Design of an In-Canopy Sprinkler Monitoring System for Center Pivot Irrigation

by

Aaron Alexander Akin

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Biological and Agricultural Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2018

Approved by:

Major Professor
Dr. Danny Rogers

Copyright

Abstract

Recent innovations in the irrigation industry indicate a transition to more water efficient and uniform systems. This transition is necessary to preserve limited aquifer resources used by irrigation systems in the U.S. Great Plains where center pivot irrigation has become the dominant irrigation method. New in-canopy sprinkler packages have allowed these center pivot systems to operate more efficiently and uniformly, however, these in-canopy sprinklers hang low in the canopy and have the potential to become entangled in crop biomass and detach from the center pivot. Detached in-canopy sprinklers can impact the uniformity of the irrigation system resulting in decreased crop yields by disrupting the designed flow and pressure requirements of the sprinkler package. Therefore, it is crucial that producers detect and replace missing in-canopy sprinklers immediately. The current method to detect missing in-canopy sprinklers is manual inspection along the center pivot which uses significant time and labor. A monitoring system to alert the user of any detachments can eliminate unnecessary exploration and direct the user to the specific location of the detached in-canopy sprinkler.

A prototype in-canopy sprinkler monitoring system was designed to monitor in-canopy sprinklers on a center pivot irrigation system and alerts the user when and where an in-canopy sprinkler becomes detached from the center pivot span. The system utilizes three major components to accomplish this task: a master controller node, a series of sprinkler nodes, and a digital compass node. The master controller node requests the status of in-canopy sprinklers from each connected sprinkler node, which constantly monitors its connected in-canopy sprinkler, and if a sprinkler is found to be missing alerts the user via an SMS text message sent to their cell phone that the in-canopy sprinkler is missing and can be found at specific geographic coordinates. The master controller node calculates the geographic coordinates of the detached in-

canopy sprinkler by requesting the current compass bearing angle of the center pivot span from the digital compass node. This angle, combined with the known coordinates of the pivot point of the center pivot system and radius of the detached in-canopy sprinkler from the pivot point can be used to calculate the coordinates of the detached in-canopy sprinkler.

To test the performance of the designed system, it was connected to a demonstration center pivot and several trials were performed. The demonstration center pivot consisted of a rotatable span with eight detachable drop hoses and in-canopy sprinklers. Trials performed were designed to test the system's ability to react to detached in-canopy sprinklers and drop hoses, detect and identify issues that might arise during normal operation, and respond to user's SMS text messages with the proper system information as part of the user interface. The system successfully passed each set of trials ensuring that this prototype will accurately detect when and where an in-canopy sprinkler becomes detached and promptly alert the user.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3.3V	- 3.3 Volts Output Port
5V	- 5 Volts Output Port
EPROM	- Erasable Programmable Read-Only Memory
GND	- Ground Port
I^2C	- Inter-Integrated Circuit
IDE	- Integrated Development Environment
LEPA	- Low Elevation Precision Application
LESA	- Low Energy Precision Application
LPIC	- Low Pressure In-Canopy
MESA	- Mid Elevation Spray Application
N/C	- Normally Closed
N/O	- Normally Open
RAM	- Random Access Memory
RX	- Serial In Port
SCL	- Serial Clock
SDA	- Serial Data
SMS	- Short Message Service
TX	- Serial Out Port
UART	- Universal Asynchronous Receiver-Transmitter
USB	- Universal Serial Bus
VIN	- Voltage In Port

Acknowledgements

I would like to thank my major professor, Dr. Danny Rogers, for his support and providing me the opportunity to create a project that improved my skills as a researcher and as an instrumentation and control systems designer. This experience has been indispensable in preparing me for my future career. I owe a huge thank you to Dr. Stacy Hutchinson, as both an educator and a mentor, for guiding me throughout my undergraduate and graduate career and teaching me what it means to be a part of the Biological and Agricultural Engineering family. I'll never be able to repay you for the years of help. I would also like to thank Dr. Jonathan Aguilar and Dr. Xiaomao Lin for their assistance on this project and for serving as members of my supervisory committee. I would also like to thank Dr. Trisha Moore and Dr. Aleksey Sheshukov for allowing me to collaborate on research projects with them during my undergraduate career and improving my research skills. I would also like to thank Kari Bigham and Jonathan Zeller for putting up with my mess during the development of this project.

I would like to thank the engineering design teams I have participated in while at Kansas State University, ChemE Car and Fountain Wars, for growing my leadership skills and giving me the resources to learn how to design instrumentation and control systems. It was through this experience that I made lifelong friends and realized that I wished to pursue a career designing environmental instrumentation and control systems.

I would like to also acknowledge the Arduino community for providing examples and guidance on how to properly utilize the Arduino boards and associated libraries.

Finally and most importantly, I'd like to thank Royan Black for her love and support during this project and listening to me talk through hurdles I was facing while designing this system.

Chapter 1 - Introduction

Problem Statement

Recent innovations in the irrigation industry indicate a transition to more water efficient and uniform systems. This transition is necessary to preserve limited aquifer resources used by irrigation systems in the U.S. Great Plains where center pivot irrigation has become the dominant irrigation method (NRCS, 2005). Additionally, if these center pivot systems are operating at decreased levels of application uniformity, a measure of how evenly water is distributed across the field, crops may receive incorrect levels of water application. The accumulative impacts of these non-uniform irrigation events can decrease overall crop yield for the field (Kranz et al., 2012). Therefore, improving the uniformity and water use efficiency of center pivot irrigation systems is extremely beneficial to the user from an economic perspective.

Recent innovations with in-canopy sprinkler packages have allowed center pivot irrigation systems to operate more efficiently and uniformly. However, because these in-canopy sprinklers hang low in the canopy they have the potential to become entangled in crop biomass and detach from the center pivot (Melvin & Martin, 2018). Detached in-canopy sprinklers can create issues associated with excessive irrigation below the in-canopy sprinkler including runoff, soil erosion, anaerobic soil conditions, and deep percolation of nutrients (Rudnick, 2017). To prevent these uniformity issues it is crucial that producers detect and replace missing in-canopy sprinklers immediately. Currently, the method for detecting a missing in-canopy sprinkler is to walk along the center pivot span and visually inspect for any problems (Rudnick, 2017). Compounding on this issue, uniformity issues caused by detached in-canopy sprinklers may not be noticeable from aerial views and difficult to detect using yield maps (Kranz et al., 2012). A monitoring system that could alert producers when and where an in-canopy sprinkler becomes

detached (Figure 1) would be extremely beneficial in promptly repairing the system before uniformity issues occur.

Figure 1. A broken drop hose resulting in a detached sprinkler (Photo courtesy of Jonathan Aguilar, Kansas State University).

Objective

The objective of this project is to create a prototype system which monitors in-canopy sprinklers on a center pivot irrigation system and alerts the user when and where an in-canopy sprinkler becomes detached from the center pivot span.

Chapter 2 - Literature Review

Center-Pivot Irrigation

The use of freshwater for crop cultivation has been used for thousands of years. As populations grew and expanded, the need for irrigation rose to be used for agricultural crop irrigation, landscape maintenance, and soil revegetation. These applications have lead to the use of around 70% of the world's freshwater for irrigation with the United States using 115,000 million gallons per day in 2010 (Perlman, 2017). With increasing demands for irrigation, the methods to accomplish this requirement evolved from simple buckets to commonly used sprinkler systems. One popular system found in the United States is the center pivot irrigation system. An efficient and mechanized system, center-pivot irrigation is the dominant method in the Great Plains (Lamm et al., 2006).

Center pivot systems were initially deployed in this area during the late 1950s and incorporated widely spaced impact sprinklers which sprayed water high into the air over the field (NRCS, 2005). These early designs lost a significant portion of the applied water to evaporation and as aquifer levels in the region began to decline in the 1970s the need for more energy and water efficient systems arose (NRCS, 2005). Center pivot technology has continued to improve and current systems utilize in-canopy sprinklers to maximize water use efficiency and application uniformity.

In-Canopy Sprinklers

In-canopy sprinklers can be divided into four categories: MESA (mid elevation spray application), LPIC (low pressure in-canopy), LESA (low elevation spray application), and LEPA (low energy precision application). MESA and LESA packages consist of a nozzle located at the end of a drop hose and differ based off applicator height with MESA packages applying water

between 1.2 and 2.5 meters above the ground and LESA packages applying water between 0.15 and 0.60 meters above the ground (Lamm et al., 2006). LEPA packages apply water at the same height as LESA packages but incorporate a bubbler nozzle to apply water (Lamm et al., 2006). LPIC is seen as an alternative when LEPA or LESA packages cannot be used and incorporates a nozzle at the end of a drop hose that applies water at a height of 0.15 to 2.5 meters (Lamm et al., 2006). Figure 2 illustrates the setup of an in-canopy sprinkler package, with differences in nozzle type and height corresponding to the four packages previously discussed.

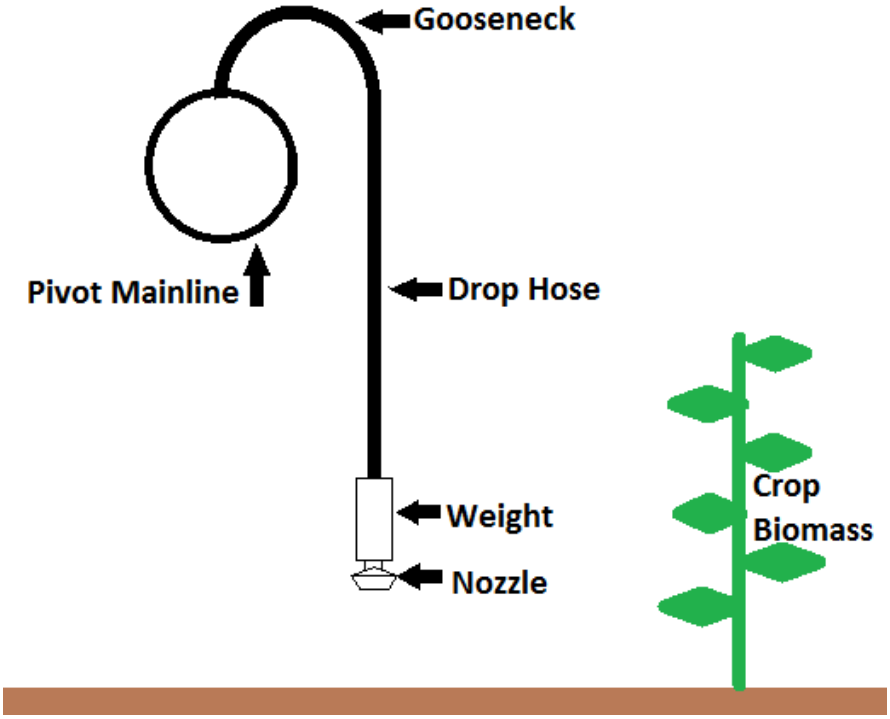

Figure 2. In-canopy sprinkler setup (Adapted from NRCS, 2005).

Water application efficiency is defined as the percentage of applied water which reaches the root zone. Lamm et al. (2006) cited that MESA systems operated at a water application efficiency of approximately 92% and LEPA systems operated at a maximum of 98% as visible in Table 1. More traditional sprinkler packages, such as the impact sprinkler, operated at a

maximum efficiency of 85% (Lamm et al., 2006). A higher water application efficiency equates to a lower quantity of water lost when irrigating crops. Therefore, it is in the user's benefit to use properly deployed in-canopy sprinkler packages on their center pivot systems to increase water application efficiency which will reduce water waste in areas where it is becoming a limited resource.

Table 1. Efficiency of sprinkler packages (Adapted from Lamm et al., 2006).

