medium, or wet.

Figure 10. The water-resistance pelican case containing the onboard electronics and custom 3D printed irrigation mounting system (in white).

Firmware

The SparkFun Simultaneous RFID Reader and the ThingMagic M6E both contain open-source APIs to handle protocol communication between a microprocessor and the EEPROM of the passive RFID tags. Due to its compatibility with the Arduino IDE, the main platform used throughout OPEnS Lab hardware, the SparkFun open-source code, written by SparkFun CEO Nathan Seidle, was used for this system.

Intended for the Arduino Uno, which runs on AVR architecture, the SparkFun Simultaneous RFID Reader uses UART soft-serial protocol to communicate with that AVR microcontroller. This was done with the intent to preserve the hardware UART functionality for the Arduino Uno, which is the main communication method from the programming PC to the Arduino. Soft-serial is an interrupt-driven library for the Arduino family of microcontrollers used to mimic the functionality of hardware serial by synchronously generating digital pulses on any available GPIO.

To read the EEPROM data on the passive RFID tags, which contain the RSSI value, dictating the strength of the signal, unique TagID, programmable EPC data, communication frequency, and surrounding moisture value, the SparkFun library initially sends out read or write power of the antenna from the microcontroller to the ThingMagic M6E. This library also sends out region information (i.e. North America, Europe, China), as different regions around the globe have different radio frequency bands associated with their ultra-high frequency RF signals. One last data packet is then sent to the ThingMagic chip, dictating byte information to be transferred back to the microcontroller.

This last data packet was intercepted early in the design process after the realization that the byte information sent back did not contain all the necessary details needed for the design. In a previous thesis, the moisture value read by the passive RFID tags and programmable EPC data were not contained in the same packet transmission. This intercepted data packet ciphered string of bytes found in the transport logs from the Universal Reader Assistant made from ThingMagic. When pulling for both the EPC data of the passive RFID tag EEPROM, a different ciphered string of bytes was intercepted:

Table 1. The transferred packet from the Feather M0 to the ThingMagic M6E module for RFID communication.	
Original Packet	0x00, 0x00, 0x01, 0x22, 0x00, 0x00, 0x05, 0x07, 0x22, 0x10, 0x00, 0x1B, 0x03, 0xE8, 0x01, 0xFF
New Packet	0x00, 0x00, 0x01, 0x22, 0x00, 0x00, 0x05, 0x16, 0x22, 0x88, 0x10, 0x01, 0x1F, 0x03, 0xE8, 0x0F, 0xFF, 0x01, 0x00, 0x01, 0x09, 0x28, 0x03, 0xE8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0B, 0x01, 0xFA, 0xE6

While these data packets are ciphered, they are abstracted out to beyond the scope of this thesis. Using SparkFun library, by feeding these bytes to the M6E reader upon start-up, it mimics the transmission of that of the Universal Reader Assistant, effectively spoofing the hardware to send EEPROM data back to the microcontroller.

As previously mentioned, the SparkFun library uses soft-serial for the TX and RX pins on GPIO 2 and 3 of the SparkFun Simultaneous RFID Reader to preserve the functionality of the hardware serial for the PC programmer. When porting the firmware over to the Feather M0, this soft-serial implementation is not available due to incompatible interrupts for the ATMEL SAM20, thus alternatives were needed. Fortunately, the Feather M0 contains multiple dormant hardware UART support. Adapted from Adafruit's website on 'Adapting Uno Sketches to the M0', the Feather M0 PIO_SERCOM pinPeripheral was initialized on the

code to the Feather M0's pins 10 and 11 for TX and RX, respectively. This allowed the creation of a new serial object in the program, one that ties in with the hardware UART communication, arbitrarily called rfidSerial.

With the adjusted packet sent to the M6E by the Feather M0, a single array of bytes is returned from the SparkFun Simultaneous RFID Reader to the microcontroller, containing pulled data from the RFID tags' EEPROMs. This array of bytes, containing at least 33 unique bytes, includes the frequency of communication, programmable tag EPC, tag ID, RSSI signal strength, and the surrounding moisture value, and is parsed after looking through the Transport Logs of the Universal Reader Assistant software. The breakdown of the parsed information are as follows:

Table 2. The indices of RFID communication in the single array of bytes.	
Tag Moisture Data	Index 28
Tag RSSI	Index 13
Tag Frequency	Index 15 to 17
Tag EPC	Index 33 to the end of message array

Tag ID is not utilized in this application due to its unprogrammable nature; every worldwide RFID tag contains its own unique RFID code, or tag ID, given during its manufacturing process. While this tag ID could be used to associate tags for different crops in Peoria Gardens through a database or look-up table, utilizing a programmable tag EPC provided more flexibility and user control to this design.

The EPCs of each RFID tag are programmed through a separate Feather M0 sketch, which utilizes a write() function for the ThingMagic M6E, and sets a write power to the module upon initial start-up.

It must be noted that the desired functionality of an autonomous irrigation system is dependent on the grower's need, dependent on the crops grown. For this application, there is a need for two distinct irrigation passes: irrigate, which waters the crops, and fertigate, which waters and fertilizes the crops. Customizing the EPCs of each RFID tag is an integral part of determining whether one tag is to be used in either the irrigate or fertigate state, and thus is an integral part of the overall autonomous functionality. The target moisture values that each tag needs to achieve are also encoded into their EPCs, that determine the stopping points for the crops' wetting and drying cycles. Without the autonomous functionality for the irrigation system, all of this process was done subjectively: the grower would feel the soil moisture with hands, and then determine its moisture value on a subjective, Likert Scale from 1-5. The stopping points for the wetting and drying cycles ranged from 2-4. A state-machine is created to do this process automatically.

Below is a detailed diagram of this state-machine.