Sprinkler Package	Approximate Sprinkler Height	Efficiency
Impact Sprinkler	4.3 meters	85%
MESA	1.5 meters	92%
LEPA	0.3 meters	98%

Detached In-Canopy Sprinklers Complications

Several disadvantages do exist with in-canopy sprinklers that can lower their application efficiency and uniformity. According to Rudnick (2017), the largest source of low uniformity is faulty sprinklers, specifically detached in-canopy sprinklers. Detached in-canopy sprinklers can cause excessive irrigation resulting in runoff, soil erosion, anaerobic soil conditions, and deep percolation of nutrients (Rudnick, 2017). An example of soil erosion caused by detached in-canopy sprinklers can be seen below in Figure 3, which features a rill channel forming below the path of the detached in-canopy sprinkler. Detachment commonly occurs when in-canopy sprinklers and their associated drop hoses entangling themselves in crop biomass (Melvin & Martin, 2018). Additionally, these detachments are hard to detect without manual inspection because they are concealed within the crop canopy during the growing season (Melvin & Martin, 2018). To detect any detached in-canopy sprinkler Rudnick (2017) recommends walking the center pivot span to identify and replace any detached in-canopy sprinklers or drop hoses.

Figure 3. Soil erosion caused by a detached in-canopy sprinkler (Photo courtesy of Jonathan Aguilar, Kansas State University).

Importance of Application Uniformity

Crop yields are directly proportional to the application uniformity of an irrigation system (Kranz et al., 2012). Therefore, increased uniformity in irrigation systems is desirable from an economic and environmental view since it can increase crop yields while decreasing water waste (Lamm et al., 2006). Detached in-canopy sprinklers decrease uniformity of the irrigation system, with the cumulative impacts of these non-uniform irrigation events resulting in lowered crop yield while remaining difficult to pinpoint with aerial views and yield maps (Kranz et al., 2012).

For example, the row under a detached in-canopy sprinkler would experience excessive irrigation, and while the crops would survive the additional water, nutrients would leach below the root zone and cause decreases in crop yield (Kranz et al., 2012). Additionally, the detached in-canopy sprinkler would also affect the surrounding sprinklers by disrupting the designed flow and pressure requirements of the sprinkler package causing them to insufficiently irrigate their rows.

Decreased uniformity also directly impacts the process of chemigation. Chemigation utilizes the center pivot to apply chemicals such as fertilizer, herbicide, insecticide, or fungicide to the field. These applications need to wet the entire canopy to be effective (Melvin and Martin, 2018). The best way to accomplish this is through a properly designed irrigation system which can apply the chemical uniformly (New & Fipps, 2017). Therefore, missing in-canopy sprinklers will decrease the effectiveness of these chemical applications.

Introduction to Arduino

To improve current methods of detecting in-canopy sprinkler detachment in center pivots, an automated system controlled by a microcontroller such as Arduino is beneficial. Arduino has become one of the most popular microcontroller platforms due to its low cost, simple user interface, and open-source nature (Monk, 2016). Arduino can refer to both the hardware (boards) and software (IDE) created by the Arduino corporation who has made their products open-source to allow users to tailor their projects to meet their specific needs.

The core component of any Arduino board is the microcontroller. A microcontroller is a simple computer that contains a processor, a few kilobytes of RAM and EPROM, a UART serial data interface, and flash memory (Monk, 2016). Arduino boards have a variety of connection ports that allow the microcontroller to interface with analog and digital devices such as sensors,

actuators, and even other Arduino boards. Additionally, most Arduino boards have a USB port that allows the board to communicate with the Arduino IDE and the user's computer.

The Arduino IDE allows users to program and interface with an Arduino board using the Arduino programming language, which is based on the C programming language (Monk, 2016). When a program created in the Arduino IDE is uploaded to an Arduino board, the IDE first checks to see if the code conforms to the rules of the C language in a process called compilation (Monk, 2016). After the code is successfully compiled it is uploaded to the Arduino board's microcontroller which will then carry out the series of processes inscribed in the code. An additional benefit of the Arduino IDE is a serial monitor which allows users to monitor ongoing processes provided they have called the correct functions to write data to the serial monitor and that the board is connected to the user's computer through a USB connection.

I^2C Communication

Inter-integrated circuit, or I^2C, is a communication protocol that allows multiple microcontrollers or peripherals to communicate with one another (Monk, 2014). This is generally done with a "master" device requesting data from a series of "slave" devices.

In order for devices to communicate using I^2C protocol, all devices must join the I^2C bus by connecting to a common serial data (SDA) wire and serial clock (SCL) wire. If the connected devices are powered individually, a common ground (GND) wire must be connected between the devices as well. When a "slave" device joins the bus it must do so with a unique identifier, or address, for it to be properly recognized by the "master" device (Monk, 2014). Arduino libraries associated with I^2C protocol are limited to 7-bit addresses which allows a "master" Arduino to directly connect to up to 112 "slave" devices (Arduino, 2018b).

I^2C protocol occurs in 4 steps: a start condition, an address frame, data frames, and a stop condition. During the start condition, the "master" device pulls the SDA line low and the SCL line high which alerts all connected "slave" devices that a transmission is about to occur (Leens, 2009). The address frame is transmitted over the SDA line synchronized with the first nine pulses on the SCL line. During the first seven pulses, the "master" device transmits the 7-bit address of the desired "slave" device as well as an eighth pulse designating if the "master" is requesting or sending data (Leens, 2009). If the "slave" device recognizes that it has been called on the I^2C bus it pulls the SDA line low on the ninth pulse signaling to the master that it is available (Leens, 2009). Data can now be transmitted between the two devices during the data frame. During the data frame, the "master" device continues to generate clock pulses on the SCL line and 8 data bits are transferred from one device to the other, after which the SDA line is pulled low to signify that transmission is complete (Leens, 2009). The "slave" device is then released from the I^2C bus during the stop condition when the SDA line transitions from low to high after a similar transition on the SCL line (Leens, 2009).

The largest disadvantage of using I^2C protocol is that if one of the "slave" devices connected to the I^2C bus becomes unpowered it can cause the "master" device to freeze when it calls the address of the unpowered "slave" device. This can cause the entire system to become unresponsive until either the "slave" device regains power or is physically disconnected from the I^2C bus.

Chapter 3 - Materials and Methods

System Overview

To accomplish the objective of this project a prototype system was created to monitor individual in-canopy sprinklers and communicate system information such as when and where an in-canopy sprinkler or drop hose becomes detached from the center pivot. The monitoring system is comprised of three basic components: a master controller node, a series of sprinkler nodes monitoring each in-canopy sprinkler, and a digital compass node to detect the compass bearing of the center pivot (Figure 4). All components of this system were joined together over an I^2C bus and mounted inside weatherproof housing.

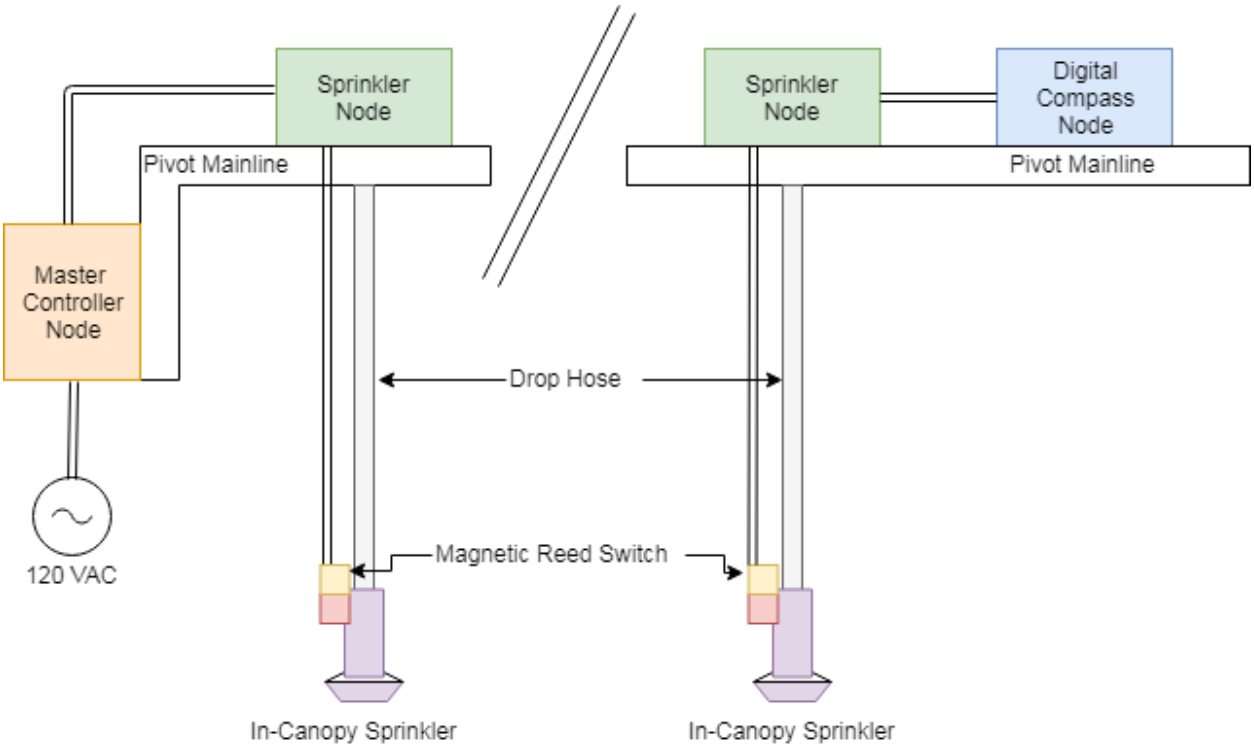

Figure 4. Overview of the in-canopy sprinkler monitoring system.

I^2C Solution

Unpowered “slave” devices can cause the master controller node to become unresponsive. A solution was developed which utilizes four relays to disconnect the “slave” device from the I^2C bus when it loses power. Figure 5 below, illustrates this solution when using an Arduino Uno microcontroller as the “slave” device connected in series on the I^2C bus. The incoming SCL line is connected to the common terminal of one of the relays. The normally closed (N/C) terminal of the relay is connected to the N/C terminal of the relay connected to the outgoing SCL line and the normally open (N/O) terminal is connected to the SCL port on the Arduino Uno. The same process is repeated for the incoming SDA line with the exception that the N/O terminal is connected to the SDA port on the Arduino Uno as well as the N/C terminal being connected to the N/C terminal on the relay connected to the outgoing SDA line. For the two relays connected to the outgoing SCL and SDA lines the N/O terminals are connected to the A5 and A4 pins on the Arduino Uno, respectively. In Figure 5 these relay terminals are represented as 1 for N/C, 2 for Common, and 3 for N/O. The incoming and outgoing GND lines are connected together into a single line and connected to the GND port on the Arduino Uno. If the “slave” Arduino Uno digitally writes the pins controlling each relay as ‘HIGH’ when powered, the device is connected to the I^2C bus. If the device loses power the SDA and SCL lines no longer connect to the device but rather bypass it completely, removing the “slave” device from the I^2C bus until power is restored. This solution efficiently keeps the monitoring system from becoming unresponsive.

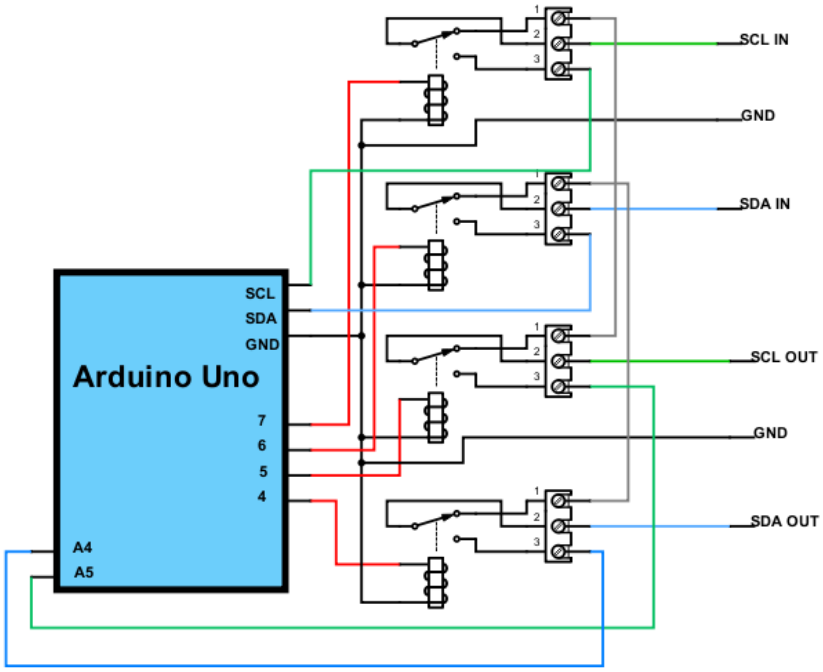

Figure 5. Schematic of the I²C solution to keep unpowered "slave" devices from causing the system to become unresponsive.

Master Controller Node

The master controller node is the brain of the entire system. It requests data from and monitors all nodes in the system as well as communicates all pertinent information to the end user. It consists of an Arduino MKR GSM 1400 connected to an Arduino Uno through a bi-directional logic level converter, CYT1076 (Figure 6). The program used for the master controller node can be found in Appendix A.

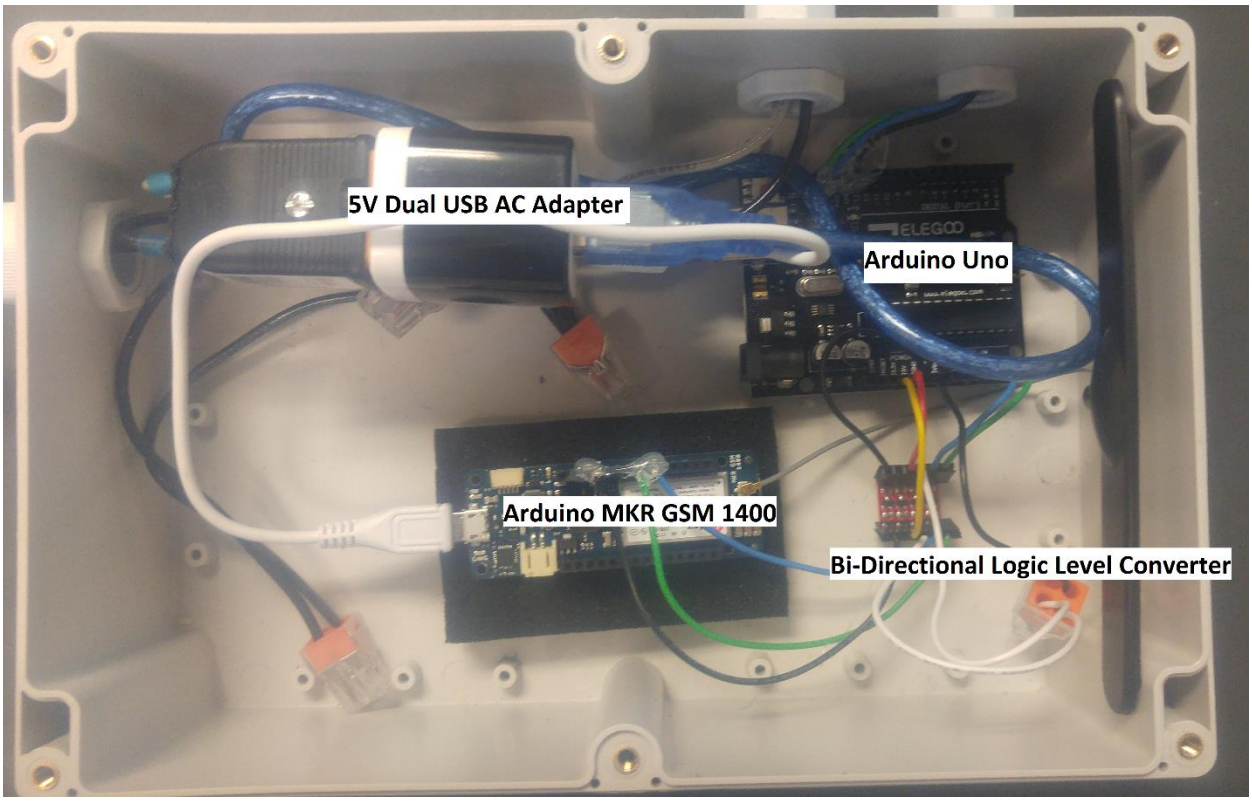

Figure 6. Components of the master controller node inside weatherproof housing.

The Arduino MKR GSM 1400 was chosen as the board for this node due to its integrated 3G cellular compatible modem. This allows the microcontroller to send and receive SMS text messages using the Arduino MKRGSM library. In addition, it contains large flash and dynamic memories as compared to the more standard Arduino Uno. This was a necessity as the Arduino Uno limits the dynamic memory for global variables to 2,048 bytes while the program developed for the master controller node used 2,102 bytes (Arduino, 2018a).

Since the Arduino MKR GSM 1400 uses 3.3 V logic and the other nodes use 5 V logic, all wires connecting the board to the I^2C bus needed to pass through a bi-direction logic level converter. The bi-directional logic level converter steps up all outgoing 3.3 V signals to 5 V while simultaneously stepping down incoming 5 V signals to 3.3 V. To accomplish this, the converter requires voltage inputs of 5 V and 3.3 V. The purpose of the Arduino Uno in the node

is to provide the necessary voltage inputs to the converter and connect to the I^2C bus for the rest of the system. A schematic of the master controller node detailing all connections can be seen below in Figure 7.

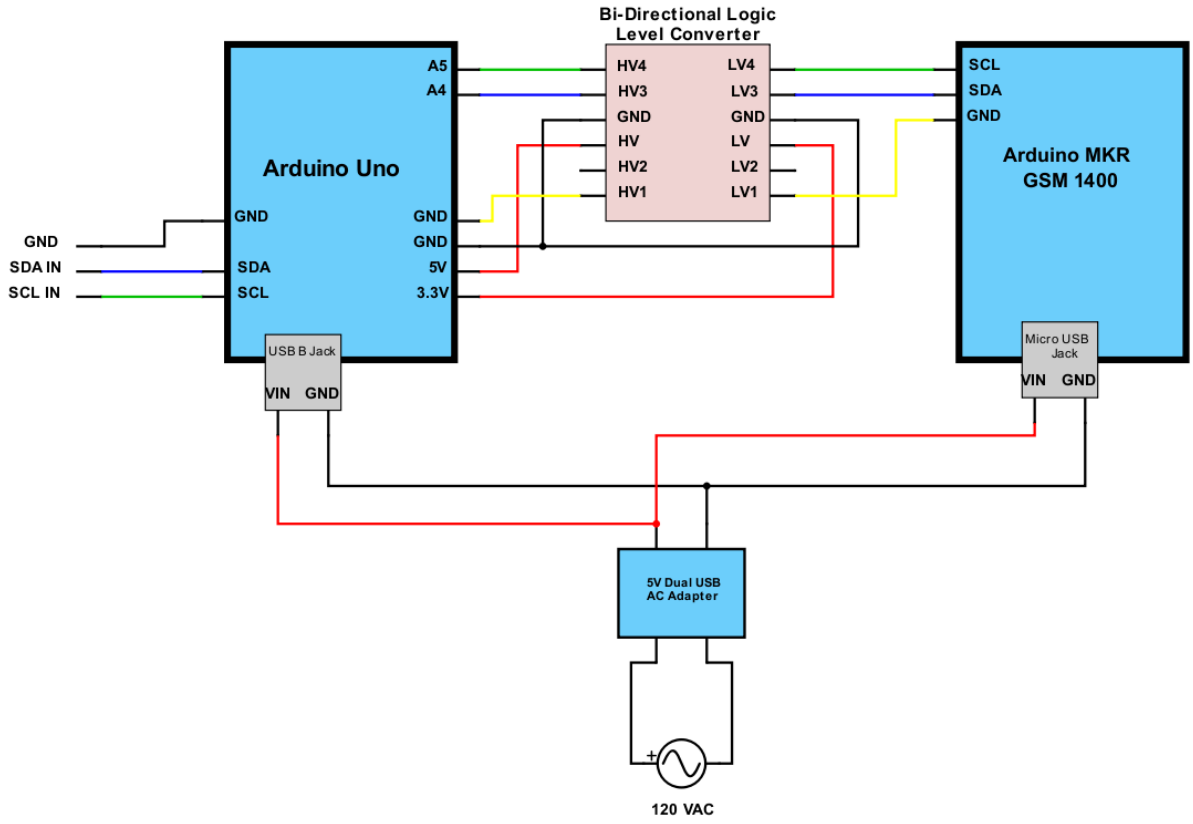

Figure 7. Schematic of master controller node detailing all connections.

The program setup is initiated when the master controller node is activated and begins by connecting the Arduino MKR GSM 1400 board to the I^2C bus and a GSM cellular network. Once the board has connected to a cellular network the program transitions from setup into the main loop. Initially, any incoming text messages are deleted from the modem memory with this process continuing up to 30 seconds since the device was activated. This is done to clear any SMS text messages that may have accumulated since the device was last activated. Once the

program has been running for more than 30 seconds all subsequent loops will skip this portion of the code. The program then requests two bytes of data over the I^2C bus from the address assigned to the digital compass node and stores these values in an array. The sum of the array is the center pivot compass bearing to be used in the checkSprinklers() subroutine. The program then checks to see which sprinkler nodes are available over the I^2C bus and stores the corresponding node addresses in an array for use in the sprinklerStatus() subroutine. The program then calls the checkSMS() and checkSprinklers() subroutines before resetting the array that stores all available nodes. The main loop of the program repeats itself indefinitely until the device is powered off.

The checkSMS() subroutine reads and responds to any incoming SMS text messages received by the system if the message is identified as one that generates a system response. A summary of all system responses to received messages can be seen below in Table 2. After the appropriate system response is completed the SMS text message is deleted from the modem's memory.

Table 2. System response to received SMS text messages during checkSMS() subroutine.

Received SMS Text Message	System Response
Sprinkler	Call sprinklerStatus() subroutine.
Help	Call systemCheck() and systemInfo() subroutines.
Compass	Respond with SMS text message informing the user of current center pivot compass bearing.
One	Pause system for 1 minute and send SMS text message informing the user of this.
Five	Pause system for 5 minutes and send SMS text message informing the user of this.

The checkSprinklers() subroutine is called during the main loop of the program and checks the status of each in-canopy sprinkler. It does this by requesting an integer value, either a '0' or '1', from each sprinkler node connected to the I^2C bus. If a value of '1' is received from a sprinkler node then the corresponding in-canopy sprinkler has become detached from the center pivot. An SMS text message is immediately sent to the user informing them which in-canopy sprinkler is missing as well as the approximate geographic coordinates where the detachment occurred. The geographic coordinates are calculated using known latitude and longitude values for the pivot point, the radius (meters) to each in-canopy sprinkler from the center stored in an array, as well as the current center pivot compass bearing angle. The latitude and longitude in decimal degree format for the pivot point must be declared as a long integer with the decimal point removed. This allows the program to calculate at a higher decimal precision than what is initially allowed by the microcontroller. The program initially converts the center pivot compass bearing angle from degrees to radians as well as the in-canopy sprinkler radius from meters to decimal degrees by multiplying it by a conversion factor. This conversion factor is the multiplication value to convert one meter to 10^{-5} decimal degrees based on the current latitude formatted as a long integer similar to the declared latitude and longitude values (Robinson, 1995). For example, this conversion factor for a latitude of 45° N is approximately 7.87. The program then uses the converted sprinkler radius and multiplies it by the sine of center pivot compass bearing in radians to get the differential longitude off of the known pivot point coordinates. The differential latitude is calculated by multiplying the center pivot compass bearing in radians by the cosine of the converted sprinkler radius. The known values of latitude and longitude for the pivot point are added to their differential counterparts resulting in a long integer for each portion of the geographic coordinates for the detachment site. These long integer

values are converted into strings and appended to include the decimal point in the appropriate position before being sent to the user as the approximate geographic coordinates of the detachment site. An example of the SMS text message response to a detached in-canopy sprinkler can be seen below in Figure 8. Finally, the checkSprinklers() subroutine calls the communicationLineCheck() subroutine.