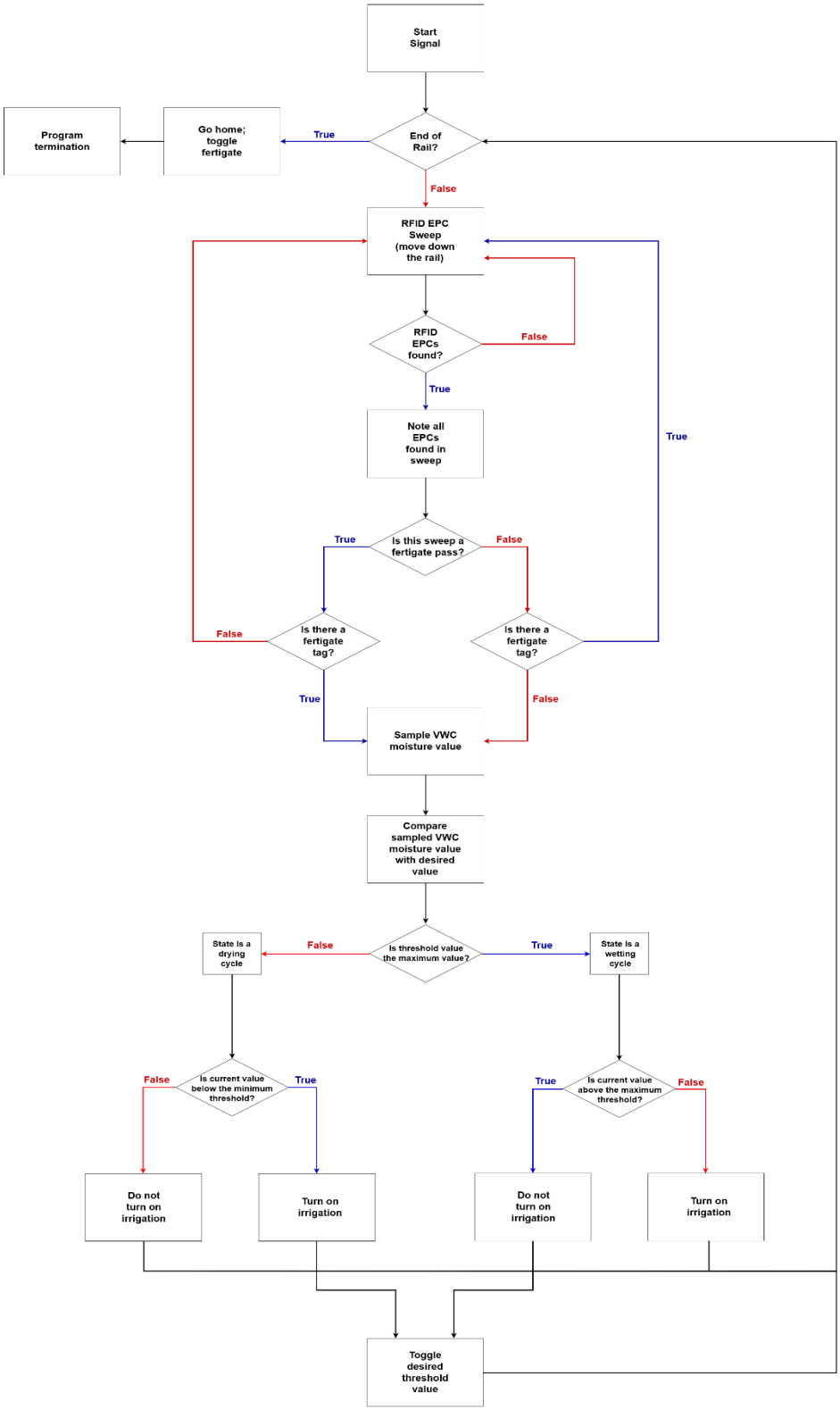


Figure 11. The State Machine Firmware Diagram.

The autonomous functionality of the irrigation system continuously loops through two conditional elements: if the irrigation boom reached the end of the rail, or if an RFID tag has been identified by the system antenna. Though the fertigate aspect of this state machine remains untested and will be a main next step for the future of this project, the logic behind this specific state has been added into the overall state machine design: upon reading an RFID tag, the program reads the current state of GPIO pin FERTIGATE to determine whether this state is in a fertigate pass or not. Thus, the fertigate state is user controlled through an onboard switch to the overall design.

Utilizing the programmable EPC, a fertigate tag is identified as RFID tags containing an arbitrary 0xFF heading to that EPC, parsed from the message array of bytes. All other tags arbitrarily contain the heading 0x0F, though as mentioned previously, can be programmed to contain different byte codes. During the fertigate state, the RFID reader will ignore all read RFID tags that contain the 0x0F heading, and will proceed to sample tags that contain the fertigate 0xFF code in the EPC.

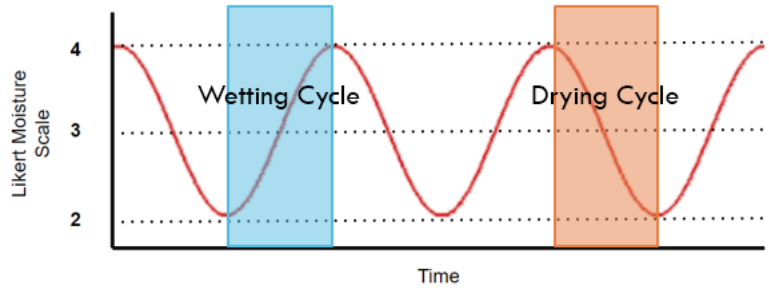


Figure 12: The oscillation curve for the wetting and drying cycle for the overall state machine.

Regardless of a fertigate state or not, the sampled RFID moisture value is then compared to predefined threshold moisture values dictating wet or dry soil. Per the request of Peoria Gardens and their Likert 1-5 moisture value scale for the pansy crops, with a 1 indicating very dry and a 5 indicating very wet, all pansy crops are initially watered up to a 4, before drying down to a 2. This process is then repeated, with it being watered back up to a 4, and then drying back down to a 2. As such, with this functionality, the irrigation boom state-machine is not handled through simple Boolean logic – is this current sampled soil value dry or wet? Rather, this value is compared to a threshold value indicating either a 2 or a 4 in the Peoria Gardens Likert scale, with additional information on whether the cycle is in a drying or wetting cycle. Once the sampled RFID moisture value is interpreted as dryer than the dry threshold value, that particular crop state information is then flipped from a drying cycle to a wetting cycle, with the compared threshold value updated to a wet threshold value. Once the sampled RFID moisture value is interpreted as wetter than the wet threshold value, that particular crop state information is then reverted back to a drying cycle, with the compared threshold value changing from a wet threshold value to a dry threshold value. Through this algorithm, the oscillation in crop irrigation is then preserved. So, in a sense, this is a self-adjusting algorithm that regulates transitioning between user-defined dry and wet states.

The data of each tag is abstracted and grouped into tag data structs in the C programming language, which is defined globally in the state-machine program. These structs contain the byte EPC, which acts as the RFID tag ID, a fertigate variable, defined as the EPC header of either 0xFF or 0x0F, the threshold moisture value of either dry or wet, a current state machine state, the RSSI information for that particular tag, the frequency of communication, and the moisture value. The motivation behind constructing tag objects is that each tag detected can have a separate state machine for added robustness on the overall implementation; rather than constantly inferring that each tag will share the same state, they can all be separated into different states for a more precise irrigation system, albeit with the cost of added computation and complexity. These tag objects are stored in a global tag object array of a known size, of which can be quickly changed depending on the crop implementing the passive RFID system.

Results

Calibration

The onboard EEPROM of the SmarTrac Dogbone Passive RFID tag contains 5 bits of resolution for the sensed moisture value of its surroundings, thus ranging from 0-31, where 31 is the driest the tags can sense. To fully integrate the functionality of the autonomous irrigation boom system, one of the earlier steps in the system design was calibration of the the 0-31 moisture value scale to Peoria Garden's Likert scale of 1-5, with 5 being the wettest the crops will see.

This calibration process was done through an intermediate step by taking pre-existing capacitive soil moisture sensors, the Adafruit I2C Capacitive Sensors, for moisture value comparison. These dedicated soil sensors were chosen for their wide availability and easy integration with the Feather M0, as well as their capacitive nature over typical resistive probes, which may degrade over time. The Adafruit I2C Capacitive Sensors suffered initially from saturation in their 10-bit resolution; at Likert Scales 4 and 5, dictating wet and very wet, respectively, the Adafruit I2C Capacitive Sensors reached its maximum possible capacitive values at 1016, thus providing inadequate data for comparison. This was combated using heat shrink to add an additional buffer layer between the capacitive probe and the surrounding soil. The number of heat shrink layers was optimized through thorough experimentation - a total of 5 different Adafruit I2C Capacitive Sensors were used, one without any heatshrink, two with one layer, and the last two with two layers, to sample the moisture values of soil with differing volumetric water content – ones that fit with Peoria Garden's Likert Scale.

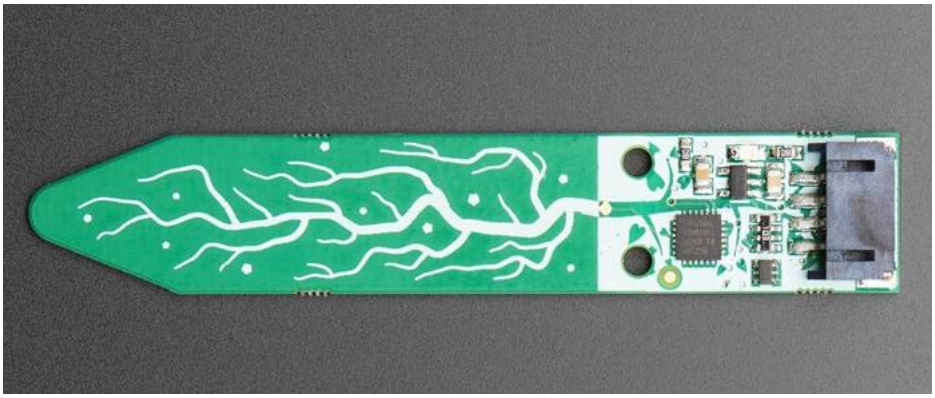
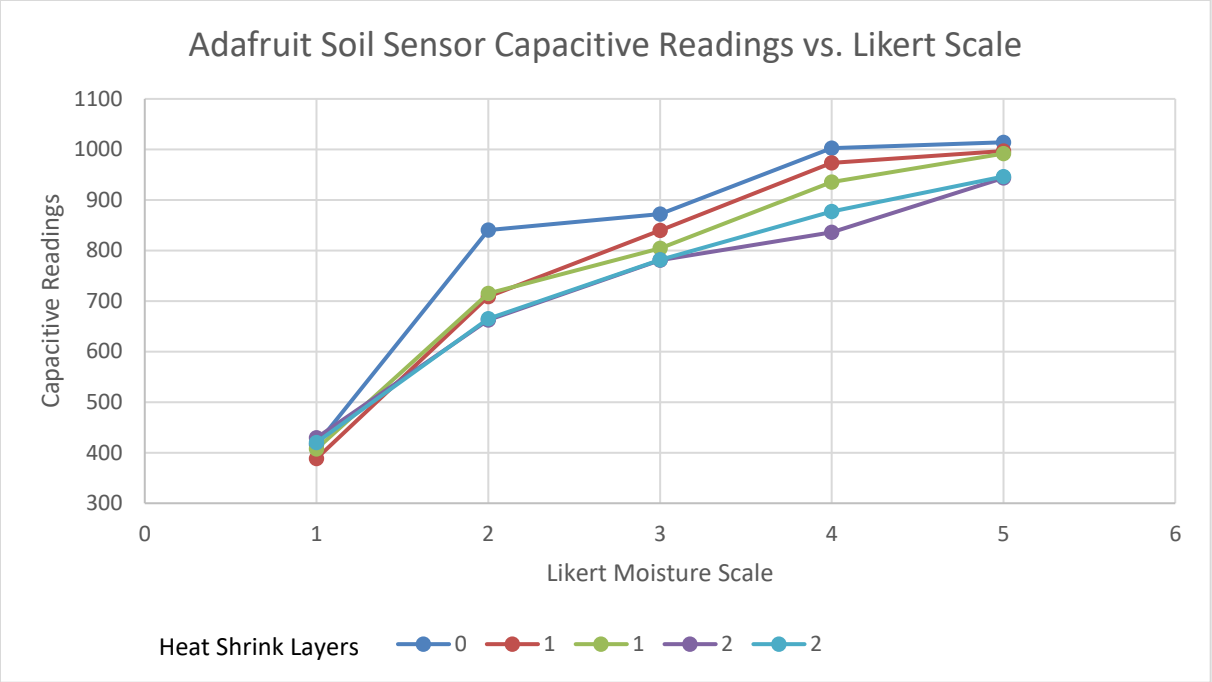


Figure 13. The Adafruit Capacitive Soil Sensor without Heat Shrink.

Figure 14. The Adafruit Capacitive Soil Sensor with Heat Shrink.

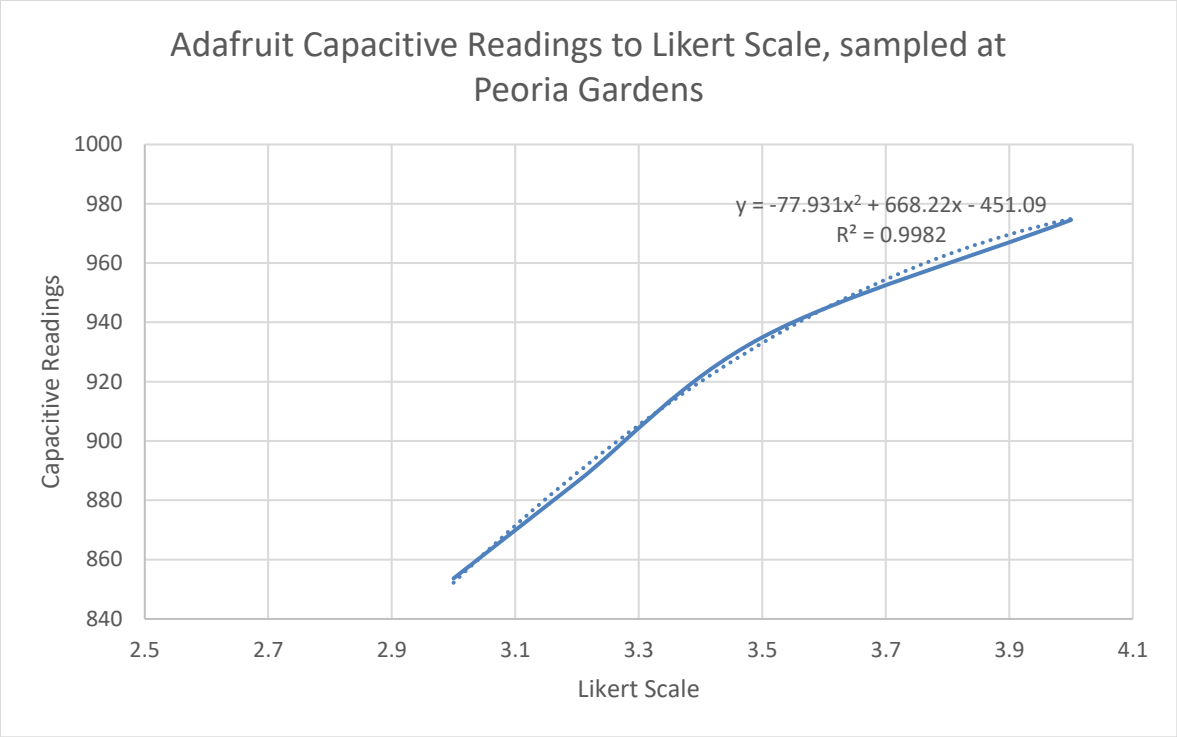


Graph 1. Adafruit Capacitive Soil Sensor capacitive readings vs. the Peoria Gardens Likert Scale with increasing heat shrink.

The Adafruit sensors sampled different soil moisture levels roughly 30 times with 5 minute intervals and logged the data on an onboard microSD card, written from the Feather M0. For these 30 samples on the differing moisture values, correlating to the Likert Scale, each were then averaged and plotted, as shown above. The graph above shows the average capacitive reading by the Adafruit sensors vs. the Likert Scale, with differing heat shrink layers.