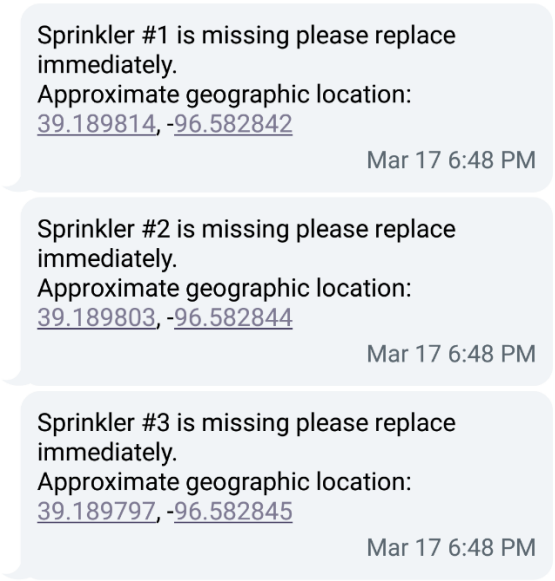

Figure 8. SMS text messages sent by the system in response to detached in-canopy sprinkler.

The communicationLineCheck() subroutine checks which sprinkler nodes are not connected to the I^2C bus and sends an SMS text message informing the user that the sprinkler node has had its communication wires disconnected or has lost power. This is an important subroutine for detecting system failure and ensures that the user can promptly perform maintenance on any faulty sprinkler nodes. An example of the SMS text message responses to sprinkler nodes being disconnected from the I^2C bus can be seen below in Figure 9.

Figure 9. SMS text messages sent by the system in response to sprinkler nodes not being detected on the I^2C bus.

The systemCheck() and systemInfo() subroutines are called when the message ‘Help’ is received as an SMS text message by the system during the checkSMS() subroutine. These subroutines send SMS text messages to the user with basic system information including if the system is operational and which messages to respond with to interact with the system. The full SMS text message response to the ‘Help’ message can be seen below in Figure 10.

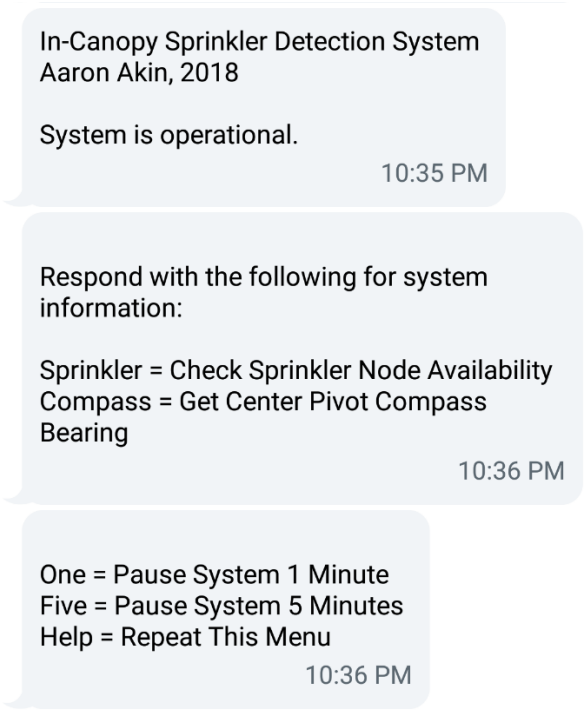

Figure 10. SMS text message sent by the system in response to the message 'Help' being received.

The sprinklerStatus() subroutine is called when the message 'Sprinkler' is received as an SMS message by the system during the checkSMS() subroutine. The subroutine initially sums the array containing all available sprinkler nodes created in the main loop. If the sum of this array is greater than zero, the subroutine sends an SMS text message to the user listing all operational sprinkler nodes. If the sum of the array is equal to zero, then no sprinkler nodes are currently operational and an SMS text message is sent to the user detailing this information. An example of the SMS text messages sent during this subroutine can be seen below in Figure 11. The subroutine then calls the communicationLineCheck() subroutine to inform the user of all nonoperational sprinkler nodes.

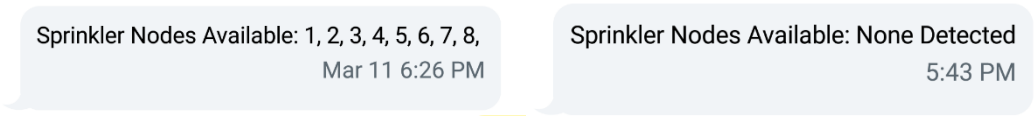

Figure 11. SMS text messages sent by system in response to the message 'Sprinkler' being received when sprinkler nodes are available (left) and when no sprinkler nodes are available (right).

User Interface

A user-interface is necessary if a designed system is required to effectively communicate with users. In the case of this system, the user-interface is primarily built into how the user receives and requests information from the system via their cell phone. By building functions into the program of the master controller node to send and respond to SMS text messages, as described in the previous sections, the system is able to efficiently and promptly communicate system information to the user.

Sprinkler Nodes

Each drop hose and in-canopy sprinkler set is equipped with a sprinkler node monitoring for loss of either component. The basic components of each sprinkler node are an Arduino Uno, a N/O magnetic reed switch, and a relay shield (Figure 12). The program used for each sprinkler node can be found in Appendix B.

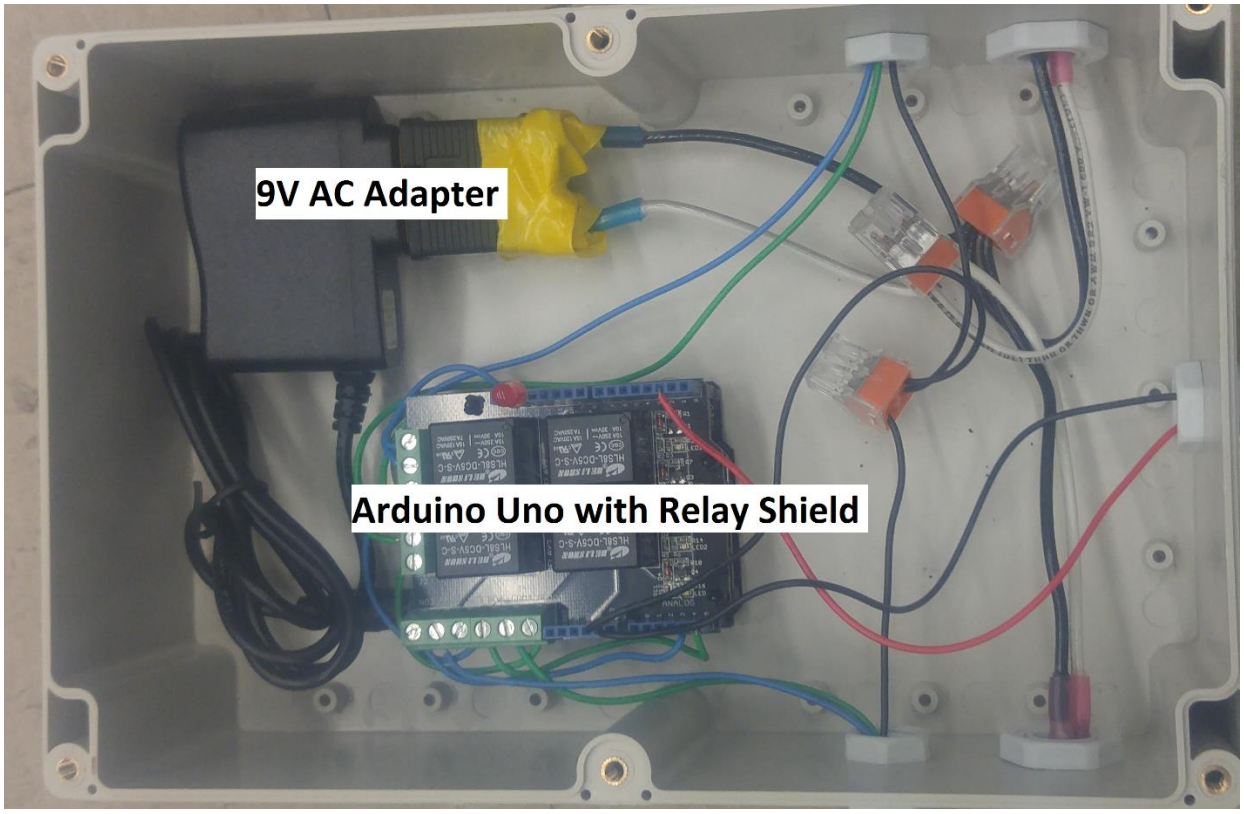

Figure 12. Components of sprinkler node in weatherproof housing.

Similarly to the schematic described in Figure 5, the incoming SCL and SDA lines are connected to the common terminal of two of the relays. The N/C terminals of the relays are connected to the N/C terminals of the relays connected to the outgoing SCL and SDA lines with the N/O terminals is connected to the SCL and SDA ports on the Arduino Uno. The two relays connected to the outgoing SCL and SDA lines have their N/O terminals connected to the A5 and A4 pins on the Arduino Uno, respectively. In Figure 13 below, these relay terminals are represented as 1 for N/C, 2 for Common, and 3 for N/O. The incoming and outgoing GND lines are connected together into a single line and connected to the GND port on the Arduino Uno.

To detect any loss of the in-canopy sprinkler or drop hose, a detachable two-wire plug was connected to digital pin 2 and the GND port on the Arduino Uno and mounted outside the weatherproof housing to the center pivot span. The other half of the detachable two-wire plug

was connected to the N/O magnetic reed switch located directly above the in-canopy sprinkler via wires along the length of the drop hose. If the in-canopy sprinkler with attached magnet becomes detached from the drop hose, the circuit is opened which is registered as a ‘HIGH’ signal on digital pin 2 of the Arduino Uno. Similarly, if the drop hose is detached, the two halves of the detachable plug disconnect and the Arduino Uno registers the loss in the same way as a detached in-canopy sprinkler. A schematic of the sprinkler node detailing all connections can be seen below in Figure 13.

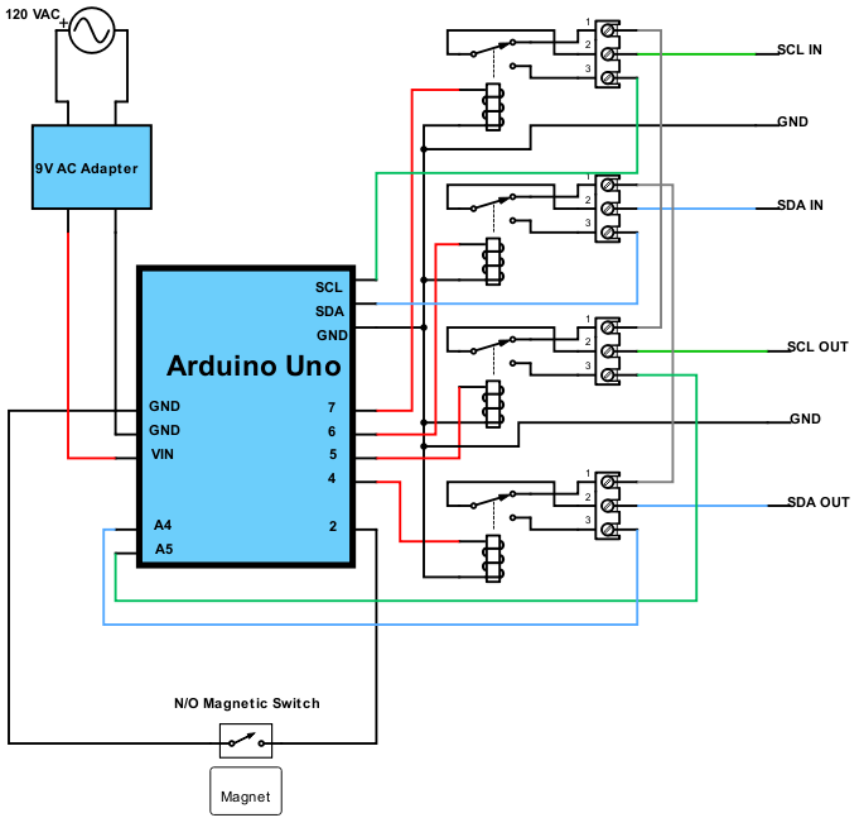

Figure 13. Sprinkler node schematic detailing all connections.