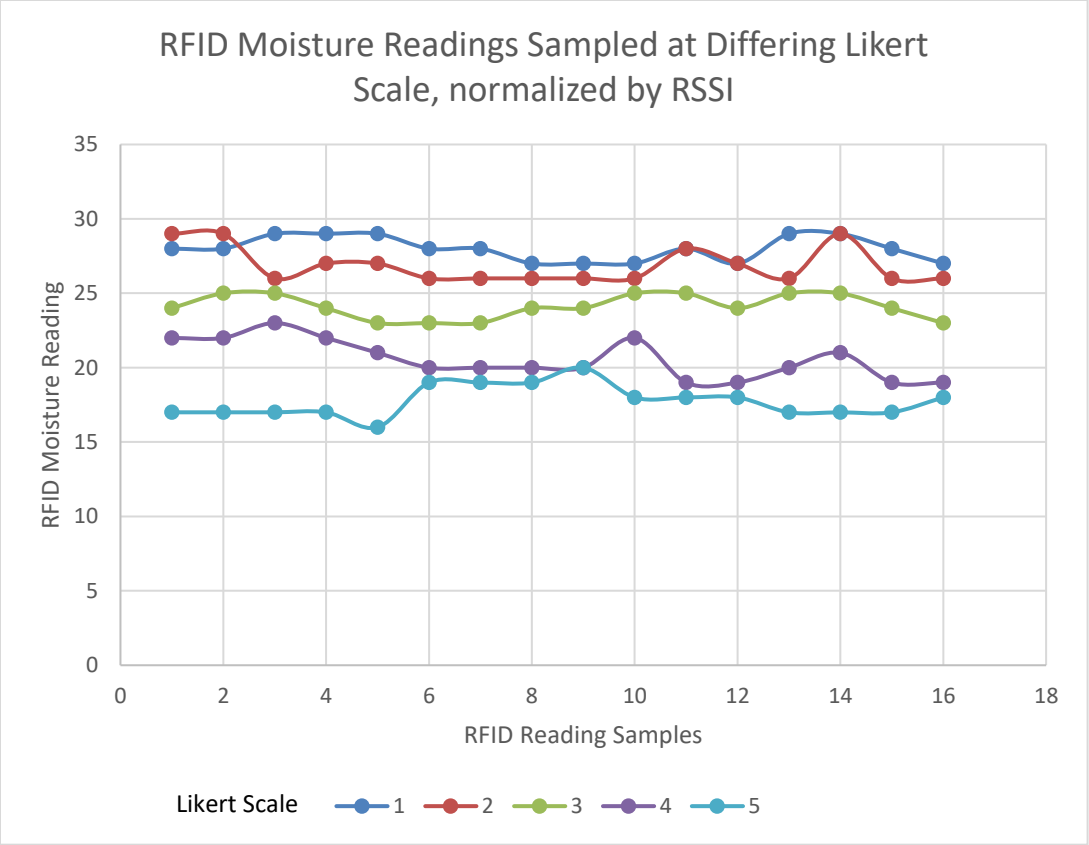
It was identified that the Adafruit sensors provided adequate resolution through one layer of heat shrink to shield the sensor from direct contact with the soil. For each additional layer of heat shrink, the maximum value read by the sensors at maximum Likert Scale decreased, thus preventing oversaturation in the readings. However, the addition of extra layers narrowed the resolution through the intermediate Likert Scale steps. Thus, the one layer solution was chosen for the optimal lower readings vs. narrower resolution trade-off.

From there, the Adafruit sensors were then plotted on different Peoria Gardens plot, sampling capacitive readings and differing Likert Scale values.



Graph 2. The Adafruit Capacitive Soil Sensor capacitive readings vs. the Peoria Gardens Likert Scale, showing linear relationship.

With the Adafruit sensor to Likert Scale conversion solidified, the passive RFID tags were then placed in soil with differing volumetric water content, initially measured by the Adafruit sensors, and using the provided polynomial line of best fit equation shown above to dictate the Likert Scale of 1-5. The RFID data was sampled 16 times for each Likert Scale soil, in 5 minute intervals. The 5 different colored data samples shown in the graph below indicate the different levels of moisture levels from the Likert Scale of 1-5.



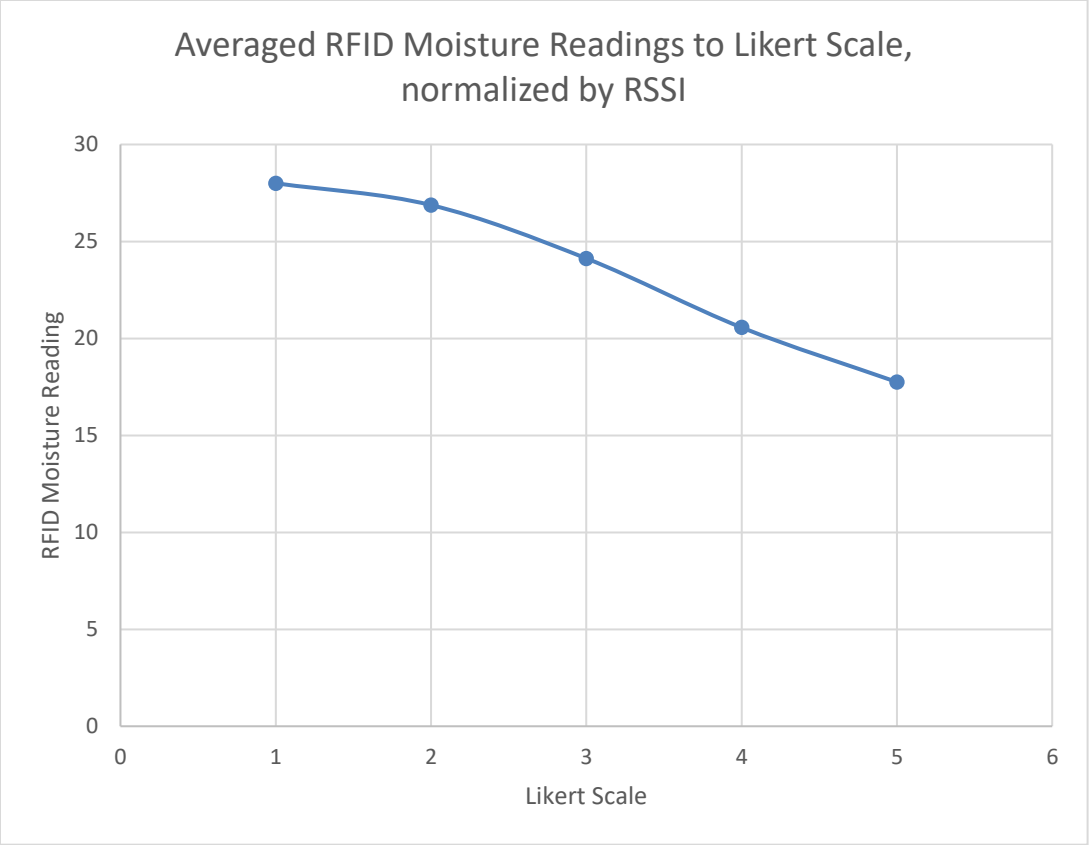
Graph 3. The SmarTrac Dogbone Passive RFID tag moisture readings sampled every 5 minutes at increasing Likert Scale values, normalized by RSSI.

Upon Initial glance on the RFID data obtained at the different Likert Scale, variation of the data can be spotted, even with the mean filter presented. Below details a table of the coefficient of variation of the presented 16 samples at the different moisture ranges.

Table 3. The Coefficient of Variation for the RFID tag data sampled at different moisture values.	
Likert Scale	Coefficient of Variation
1	0.029
2	0.044
3	0.033
4	0.064
5	0.059

The low coefficient of variation from the collected data suggests a relatively small level of dispersion and variation around the mean of each Likert Scale reading, thereby giving a certain degree of precision from the sampled dataset.

These 16 samples were then averaged throughout the different soil moisture levels, providing an RFID 5-bit reading that correlates with the Likert Scale used at Peoria.



Graph 4. The SmarTrac Dogbone Passive RFID tag moisture readings averaged at increasing Likert Scale values, normalized by RSSI.

Peoria Demonstration

Once the electrical, mechanical, and software designs were completed, the system was taken to Peoria Gardens to prove its viability out in the field. The water-tight Pelican case containing the onboard electronics was mounted to the side irrigation rail, near the irrigation boom controller. The RFID antenna, connected via a 3-feet coax cable to the SparkFun Simultaenous RFID Reader, was mounted onto the metal 2” diameter rail that contained the irrigation spray, with the antenna positioned parallel to the RFID tags for best response.

Figure 15. Final hardware implementation setup at the Peoria Gardens irrigation boom.

Eight SmarTrac RFID tags, encased in the ABS plastic casing, were programmed with unique EPCs, ranging from 0xA8 to 0xAF, and placed in the soil of the pansy crops, as shown below.

Figure 16. The SmarTrac Dogbone Passive RFID tags in pansy crops at Peoria Gardens.

The irrigation controller was connected to the Adafruit Power Relay board, connected to the Feather M0, which controlled the irrigation system of the RFID tags. A video was shown of the irrigation system spray functioning autonomously. For the purpose of clarity, and for needed extra calibration to the RFID tags (see the Potential Improvements section), the video shows this functionality on only selected crops. The link is provided below.