The sprinkler node program setup begins by digitally writing pins 4 through 7 as ‘HIGH’ to ensure that all relays on the relay shield remain powered. The program then joins the sprinkler

node to the I^2C bus as a "slave" device with an address matching the in-canopy sprinkler it is monitoring. When the master controller node requests data from the sprinkler node over the I^2C bus it responds with an integer value of a '1' or a '0'. If the in-canopy sprinkler or drop hose is detached from the center pivot span, the integer value sent to the master controller node is '1'.

Digital Compass Node

The digital compass node measures the center pivot's compass bearing angle and transfers this value over the I^2C bus to the master controller node. The primary components of this node are two Arduino Uno boards, a relay shield, and a digital compass module. The digital compass module, HMC5883L, senses 3-axis magnetic vectors and reports this information to a connected device over an I^2C bus. A schematic of the digital compass node detailing all connections can be seen below in Figure 14. Since the digital compass node utilizes two Arduino Uno boards to accomplish its task, a unique program was written for each board. The 'Digital Compass Module Arduino' program was uploaded to the Arduino Uno connected to the digital compass module (right in Figure 14) and the 'Digital Compass Slave Sender Arduino' program was uploaded to the Arduino Uno connected to the I^2C bus of the monitoring system (left in Figure 14). These programs can be found in Appendix C and D.

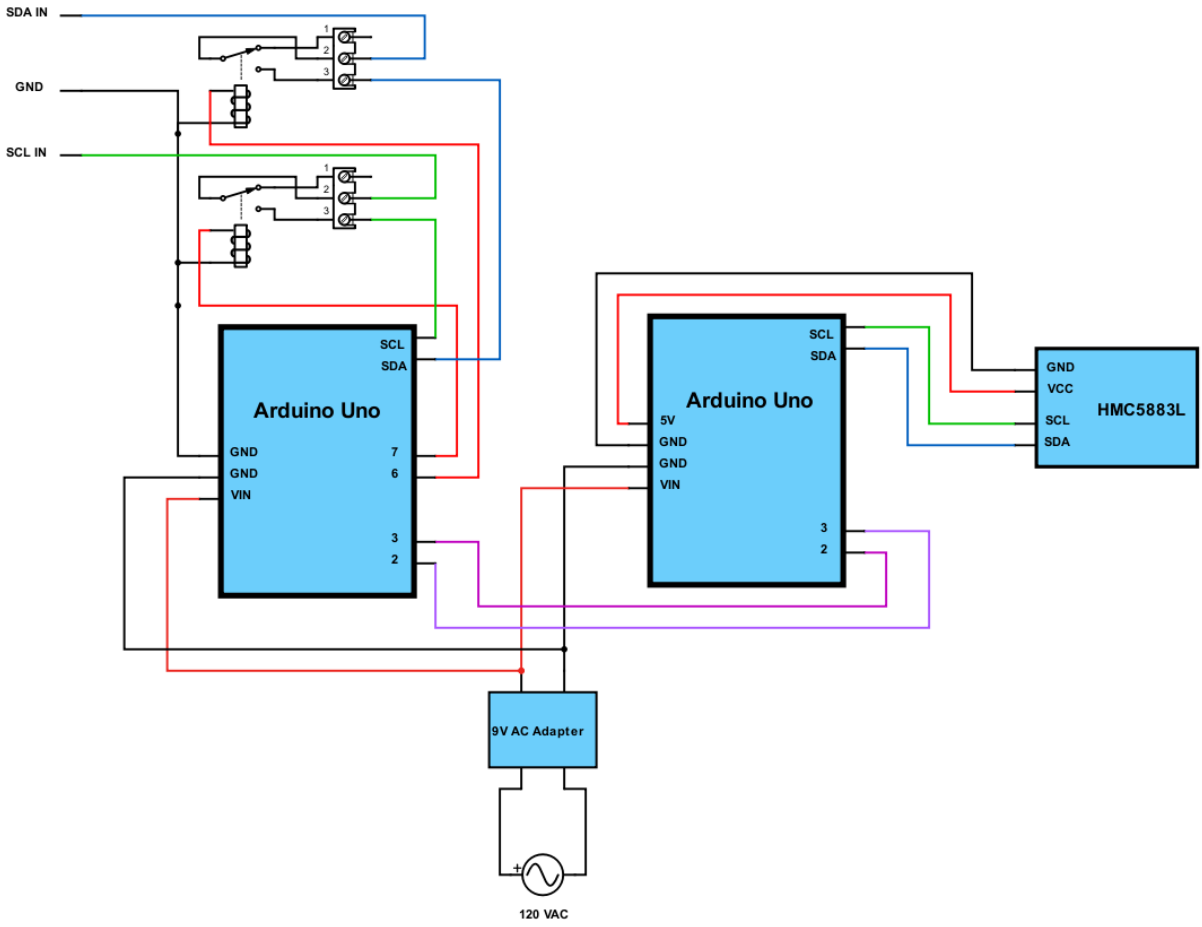

Figure 14. Schematic of digital compass node detailing all connections.

The ‘Digital Compass Module Arduino’ program begins by initializing software serial communication for digital pins 2 and 3, as RX and TX, respectively, as well as joining the I^2C bus connected to the digital compass module as a “master” device. The program then checks for the availability of the digital compass module on the I^2C bus and, if connected, displays details about the sensor in the serial monitor (if available). The program then requests magnetic vector information from the digital compass module and uses the ‘Y’ and ‘X’ vectors to calculate the current compass bearing angle. The local magnetic declination is then added to this value and the bearing angle is checked for wrap due to the addition of this declination angle. The compass

bearing angle is then converted into degrees and transferred to the 'Digital Compass Slave Sender' Arduino Uno (left in Figure 14) over the software serial communication protocol by dividing the value into high and low bytes over the two-wire connection.

The 'Digital Compass Slave Sender' program begins by initializing software serial for digital pins 2 and 3, as RX and TX, respectively, as well as joining the monitoring system's I^2C bus as a "slave" device. Additionally, all digital output pins connected to the stackable relay shield are kept 'HIGH' to keep the system from becoming unresponsive if this node loses power. When the master controller node requests data from this node the program converts the high and low bytes its constantly receiving from the 'Digital Compass Module Arduino' board back into an integer value containing the center pivot compass bearing angle. The value is then prepared for transfer over the I^2C bus to the master controller node by being split into a 2-byte array. If the bearing angle is greater than 255, the maximum integer value of one byte, the first byte transferred contains the value 255 while the second byte contains the difference between the bearing angle and 255. If the bearing angle is less than or equal to 255, the first byte contains the bearing angle and the second byte contains a value of 0. This allows the program to successfully meet the limitations of I^2C data transfer and only requires the master controller node to calculate the sum of the array to receive the center pivot compass bearing angle.

Testing Setup

To evaluate the effectiveness of the monitoring system, it was connected to a demonstration center pivot located behind Seaton Hall on the campus of Kansas State University. The demonstration center pivot, Figure 15, consisted of a single span with eight detachable drop hoses and in-canopy sprinklers. Additionally, the span could pivot around a hinge which mounted the demonstration center pivot to a vertical pole.

Figure 15. Demonstration center pivot used to test system.

Eight sprinkler nodes were constructed and attached to the span above each drop hose. Wires from each sprinkler node extended down to connect the magnetic reed switch placed above the in-canopy sprinkler to the rest of the circuit. At the end of the pivot span, the digital compass node and module were attached ensuring that the digital compass module was in line with the center pivot span to guarantee that it accurately measured the center pivot's compass bearing angle (Figure 16). All nodes were connected using 18-gauge solid core. The in-canopy sprinklers were attached to the drop hose via a hose quick connect (detachment point in Figure 17) to allow for quick detachment during system testing and the magnet counterpart for the magnetic reed switch was tightly secured to ensure that any detachment was registered by the system.

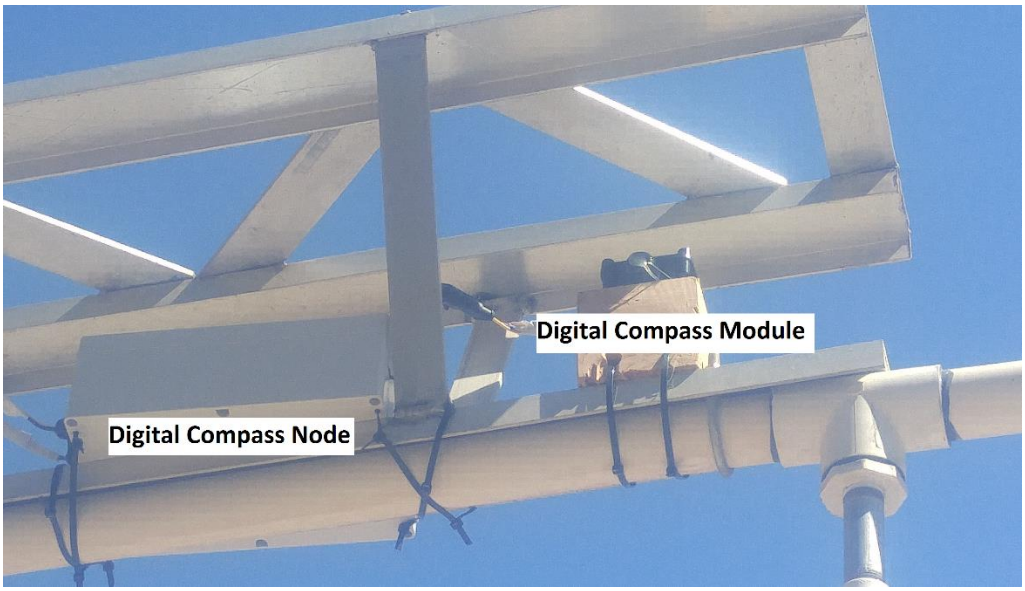

Figure 16. Digital compass node and module in their respective weatherproof housing.

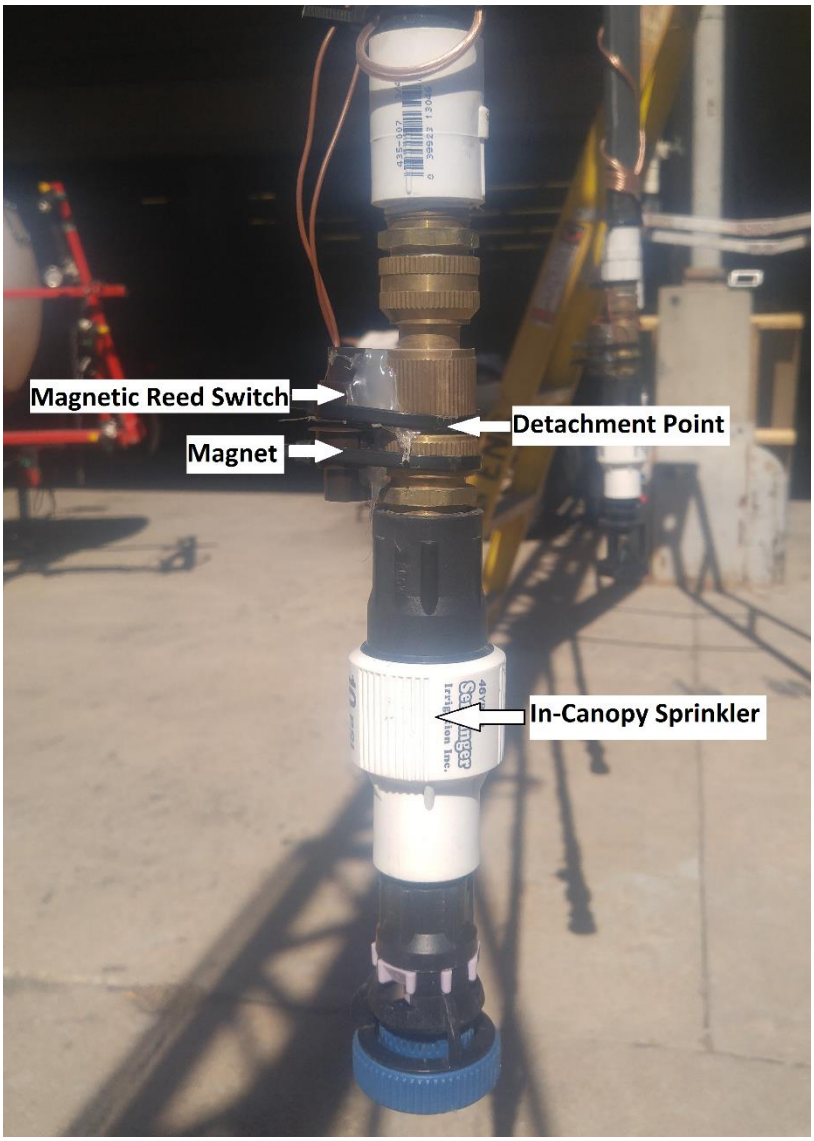

Figure 17. Test in-canopy sprinkler setup.

Chapter 4 - Results and Discussion

Detection of Detached In-Canopy Sprinklers

To guarantee that the system would detect any in-canopy sprinkler loss, testing trials were performed by detaching an in-canopy sprinkler from the drop hose and waiting for a system response. Trials were considered a success if the system sent an SMS text message reporting that the detached in-canopy sprinkler was missing. Two sets of trials were performed, the first set involved detaching each in-canopy sprinkler in triplicate while the second set repeated the same number of trials but used a random number generator to determine which in-canopy sprinkler was detached. Both sets of trials reported a perfect success rate ensuring the design will consistently inform the user of the correct in-canopy sprinkler when it becomes detached from the drop hose. The full results from both sets of trials can be found in Appendix E.

Detection of Detached Drop Hoses

To guarantee that the system would detect any detached drop hose, trials were conducted by detaching the wires connecting the drop hose to the corresponding sprinkler node (Figure 18) and waiting for a system response. The intention of these trials was to simulate the tested drop hose detaching from the pivot span and falling to the ground. Trials were considered a success if the system sent an SMS text message reporting that the in-canopy sprinkler connected to the detached drop hose was missing. Two sets of trials were performed, the first set involved detaching each drop hose in triplicate while the second set repeated the same number of trials but used a random number generator to determine which drop hose was detached. Both sets of trials reported a perfect success rate ensuring the design will consistently inform the user of the correct in-canopy sprinkler when it becomes detached from the system as a result of the drop hose

becoming detached from the center pivot span. The full results from both sets of trials can be found in Appendix F.

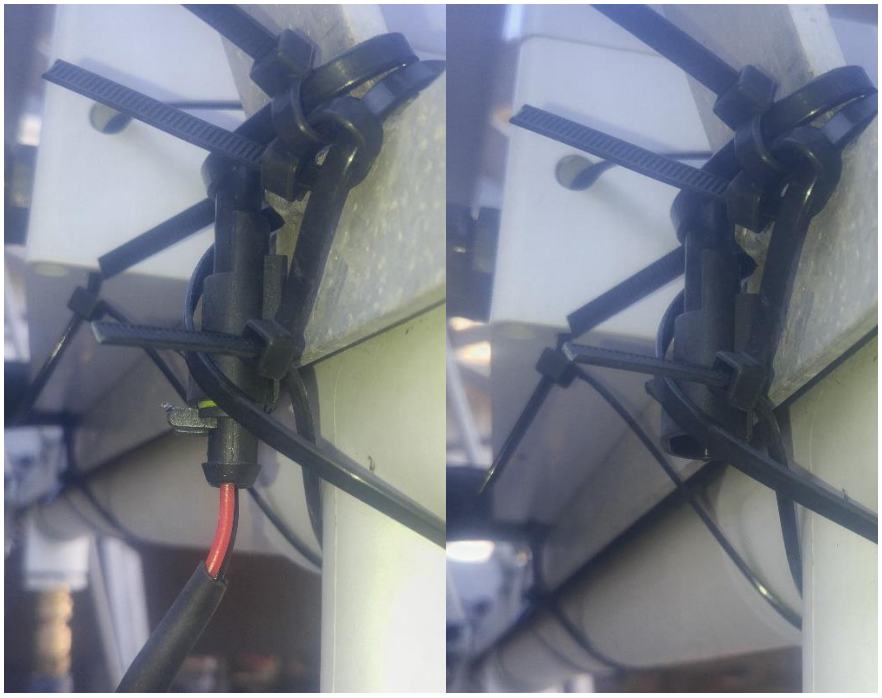

Figure 18. Wires connecting the drop hose to the sprinkler node connected (left) and disconnected (right) while testing the detection of detached drop hoses.

Detection of Disconnected Sprinkler Nodes

To ensure that the system would accurately relay any issues with the sprinkler nodes to the user, two sets of trials were performed. The first set involved disconnecting each sprinkler node from the I^2C bus in triplicate and recording the system response, while the second set of trials repeated the same number of trials but used a random number generator to determine which sprinkler node was tested. These trials simulated a sprinkler node losing power or having its communication wires severed. A trial was considered a success if the system sent an SMS text message to the user detailing that the disconnected sprinkler node was unavailable and required attention. Both sets of trials reported a perfect success rate providing the system with an

additional layer of assisted troubleshooting. The full results from these trials can be found in Appendix G.

Locating Missing In-Canopy Sprinklers

To check the accuracy of the system generated coordinates of each detached in-canopy sprinkler, six trials were conducted. Each trial consisted of rotating the demonstration center pivot to a unique compass bearing and detaching all in-canopy sprinklers before recording the system generated coordinates. The compass bearing angle, as measured by the system, was determined by sending an SMS text message of "Compass" to the system and recording the response. The system measured bearing angle as compared to the actual bearing angle can be seen below in Table 3. The system generated coordinates were then compared to a calculated set of coordinates which were determined by measuring the compass bearing of the demonstration center pivot using a compass and accounting for local magnetic declination. Since the demonstration center pivot was limited to rotating between approximately 150 and 190 degrees, the trials were conducted at the following system measured compass bearing angles: 150, 156, 169, 174, 185, and 189 degrees. Figure 19, below, depicts the system generated and calculated coordinates comparison for each trial with an 'X' illustrating system generated coordinates and a '+' illustrating calculated coordinates for each in-canopy sprinkler. The measured compass bearing angles of 150, 156, 169, 174, 185, and 189 were represented by black, purple, yellow, orange, blue, and red, respectively, in Figure 19. The complete results of calculated coordinates for each trial as well as individual trial maps can be found in Appendices I and J.

Table 3. System sensed vs. measured compass bearing angles.

Trial No.	System Measured Bearing Angle	Actual Bearing Angle
1	150°	153°
2	156°	159°
3	169°	172°
4	174°	175°
5	185°	186°
6	189°	188°

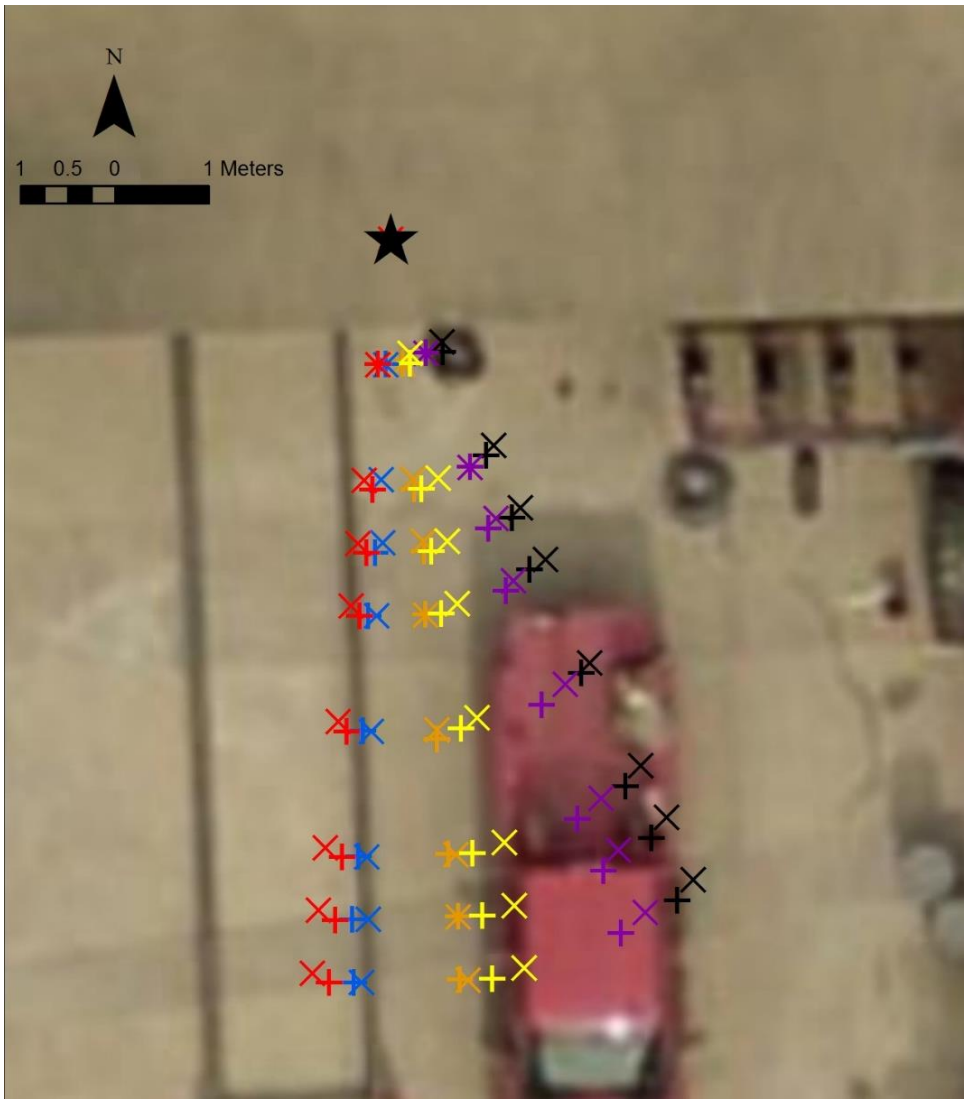

Figure 19. System generated vs. calculated coordinates of detached in-canopy sprinklers for all tested compass bearing angles.

The system generated location of any detached in-canopy sprinkler was a maximum of 0.36 meters away from the calculated location at any time. Extrapolating this error in coordinates to a quarter section center pivot, with a span of approximately 400 meters, equates to the detached in-canopy nozzle being located within a 15-meter radius of the system generated coordinates if the in-canopy sprinkler was located at the end of the center pivot span. The closer to the center the detached in-canopy sprinkler was originally attached along the center pivot

span, the closer the actual location of the sprinkler is to the system generated coordinates. The observed system error is most likely caused by improperly calibrating for local magnetic declination and the digital compass module being unable to compensate for tilt.