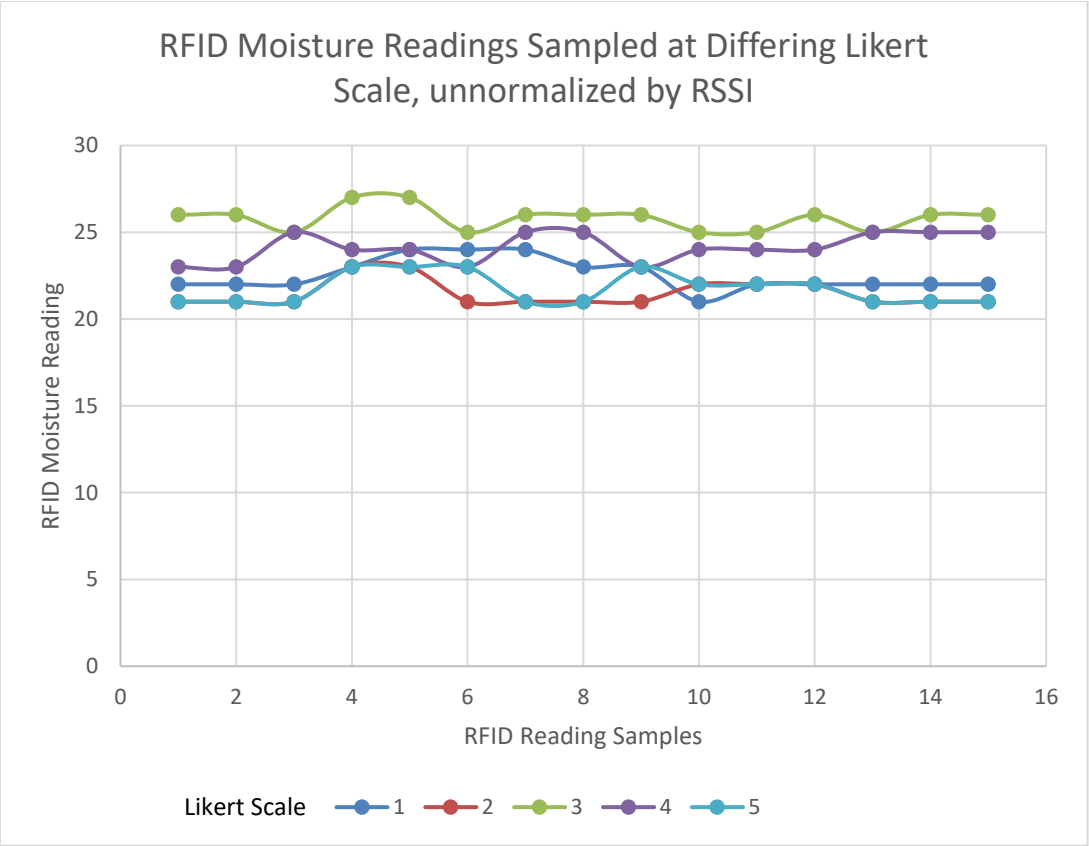
https://tinyurl.com/y3wetnsf

Conclusion

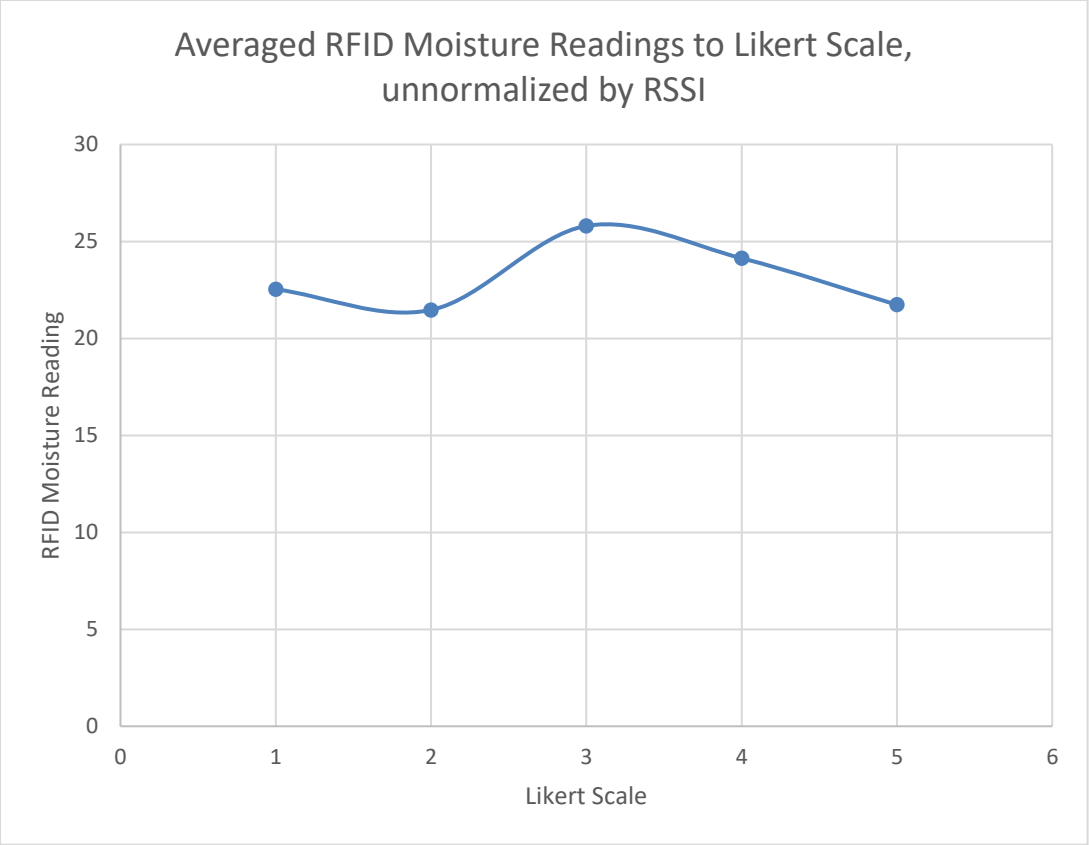
Potential Improvements

The autonomous irrigation system design at Peoria Gardens serves as a proof-of-concept design aimed to minimize water consumption and irrigation implementation expenses. While the outlook for this system is promising, there are improvements to be made for a more universal appeal.

During the calibration phase for this project, it was identified that the RSSI value stored in the EEPROM of the passive RFID tag had certain influence to the overall moisture value readings made by the SmarTrac tags. These unnormalized RSSI and moisture value readings are shown in the graphs below.



Graph 5. The SmarTrac Dogbone Passive RFID tag moisture readings sampled every 5 minutes at increasing Likert Scale values, unnormalized by RSSI.



Graph 6: The SmarTrac Dogbone Passive RFID tag moisture readings averaged at increasing Likert Scale values, unnormalized by RSSI.

It can be easily noticed that the graph obtained in Graph 6 varies drastically from the values shown in Graph 4. For Graph 4, the moisture readings from the passive RFID tags were normalized by RSSI. The procedure done to normalize the RSSI was a physical one: the RFID antenna's distance from the RFID tags were manipulated for each Likert Scale value such that each trial contained consistent RSSI values. However, for Graph 6, the RFID antenna was instead normalized by distance rather than RSSI; for the five different Likert Moisture Scale values, one consistent distance was used to read the data from the RFID tag. This leads to a different produced curve, showing that RSSI affects the moisture value readings produced by the passive tag.

One suggested reasoning behind this condition is due to the way the moisture values is obtained from the passive RFID tags. Due to their passive nature, all of the energy to sense the dielectric constant onboard the tag is done through the RF energy given by the antenna. Obscuring the beam path from the antenna to the associated tag downplays the RF energy needed to create the moisture value measurements, thus provides inaccurate moisture value readings, as shown in the collected data above. The solution for this is twofold: one can address this potential improvement through a mechanical aspect, in that the distance from the antenna to the RFID tag is adjusted during a read cycle to normalize the RSSI value. Another approach is a software one, in which an algorithm is created that takes in both the RSSI and

moisture value readings to appropriately normalize the readings based off of that given RSSI value.