Testing User Interface Design

To ensure that the system accurately responded to received SMS text messages with the proper system information as detailed in Table 2, trials were conducted by sending an SMS text message to the system and checking the accuracy of the response. A trial was considered a success if the system replied to the sent SMS text message with the correct information. Two sets of trials were performed, the first set involved sending each possible message in triplicate while the second set repeated the same number of trials but used a random number generator to determine which message was sent to the system. Both sets of trials recorded only successes, ensuring that the system will accurately reply to a user's SMS text message if it is in the format detailed in Table 2. The complete results of these trials can be seen in Appendix H.

Cost Analysis

The total cost of the system, with the maximum number of sprinkler nodes of 111, would be $7952.59 or $71.65 per in-canopy sprinkler monitored. A breakdown of the cost of individual components can be seen below in Table 4.

Table 4. Cost analysis of in-canopy sprinkler monitoring system.

Master Controller Node			
Component	Unit Cost	Quantity	Component Cost
Arduino MKR GSM 1400	$ 69.00	1	$ 69.00
Arduino Uno	$ 22.00	1	$ 22.00
Bi-Directional Logic Level Converter	$ 1.10	1	$ 1.10
Weatherproof Housing	$ 16.99	1	$ 16.99
5V Dual USB AC Adapter	$ 3.49	1	$ 3.49
		Node Cost	$ 112.58

Digital Compass Node			
Component	Unit Cost	Quantity	Component Cost
Arduino Uno	$ 22.00	2	$ 44.00
Arduino Relay Shield	$ 1.10	1	$ 1.10
Weatherproof Housing	$ 16.99	1	$ 16.99
Digital Compass Module	$ 31.99	1	$ 31.99
9V AC Adapter	$ 5.99	1	$ 5.99
		Node Cost	$ 100.07

Sprinkler Node			
Component	Unit Cost	Quantity	Component Cost
Arduino Uno	$ 22.00	1	$ 22.00
Arduino Relay Shield	$ 9.99	1	$ 9.99
Weatherproof Housing	$ 16.99	1	$ 16.99
N/O Magnetic Reed Switch	$ 2.10	1	$ 2.10
9V AC Adapter	$ 5.99	1	$ 5.99
		Node Cost	$ 57.07

Miscellaneous			
Component	Unit Cost	Quantity	Component Cost
Wire (50 feet)	$ 18.71	27	$ 505.17
Marine Connectors	$ 2.25	400	$ 900.00
		Total Misc	$ 1,405.17

The estimated cost of the system can be significantly reduced in a variety of ways. The first cost reduction method would be to substitute the Arduino boards with off-brand clones. Since the Arduino corporation makes all of their products open source, cheaper clones of their boards are commonly sold. For example, clones of the Arduino Uno board are commonly sold at half the price of the original. The second method to reduce the cost of the system is to increase the number of in-canopy sprinklers each sprinkler node monitors. This would allow the system to not only monitor additional in-canopy sprinklers, but reduce the number of sprinkler nodes that must be constructed which reduces the total costs from these nodes. The last cost reduction method would be to substitute all electronic components in each node with mass produced custom PCBs (printed circuit boards). These custom boards, when designed and purchased in bulk, can significantly reduce the cost of each node by replacing the need to purchase individual components for each node.

Additional Benefits

Two additional benefits to the user were accidentally created throughout the design of this system. The first is the ability of the system to warn the user if wire theft has occurred. By designing the master controller node to alert the user if sprinkler nodes have had their communication lines disconnected or have lost power during normal system use, the system can indirectly detect wire theft. If wires connecting the sprinkler nodes to the master controller node are cut, the user will receive a near immediate response from the system detailing which sprinkler nodes are unavailable. This feature will give users peace of mind that their system hasn’t been stolen and is still operational.

The second benefit is the ability to remotely sense center pivot rotational direction. With the incorporation of the ‘Compass’ message option in the user interface, users can request the

current center pivot compass bearing angle. If a user wishes to remotely sense the direction the center pivot system is rotating, they would simply need to call this function a few minutes apart and compare the responses. If the bearing angle increases between the two messages then the center pivot is moving in a clockwise direction. Inversely, if the bearing angle decreases between the two messages then the center pivot is moving in a counter-clockwise direction. These relationships hold true with the exception of if the center pivot crosses North (0°) between the two messages.

Future Work

Overall, this prototype system has proven to be quite effective in accurately detecting when and where an in-canopy sprinkler detaches from the drop hose on a center pivot. However, several improvements could be made to the system to improve its resiliency and accuracy to ensure that this prototype is ready for deployment on a full center pivot system.

The first improvement involves the incorporation of a watchdog timer. The purpose of a watchdog timer is to reset a computerized system if an error arises that causes the system to become unresponsive or prevents the system from carrying out normal functions. This component would need to be embedded into the master controller node and would check for signals from the node. Anytime a signal is received the watchdog timer resets its countdown. If the master controller node fails to check in before the countdown elapses, the watchdog timer would reset the master controller node and restart the countdown. The watchdog timer would allow the system an additional level of automatic repair preventing the user from needing to physically restart the system anytime a system breaking error occurs.

Currently, the system has the capacity for up to 112 “slave” devices on the I^2C bus. If the only one sprinkler node is used to monitor each in-canopy sprinkler the number of sprinkler

nodes may surpass this limit. The solution to this limitation would be to adapt the current sprinkler node design to monitor multiple in-canopy sprinklers. This could be accomplished by creating a unique response by each sprinkler node to represent the detached in-canopy sprinkler instead of the current method of responding with either a value of a '1' or '0'. Additionally, the code of the master controller node would need to be modified to properly analyze the incoming messages and match it to a missing in-canopy sprinkler. If each sprinkler node monitors two in-canopy sprinklers, the maximum number of in-canopy sprinklers the system can monitor immediately doubles. This relationship expands linearly and can be adjusted to meet the specific number of in-canopy sprinklers found on the center pivot system without spacing sprinkler nodes too far apart.

The error found in the system generated coordinates of detached in-canopy sprinklers can primarily be accounted by error in sensing the correct compass bearing angle. To reduce this system error the current digital compass module, HMC5883L, should be replaced with a tilt-compensated version. Most error observed when testing the digital compass module occurred when the device was rotated off horizontal. Therefore, incorporating a tilt-compensated digital compass will ensure more accurately sensed compass bearing angles when the system is calculating the geographic coordinates of detached in-canopy sprinklers.

Chapter 5 - Summary and Conclusions

Modern center pivot irrigation systems incorporate in-canopy sprinkler packages in order to increase water efficiency and application uniformity. However, if these in-canopy sprinklers detach from the center pivot span, the uniformity of the irrigation system is directly impacted. The accumulative impact of these non-uniform irrigation events can decrease overall crop yield for the field. These detachments are generally the result of the in-canopy sprinkler entangling itself in crop biomass since these in-canopy sprinklers operate within the crop canopy. Additionally, the current method for detecting these detachments requires manual inspection of the center pivot span which can be time-consuming and labor intensive. A monitoring system could immediately alert users when a detachment occurs allowing them to quickly repair the missing in-canopy sprinkler and prevent uniformity issues.

A prototype in-canopy sprinkler monitoring system for center pivot irrigation systems was created with the objective of accurately detecting and alerting the user when and where an in-canopy sprinkler is detached from the center pivot span. The system utilized three major components: a master controller node, a series of sprinkler nodes, and a digital compass node. The system monitors all connected in-canopy sprinklers and communicates detachment and system information to the user through SMS text messages sent over a cellular network. Trials designed to test the performance of the system concluded that the system accurately meets its objective with the additional benefit of incorporating a simple and interactive user interface.

With a prototype monitoring system developed, future work is necessary to improve and deploy the system on a full center pivot irrigation system. These improvements will increase the resiliency and accuracy of the system and include incorporating a watchdog timer, expanding the

sprinkler nodes to monitor additional in-canopy sprinklers, and upgrading the current digital compass module to a tilt-compensated version.

References

Arduino. (2018a). *Arduino Uno Rev3*. Retrieved from Arduino: https://store.arduino.cc/usa/arduino-uno-rev3.

Arduino. (2018b). *Wire Library*. Retrieved from Arduino: https://www.arduino.cc/en/Reference/Wire.

Kranz, W., Glaser, R., Ross, G., & Rogers, D. (2012). Center Pivot Sprinkler Nozzle Replacement and Maintenance. *Proc. 24th Annu. Central Plains Irrigation Conf.,* pp. 102-110. Colby, KS: CPIA

Lamm, F., Howell, T., & Bordovsky, J. (2006). Concepts of In-Canopy and Near-Canopy Sprinkler Irrigation. *2006 World Environmental and Water Resources Congress*, 2006 World Environmental and Water Resources Congress.

Leens, F. (2009). An introduction to I^2C and SPI protocols: *Instrumentation & Measurement Magazine, IEEE,* 12(1), 8-13.

Melvin, S. & Martin, D. (2018). In-canopy vs. above-canopy sprinklers, which is better suited to your field?. *Proc. 30th Annu. Central Plains Irrigation Conf.,* pp. 157-165. Colby, KS: CPIA.

Monk, S. (2014). *Programming Arduino Next Steps*. McGraw-Hill Education.

Monk, S. (2016). *Programming Arduino* (2nd ed.). McGraw-Hill Education.

New, L., & Fipps, G. (2017). Center pivot irrigation. College Station, TX: The Texas A&M University System. Retrieved from http://aglifesciences.tamu.edu/baen/wp-content/uploads/sites/24/2017/01/B-6096-Center-Pivot-Irrigation.pdf

NRCS (2005). Utilizing Center Pivot Sprinkler Irrigation Systems to Maximize Water Savings. Washington, DC: USDA-NRCS. Retrieved from http://cotton.tamu.edu/Irrigation/NRCS%20Center%20Pivot%20Irrigation.pdf.

Perlman, H. (2017). *Irrigation water use.* Retrieved from USGS: https://water.usgs.gov/edu/wuir.html.

Robinson, A. (1995) *Elements of Cartography* (6th ed.). New York: Wiley.

Rudnick, D (2017). *Sprinkler Irrigation System Maintenance for Improved Uniformity and Application Efficiency*. Retrieved from University of Nebraska-Lincoln Institute of Agricultural and Natural Resources: https://water.unl.edu/article/agricultural-irrigation/sprinkler-irrigation-system-maintenance-improved-uniformity-and-.