The antenna is prone to premature readings of the RFID tags and may, as a result, prematurely activate the irrigation spray system. This can be identified in the figure below.

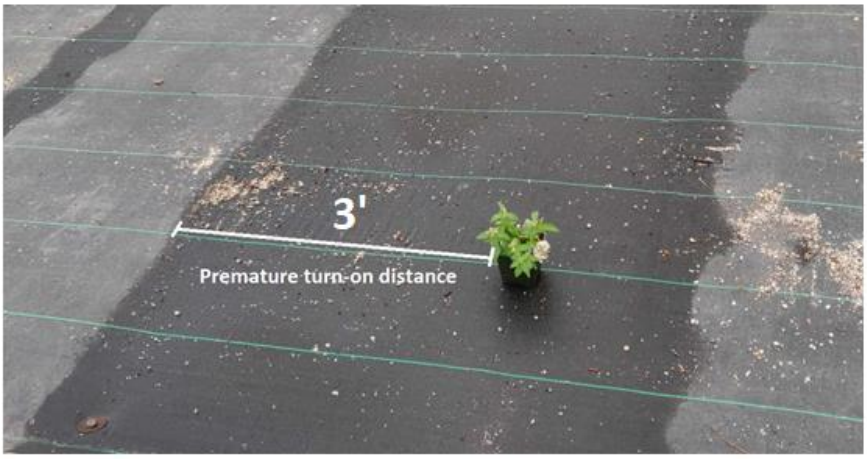


Figure 17. Result of the irrigation spray, showing premature turn-on distance of approximately 3 feet.

Further calibration is needed on the RFID tags and the antenna readings of the RSSI and moisture values to determine the appropriate delay time procedure to improve upon the premature irrigation spray in an attempt to avoid heavy water consumption. Possible solutions include appropriately setting the antenna power upon initial start-up and creating an RSSI threshold value to ignore read tags of a certain distance from the reader. One must note, however, that different moisture values on the soil may impede with the RSSI values, thus a constant RSSI threshold may not provide for a robust implementation for this design.

Further Development

Project Loom at the OPEnS Lab is a multidisciplinary collaboration of multiple open-source projects aimed to create fully modular and user-friendly sensor and actuator system kit for environmental research. A fully extensive library system for the Arduino IDE, Project Loom allows for rapid prototyping on commonly used electrical and mechanical hardware. Many OPEnS Lab associated projects have been adapted to the Project Loom framework, and the RFID Soil Moisture Sensor is near the top of the list for this systematic integration.

The motivation behind the Project Loom integration as a potential further development of the project is to provide additional user-friendly control to the project state machine. While the state-machine sketch abstracts many variables, soil moisture thresholds, and program objects to readable macros, the core of the OPEnS Lab community is to tailor environmental research and benefits to interested businesses and individuals outside the engineering realm. While slight adjustment to the program macros calls for minimal coding expertise, consistent adjustments through the design's calibration phase leaves much to be desired. Through integration with Project Loom, these macros can easily be adjusted through JSON text files,

which are automatically imported over to the program once the source code is compiled. This provides a rudimentary user interface for program adjustment, and takes the necessary steps to ensure proper plug-and-play hardware. The Project Loom team is currently collaborating with the greenhouse managers to identify user-driven features to interact with the autonomous irrigation system to further explore ways of providing user-friendly experiences with this hardware.

The current method for logging the soil moisture data is through an onboard, local microSD card attached to the Hypnos Board in the electronic stack. This allows all crops with the passive RFID tags to be later reviewed by the tag EPC, see the soil moisture value at a specific time, and provide information on drying or wetting cycle. Through an onboard microSD card, the system utilizes no WiFi capabilities, tailored towards individuals or businesses with limited WiFi or cellular connectivity. With an ever growing IoT domain to this society, however, it is believed by the OPEnS Lab to start translating services and data to the cloud. In the case for this application, the logged RFID data can be transferred over to a Google Sheet for a more interactive database system with the data collected. This would provide additional functionality to the design that tailors towards a larger commercial audience. Such functionality would require the utilization of a Feather M0 WiFi board.

Acknowledgements

I would like to express many thanks to my Committee Members, consisting of Dr. Chet Udell, Dr. Alec Kowalewski, and Nico Ardans, who helped with guidance, financial support, and analysis of this thesis. I would also like to express many thanks to Max Chu, who aided the mechanical design for the system, Brett Stoddard, who aided the calibration process for the RFID tags, and Peoria Gardens, for letting me use their irrigation system for this proof-of-concept design. Cara Walters, Dr. John Selker, Kamron Ebrahimi, Bao Nguyen, and Elijah Winkelman have also helped make this project a success.

References

[1] “Water Scarcity.” WWF, World Wildlife Fund, 2018, https://www.worldwildlife.org/threats/water-scarcity

[2] E. Folk, “Water Waste Around the World: Water Consumption by Country,” 2018. [Online]. Available: https://conservationfolks.com/stats-water-waste-around-the-world/

[3] M. Stubbs, “Irrigation in U.S. Agriculture: On-Farm Technologies and Best Management Practices,” *Congressional Research Service*, CRS Report No. R44158, October 2016. [Online] Available: https://fas.org/sgp/crs/misc/R44158.pdf

[4] “USDA – National Agricultural Statistics Service – 2012 Census of Agriculture – List of Reports and Publications,” 2012. [Online]. Available: https://www.nass.usda.gov/Publications/AgCensus/2012/

[5] F. Vuolo, L. Essl, and C. Atzberger, “Costs and benefits of satellite-based tools for irrigation management,” *Frontiers in Environmental Science*, vol. 3, no. 1, pp. 1-52, July 2015. [Online]. Available: https://doi.org/10.3389/fenvs.2015.00052

[6] “What is RFID and How Does RFID Work?” 2020. [Online]. Available: https://www.abr.com/what-is-rfid-how-does-rfid-work/

[7] “Types of RFID Tags,” 2020. [Online]. Available: https://www.abr.com/passive-rfid-tags-vs-active-rfid-tags/

[8] S. Smiley, “Active RFID vs. Passive RFID: What’s the Difference?” Dec 2019. [Online]. Available: https://www.atlasrfidstore.com/rfid-insider/active-rfid-vs-passive-rfid

[9] “Passive Sensors- A Guide to Passive Sensing Tags,” 2020. [Online]. Available: https://www.universalrfid.com/blog/passive-sensors-guide-passive-sensing-tags

[10] C. Swedberg, “Smartrac Group and RFMicron to Develop Passive Sensor Tags,” April 2014. [Online]. Available: https://www.rfidjournal.com/smartrac-group-and-rfmicron-to-develop-passive-sensor-tags

[11] “AN-FAM1601 Passive Sensors Technical Guide,” ver 1.0, October 2016. [Online]. Available: https://www.smartrac-group.com/files/content/Products_Solutions/PDF/Passive%20RFID%20Sensors%20Technical%20Guide_AN-FAM-1601_web.pdf

[12] “Sensor Dogbone™,” 2019. [Online]. Available: https://www.smartrac-group.com/files/content/Products_Solutions/PDF/0027b_SMARTRAC_SENSOR_DOGBONE.pdf

[13] G. C. Topp, J. L Davis, and A. P. Annan, “Electromagnetic determination of soil water content: Measurements in coaxial transmission lines,” *Water Resources Research*, vol. 16, no. 3, pp. 574-582. 1980. [Online]. Available: doi:10.1029/wr016i003p00574

[14] B. Stoddard, “Applying RFID Tags to Produce Economical Soil Moisture Sensors,” H.B.S. thesis, Dept. Elect. Eng., Oregon State University, Corvallis, OR, 2019.

[15] “ALR-8696-C Antenna” 2014. [Online]. Available: http://www.alientechnology.com/wp-content/uploads/Alien-Technology-ALR-8696-C-Antenna.pdf

Appendix

Source Code

```
/*
  Reading multiple RFID tags, simultaneously!
  By: Nathan Seidle @ SparkFun Electronics
  Appended By: Matt Guo
  Date: March 4th, 2020
  https://github.com/sparkfun/Simultaneous_RFID_Tag_Reader

  Constantly reads and outputs any tags heard

  If using the Simultaneous RFID Tag Reader (SRTR) shield, make sure the ser
ial slide
  switch is in the 'SW-UART' position
*/

//TODO: clean up relay_switch function so that toggling is not necessary for
 each iteration
//      of state machine.
//TODO: test fertigate functionality

//#include <Loom.h>
#include <Adafruit_NeoPixel.h>        // Library for NeoPixel functionality
#include "wiring_private.h"           // pinPeripheral() function
#include "SparkFun_UHF_RFID_Reader.h" // Library for controlling the M6E Nan
o module

#define NEOPIXEL A1
#define RELAY_SWITCH A0
#define UART_TX 10
#define UART_RX 11
#define FERTIGATE_SWITCH A2
#define RELAY_RSSI_THRESHOLD -80

#define NUMPIXELS 1
#define NUM_TAGS 10
#define RSSI_THRESHOLD -60
#define WET 16
#define DRY 21
#define MAX_VALUE 26
#define MIN_VALUE 17
```

```
bool _debug_serial = true;      //global bool for serial debug mode

enum state_enum {
   COMPARE_THRESHOLD,
   WET_CYCLE,
   DRY_CYCLE,
};

//struct for tag object, stores tag EPC, fertigates,
//moisture value, frequency of communication, etc
typedef struct{

   byte EPCHeaderName;
   byte fertigate;
   uint8_t threshold;
   uint8_t state = COMPARE_THRESHOLD;
   int rssi;
   long freq;
   long moisture;
   byte tagEPCBytes;

} tag;

tag tag_array[NUM_TAGS];      //global array for tag objects
int current_num_tags = 0;     //global counter for current number of tags
uint8_t fertigate_state = 0;  //global variable for the state of fertigate p
ass

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, NEOPIXEL, NEO_GRB +
NEO_KHZ800);


//const char* json_config =
//#include "config.h"
//;
//// Set enabled modules
//LoomFactory<
//  Enable::Internet::Disabled,
//  Enable::Sensors::Enabled,
//  Enable::Radios::Disabled,
//  Enable::Actuators::Disabled,
//  Enable::Max::Disabled
//> ModuleFactory{};
//
//LoomManager Loom{ &ModuleFactory };
```

```
//Tx ---> 10
//Rx ---> 11
Uart rfidSerial(&sercom1, UART_RX, UART_TX, SERCOM_RX_PAD_0, UART_TX_PAD_2 )
;

void SERCOM1_Handler() {
   rfidSerial.IrqHandler();
}

RFID nano; //Create instance
char color[10];
char EPCHeader_Hex[3];
int timeout_counter = 0;

int rssi;
long freq;
long moisture;
byte tagEPCBytes;
byte EPCHeader;
byte EPCFertigate;

void setup()
{

//  Loom.begin_serial(true);
//  Loom.parse_config(json_config);
//  Loom.print_config();
//
//  LPrintln("\n ** Setup Complete ** ");

   pinMode(RELAY_SWITCH, OUTPUT);
   pinMode(FERTIGATE_SWITCH, INPUT_PULLUP);

   pixels.begin();
   setColor(-1, -1);
   Serial.begin(115200);
   //while (!Serial); //Wait for the serial port to come online

   rfidSerial.begin(115200);

   pinPeripheral(10, PIO_SERCOM);
   pinPeripheral(11, PIO_SERCOM);
```

```
   while(!rfidSerial);

   delay(1000);

   StartOver:

    if (setupNano(115200) == false) //Configure nano to run at 38400bps
    {
      Serial.println(F("Module failed to respond. Please check wiring."));
      for(int j = 0; j < 5; j ++){
         for(int i = 0; i < NUMPIXELS; i++){
            pixels.setPixelColor(i, pixels.Color(150, 150, 150));
            pixels.show();
         }
         delay(300);
         for(int i = 0; i < NUMPIXELS; i++){
            pixels.setPixelColor(i, pixels.Color(0, 0, 0));
            pixels.show();
         }
         delay(300);
      }

      delay(10000);
      goto StartOver;
    }

   nano.setRegion(REGION_NORTHAMERICA); //Set to North America

   nano.setReadPower(2600); //5.00 dBm. Higher values may caues USB port to
brown out
   //Max Read TX Power is 27.00 dBm and may cause temperature-
limit throttling

   //Serial.println(F("Press a key to begin scanning for tags."));
   //while (!Serial.available()); //Wait for user to send a character
   //Serial.read(); //Throw away the user's character

   nano.startReading(); //Begin scanning for tags
}

//Gracefully handles a reader that is already configured and already reading
 continuously
//Because Stream does not have a .begin() we have to do this outside the lib
rary
boolean setupNano(long baudRate)
```

```
{

   delay(200);
   nano.enableDebugging();

   nano.begin(rfidSerial);

   //Test to see if we are already connected to a module
   //This would be the case if the Arduino has been reprogrammed and the mod
ule has stayed powered
   while(!rfidSerial); //Wait for port to open

   //About 200ms from power on the module will send its firmware version at
115200. We need to ignore this.

   while(rfidSerial.available()) rfidSerial.read();
   delay(100);

   Serial.println("Getting Version...");
   nano.getVersion();

   if (nano.msg[0] == ERROR_WRONG_OPCODE_RESPONSE)
   {
      //This happens if the baud rate is correct but the module is doing a c
continuous read
      nano.stopReading();

      Serial.println(F("Module continuously reading. Asking it to stop..."))
;

      delay(1500);
   }
   else
   {
      //The module did not respond so assume it's just been powered on and c
ommunicating at 115200bps
      Serial.println("Setting Baud...");
      nano.setBaud(baudRate); //Tell the module to go to the chosen baud rat
e. Ignore the response msg

   // rfidSerial.begin(baudRate); //Start the software serial port, this tim
e at user's chosen baud rate
   }

   //Test the connection
```

```
   Serial.println("Getting Version...");
   nano.getVersion();

   delay(200);

   nano.getVersion();

   if (nano.msg[0] != ALL_GOOD){
      Serial.println(nano.msg[0], HEX);
      return (false); //Something is not right
   }

   Serial.println(nano.msg[0], HEX);

   Serial.println("Setting Tag Protocol...");
   //The M6E has these settings no matter what
   nano.setTagProtocol(); //Set protocol to GEN2

   Serial.println("Setting Antenna Port...");
   nano.setAntennaPort(); //Set TX/RX antenna ports to 1

   nano.disableDebugging();

   return (true); //We are ready to rock
}

//main loop of the program, calls the state machine for individual
//tags upon successive tag responses
void loop()
{
   //readFertigateSwitch();
  if (nano.check() == true) //Check to see if any new data has come in from
module
  {
    byte responseType = nano.parseResponse(); //Break response into tag ID,
RSSI, frequency, and timestamp

    if (responseType == RESPONSE_IS_KEEPALIVE)
    {
      Serial.println(F("Scanning"));
    }
    else if (responseType == RESPONSE_IS_TAGFOUND)
    {
      //If we have a full record we can pull out the fun bits
```

```
      rssi          = nano.getTagRSSINew();     //Get the RSSI for this tag
read
      freq          = nano.getTagFreqNew();     //Get the frequency this tag
 was detected at
      tagEPCBytes   = nano.getTagEPCBytesNew(); //Get the number of bytes of
 EPC from response
      EPCHeader     = nano.getEPCHeader();      //Get the EPC header for ide
ntification
      EPCFertigate  = nano.getFertigateTag();   //Get the EPC fertigate code
      moisture      = nano.getMoistureData();   //Get the moisture data of t
he current read

      if(freq < 1000000 && nano.getAntennaeIDNew() == 17 && tagEPCBytes <= 2
0){

//       if(_debug_serial){
//          Serial.print("Fertigate State: ");
//          Serial.println(fertigate_state);
//       }

         //checking the fertigate pass
         //means that current sweep is not fertigate sweep
         //if(!fertigate_state){

             for(int i = 0; i <= current_num_tags; i++){
                if(tag_array[i].EPCHeaderName == EPCHeader){
                   //run state machine at i
                   updateData(i);
                   state_machine(i);
                }
                //no tag found, add in this new tag
                else if(i == current_num_tags){
                  if(current_num_tags < NUM_TAGS){
                    addNewTag(i);
                    current_num_tags++;
                  }
                }
           }

         //}
         //we are in the fertigate sweep, do not call state machine on non-
fertigate tags
//       else{
//
//             for(int i = 0; i <= current_num_tags; i++){
```

```
//                if(tag_array[i].EPCHeaderName == EPCHeader && tag_array[i]
.fertigate == 0xFF){
//                   updateData(i);
//                   state_machine(i);
//                }
//                //no tag found, add in this new tag
//                else if(i == current_num_tags){
//                   if(current_num_tags < NUM_TAGS){
//                      addNewTag(i);
//                      current_num_tags++;
//                   }
//                }
//             }
//
//       }

        //reset timeout
        timeout_counter = 0;

        delay(100);

        setColor(moisture, rssi);


        itoa(EPCHeader, EPCHeader_Hex, 16);

//        Loom.measure();
//        Loom.package();
//        Loom.display_data();
//
//        Loom.add_data("RFID", "RSSI", rssi);
//        Loom.add_data("Frequency", "Frequency", freq);
//        Loom.add_data("Moisture", "Moisture", moisture);
//        Loom.add_data("EPC", "EPC", EPCHeader_Hex);
//        Loom.add_data("Color", "Color", color);
//        Loom.SDCARD().log();
//        Loom.pause();

      }
      else{
         //implement timeout function
        timeout_counter++;
        if(timeout_counter > 10){
          setColor(-1, -1);
        }
```

```
      }

    }
    else if (responseType == ERROR_CORRUPT_RESPONSE)
    {
      Serial.println("Bad CRC");
    }
    else
    {
      //Unknown response
      Serial.println("Scanning...");
    }
  }
}

//Takes inputs of moisture and rssi values, sets the color on the NeoPixel
//accordingly.
//NOTE: DEBUG FUNCTION ONLY
void setColor(int moisture, int rssi){
  if(moisture >= 0 && rssi >= RSSI_THRESHOLD){
      if(moisture <= WET){
           for(int i = 0; i < NUMPIXELS; i++){
              pixels.setPixelColor(i, pixels.Color(0, 0, 150));
              pixels.show();
              strcpy(color, "blue");
           }
      }
      else if(moisture < DRY){
           for(int i = 0; i < NUMPIXELS; i++){
              pixels.setPixelColor(i, pixels.Color(0, 150, 0));
              pixels.show();
              strcpy(color, "green");
           }
      }
      else{
          for(int i = 0; i < NUMPIXELS; i++){
              pixels.setPixelColor(i, pixels.Color(150, 0, 0));
              pixels.show();
              strcpy(color, "red");;
           }
      }
  }
  else{
       for(int i = 0; i < NUMPIXELS; i++){
          pixels.setPixelColor(i, pixels.Color(0, 0, 0));
```

```
            pixels.show();
        }
    }
}

//Create a new struct tag object with a default initialization
//setup, then store the information to the tag array
void addNewTag(int i){
    i++;
    tag newTag;
    newTag.EPCHeaderName = EPCHeader;
    newTag.fertigate     = EPCFertigate;
    newTag.threshold     = MIN_VALUE;
    newTag.state         = COMPARE_THRESHOLD;

    tag_array[i] = newTag;
}

//Update the current tag object array to the most current values
//from the irrigation sweep
void updateData(int i){
    tag_array[i].fertigate   = EPCFertigate;
    tag_array[i].rssi        = rssi;
    tag_array[i].moisture    = moisture;
    tag_array[i].freq        = freq;
    tag_array[i].tagEPCBytes = tagEPCBytes;
}

//function that turns on the relay switch by pulsing high on
//the GPIO
void turnOnRelay(){
    digitalWrite(RELAY_SWITCH, HIGH);
}

void turnOffRelay(){
    digitalWrite(RELAY_SWITCH, LOW);

}

//void readFertigateSwitch(){
// if(digitalRead(FERTIGATE_SWITCH) == HIGH){
//    fertigate_state = 1;
// }
// else{
//    fertigate_state = 0;
```

```
// }
//}

//Irrigation state machine with three states: compare the threshold
//wet cycle (for continuous watering until low threshold is met), and dry cy
cle
//(for continuous drying until high threshold is met).
void state_machine(int index){

  switch(tag_array[index].state){

    case COMPARE_THRESHOLD:

       if(_debug_serial){
          Serial.print("EPC Tag: ");
          Serial.print(tag_array[index].EPCHeaderName, HEX);
          Serial.println(" COMPARE_THRESHOLD state");

       }

       if(tag_array[index].threshold == MAX_VALUE){
          if(_debug_serial){
             Serial.println("Transitioning to: DRY_CYCLE");
             Serial.println();
          }
          tag_array[index].state = DRY_CYCLE;
       }
       else if(tag_array[index].threshold == MIN_VALUE){
          if(_debug_serial){
             Serial.println("Transitioning to: WET_CYCLE");
             Serial.println();
          }
          tag_array[index].state = WET_CYCLE;
       }

       break;

     case WET_CYCLE:

      if(_debug_serial){
         Serial.print("EPC Tag: ");
         Serial.print(tag_array[index].EPCHeaderName, HEX);
         Serial.print(" Current RSSI: ");
         Serial.print(tag_array[index].rssi);
         Serial.print(" WET_CYCLE state ");
```

```
    Serial.print("Current moisture: ");
    Serial.print(tag_array[index].moisture);
    Serial.print(" Threshold moisture: ");
    Serial.println(tag_array[index].threshold);
    Serial.println();
  }

  if(tag_array[index].moisture > tag_array[index].threshold){
     if(tag_array[index].rssi > RELAY_RSSI_THRESHOLD){
       turnOnRelay();
       tag_array[index].state = COMPARE_THRESHOLD;
     }
     else
       turnOffRelay();
  }
  else if(tag_array[index].moisture <= tag_array[index].threshold){
     tag_array[index].threshold = MAX_VALUE;
     tag_array[index].state = COMPARE_THRESHOLD;
     turnOffRelay();
  }

  break;

case DRY_CYCLE:

   if(_debug_serial){
      Serial.print("EPC Tag: ");
      Serial.print(tag_array[index].EPCHeaderName, HEX);
      Serial.print(" Current RSSI: ");
      Serial.print(tag_array[index].rssi);
      Serial.print(" DRY_CYCLE state ");
      Serial.print("Current moisture: ");
      Serial.print(tag_array[index].moisture);
      Serial.print(" Threshold moisture: ");
      Serial.println(tag_array[index].threshold);
      Serial.println();
   }

   if(tag_array[index].moisture > tag_array[index].threshold){
      if(tag_array[index].rssi > RELAY_RSSI_THRESHOLD){
          tag_array[index].threshold = MIN_VALUE;
          turnOnRelay();
          tag_array[index].state = COMPARE_THRESHOLD;

      }
```

```
        }
        else if(tag_array[index].moisture <= tag_array[index].threshold){
          tag_array[index].state = COMPARE_THRESHOLD;
          turnOffRelay();
        }
  }
}
```