Appendix A - Program for Master Controller Node

```
/*
  Master Controller Node
  by Aaron Akin

  Operates as user interface and system controller for in-canopy sprinkler monitoring system.
  Monitors individual sprinkler nodes over an I2C bus and informs end-user of sprinkler loss and
  approximate geographic coordinates of drop site.
  Responds to user's SMS text messages and generates a system response.

  Required Libraries: MKRGSM, Wire

  Last Edited March 15, 2018
*/

#include <Wire.h>
#include <MKRGSM.h>
#define PINNUMBER "" //SIM Card PIN number, if no PIN number is needed leave as ""

GSM gsmAccess;
GSM_SMS sms;

char remoteNumber[20] = "15555555555"; //phone number with country and area code EX: 1-555-555-5555
char senderNumber[20]; //array to hold number SMS is received from


const int TOTAL_NODES = 8; //Max number of sprinkler nodes
int headingDegrees;
float MtoDD = 7.87; //Conversion factor from 1 meter to 10^-5 Decimal Degrees In Long Integer format
long centerLON = -96582840; //Longitude of center pivot with decimal point removed. EX: -96.582836 is centerLAT = -96582836
long centerLAT = 39189826; //Latitude of center pivot with decimal point removed. EX: 39.189824 is centerLON = 39189824
int myArray[2]; //Array to hold incoming bytes for calculating center pivot compass bearing
int sprinklerArray[8]; //Array to hold sprinkler node availability
float nodeSpacing[TOTAL_NODES] = {1.55, 3.06, 3.82, 4.61, 6.06, 7.58, 8.34, 9.11}; //Array of radius(m) of each in-canopy sprinkler.



void setup() {
  Wire.begin();        // Join I2C Bus
  Serial.begin(9600);  // start serial for output
```

```
  //while (!Serial); //Use only when joined by USB to computer for debugging
  Serial.println("In-Canopy Sprinkler Monitoring System by Aaron Akin (2018)");
  boolean notConnected = true;
  while (notConnected)
  {
    if (gsmAccess.begin(PINNUMBER) == GSM_READY)
      notConnected = false;
    else
    {
      Serial.println("Not Connected");
      delay(1000);
    }
  }
  Serial.println("GSM initialized");
}

void systemCheck() {
  Serial.println("systemCheck");
  sms.beginSMS(remoteNumber);
  sms.print("In-Canopy Sprinkler Monitoring System");
  sms.print("\nAaron Akin, 2018");
  sms.print("\n");
  sms.print("\nSystem is operational.");
  sms.endSMS();
  delay(5000);
}

void systemInfo() {
  Serial.println("systemInfo");
  sms.beginSMS(remoteNumber);
  sms.print("\nRespond with the following for system information:");
  sms.print("\n");
  sms.print("\nSprinkler = Check Sprinkler Node Availability");
  sms.print("\nCompass = Get Center Pivot Compass Bearing");
  sms.endSMS();
  sms.beginSMS(remoteNumber);
  sms.print("\nOne = Pause System 1 Minute");
  sms.print("\nFive = Pause System 5 Minutes");
  sms.print("\nHelp = Repeat This Menu");
  sms.endSMS();
}

void sprinklerStatus() {
  delay(5000);
  int b = TOTAL_NODES - 1;
  int arrayTotal = 0;
```

```
 for (int i = 0; i <= b; i++) { //Calculate total of sprinklerArray[8]
   arrayTotal += sprinklerArray[i];
 }
 Serial.print("Array Total: ");
 Serial.println(arrayTotal);
 sms.beginSMS(remoteNumber);
 sms.print("Sprinkler Nodes Available: ");
 Serial.print("Sprinkler Nodes Available: ");
 if (arrayTotal > 0) { //If sprinklerArray[8] has values totaling more than 0 send user SMS listing
all available sprinkler nodes
   for (int i = 0; i <= b; i++) {
     if (sprinklerArray[i] > 0) {
       sms.print(sprinklerArray[i]);
       sms.print(", ");
       Serial.print(sprinklerArray[i]);
       Serial.print(", ");
     }
   }
 }
 else { //Else send user SMS informing them that no sprinkler nodes have been detected
   sms.print("None Detected");
   Serial.print("None Detected");
 }
 sms.endSMS();
 sms.flush();
 delay(5000);
 Serial.println();
 communicationLineCheck(); ///Check for any sprinkler nodes that are disconnected from the
I2C bus or are unpowered
}

void loop() {
  while (millis() < 30000) {
    Serial.println("SMS Flush");
    char c;
    if (sms.available()) {
      Serial.println("Message received from:");

      // Get remote number
      sms.remoteNumber(senderNumber, 20);
      Serial.println(senderNumber);

      // An example of message disposal
      // Any messages starting with # should be discarded
      if (sms.peek() == '#') {
        Serial.println("Discarded SMS");
```

```
      sms.flush();
     }

     // Read message bytes and print them
     if (c = sms.read()) {
      Serial.println(c);
     }
     Serial.println("\nEND OF MESSAGE");
     // Delete message from modem memory
     sms.flush();
     Serial.println("MESSAGE DELETED");
    }
    delay(1000);
  }
  delay(5000);
  Wire.requestFrom(9, 2);    // Request 2 bytes from I2C address #9 (node reading magnometer)
  while (Wire.available()) {
   for (int i = 0; i <= 1; i++) { //fill array with each byte of data
    int b = Wire.read();
    myArray[i] = b;
   }
  }
  headingDegrees = myArray[0] + myArray[1];
  Serial.print("Heading: ");
  Serial.print(headingDegrees); //Center pivot compass bearing
  Serial.println(char(176));
  for (int i = 1; i <= TOTAL_NODES; i++) { //Fill sprinklerArray[8] with integer values
corresponding to I2C address
   if (Wire.requestFrom(i, 1)) {    // Request 1 byte from slave device #n
    if (Wire.available()) {
     int b = i - 1;
     sprinklerArray[b] = i;
    }
   }
  }
  checkSMS(); //Check for incoming SMS subroutine
  checkSprinklers(); //Check all in-canopy sprinklers subroutine
  memset(sprinklerArray, 0, sizeof(sprinklerArray)); //Reset sprinklerArray[8]
  delay(5000);
}

void checkSprinklers() {
  delay(5000);
  Serial.println("checksprinklers subroutine");
  for (int i = 1; i <= TOTAL_NODES; i++) {
   if (Wire.requestFrom(i, 1)) {    // request 1 byte from slave device #i
```

```
if (Wire.available()) { // slave may send less than requested
  int c = Wire.read(); // receive a byte as an integer
  if (c > 0)
  {
    float headingRadians = headingDegrees * PI / 180;  //Convert center pivot bearing from
     degrees to radians
    float sprinklerRadius = nodeSpacing[i - 1] * MtoDD; //Convert sprinkler radius in meters
     to decimal degrees
    long polarLAT = cos(headingRadians) * sprinklerRadius; //Convert sprinkler location
     from polar to rectangular
    long polarLON = sin(headingRadians) * sprinklerRadius;
    long sprinklerLAT = polarLAT + centerLAT; //Calculate sprinkler latitude as long integer
    Serial.print("stringLAT");
    Serial.println(sprinklerLAT);
    long sprinklerLON = polarLON + centerLON; //Calculate sprinkler longitude as long
     integer
    Serial.print("stringLON");
    Serial.println(sprinklerLON);
    //Convert sprinkler lattitude and longitude into strings
    String stringLAT = String(sprinklerLAT);
    String stringLAT2 = String(sprinklerLAT);
    String stringLON = String(sprinklerLON);
    String stringLON2 = String(sprinklerLON);
    //Remove portions of each string and then appen them to include decimal point in
     appropriate spot for decimal degrees
    stringLAT.remove(2);
    stringLAT2.remove(0, 2);
    stringLON.remove(3);
    stringLON2.remove(0, 3);
    String DDLAT = stringLAT + "." + stringLAT2;
    String DDLON = stringLON + "." + stringLON2;
    Serial.print("DDLAT: ");
    Serial.println(DDLAT);
    Serial.print("DDLON: ");
    Serial.println(DDLON);
    //Send SMS to inform user that sprinkler #i is missing
    Serial.print("Sprinkler #");
    Serial.print(i);
    Serial.print(" is missing please replace immediately.");
    Serial.println();
    //Send SMS to inform user of sprinkler #i's approximate lattitude and longitude
    sms.beginSMS(remoteNumber);
    sms.print("Sprinkler #");
    sms.print(i);
    sms.print(" is missing please replace immediately.");
    sms.print("\nApproximate geographic location: ");
```

```
          sms.print(DDLAT);
          sms.print(", ");
          sms.print(DDLON);
          sms.endSMS();
          Serial.println("SMS Sent");
          delay(5000);
        }
      }
    }
  }
  communicationLineCheck(); //Check for any sprinkler nodes that are disconnected from the
I2C bus or are unpowered
}

void communicationLineCheck() {
  delay(5000);
  for (int i = 1; i <= TOTAL_NODES; i++) {
    Wire.requestFrom(i, 1);    // Request 1 byte from I2C address #i
    if (!Wire.available()) {  // If I2C address is not available send SMS to remote number
      sms.beginSMS(remoteNumber);
      sms.print("Sprinkler Node #");
      sms.print(i);
      sms.print(" has communication line disconnected or is unpowered.");
      sms.endSMS();
      Serial.print("Sprinkler Node #");
      Serial.print(i);
      Serial.println(" has communication line disconnected or is unpowered.");
      delay(5000);
    }
  }
}

void checkSMS() {
  Serial.println("checkSMS subroutine");
  char c;

  // If there are any SMSs available
  if (sms.available()) {
    Serial.println("Message received from:");

    // Get remote number
    sms.remoteNumber(senderNumber, 20);
    Serial.println(senderNumber);
    c = sms.read();
    Serial.println(char(c));
    if (c == 'S') sprinklerStatus();
```

```
  if (c == 'H') {
    systemCheck();
    systemInfo();
  }
  if (c == 'C') {
    Serial.print("Center Pivot Compass Bearing: ");
    Serial.println(headingDegrees);
    sms.beginSMS(remoteNumber);
    sms.print("Center Pivot Compass Bearing: ");
    sms.print(headingDegrees);
    sms.print(" degrees");
    sms.endSMS();
  }
  if (c == 'O') { //Pause the system for 1 minute when the character "O" is received
    Serial.println("pause 1 minute");
    sms.beginSMS(remoteNumber);
    sms.print("System Paused for 1 Minute");
    sms.endSMS();
    Serial.println("sms sent");
    delay (60000);
  }
  if (c == 'F') { //Pause the system for 5 minutes when the character "F" is received
    Serial.println("pause 5 minutes");
    sms.beginSMS(remoteNumber);
    sms.print("System Paused for 5 Minutes");
    sms.endSMS();
    Serial.println("sms sent");
    delay (300000);
  }
  Serial.println("\nEND OF MESSAGE");
  sms.flush();  //Delete the SMS message from modem memory
  Serial.println("MESSAGE DELETED");
 }
 delay(1000);

}
```

Appendix B - Program for Sprinkler Nodes

```
/*
Sprinkler Node
Aaron Akin, 2018

Waits for request over I2C bus and responds with either a 1 or 0 depending on the state of a magnetic reed
switch.

Required Libraries: Wire

Last Edited: March 9, 2018
*/

#include <Wire.h>
int nozzle = 2; // I2C address (assigned as Sprinkler #)
int val = 1;
int val2 = 0;
int EMR = 2; //Digital pin connected to magnetic reed switch

void setup() {
  Wire.begin(nozzle);                //Join I2C Bus with assigned address
  Wire.onRequest(requestEvent); // register event
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(EMR, INPUT_PULLUP);
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
}

void loop() {
  delay(100);
}

void requestEvent() {
  int state = digitalRead(EMR);
  if (state == HIGH) {
  Wire.write(val); //If sprinkler is missing respond send integer value of 1
  }
  else {
    Wire.write(val2); //Otherwise respond with integer value of 0
  }
}
```

Appendix C - Program for Digital Compass Slave Sender Arduino

```
/*
Digital Compass Slave Sender
by Aaron Akin

On request from master controller node, gets center pivot compass bearing from Digital Compass Module Arduino.
Splits center pivot compass bearing into two bytes for transfer to master controller node over I2C bus.

Required Libraries: Software Serial, Wire

Last Edited March 11, 2018
*/

#include <SoftwareSerial.h>
SoftwareSerial reciever(2, 3); //RX, TX for software serial communication
#include <Wire.h>
volatile byte sendArray[2]; //Array to send two bytes over I2C
int headingDegrees; //Center pivot compass bearing as received over software serial communication

void setup() {
  Serial.begin(9600);
  reciever.begin(9600);
  Wire.begin(9);                // Join I2C bus with address #9
  Wire.onRequest(requestEvent); // register event
  //Keep all digital pins controlling relays HIGH while Arduino is powered
  //This allows for the Arduino to be disconnected from the I2C bus when power is lost
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  digitalWrite(7, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(4, HIGH);
}

void loop() {
  delay(100);
}

void requestEvent() {  //On request from master controller node
  if (reciever.available() > 1) //Get compass bearing from Digital Compass Module Arduino over software serial communication
  {
    byte h = reciever.read();
    byte l = reciever.read();
    headingDegrees = (h << 8) + l;
  }
```

```
  if (headingDegrees > 255) { //If headingDegrees is greater than 255 split value into 2 bytes for transfer
over I2C bus
    sendArray[0] = headingDegrees - 255;
    sendArray[1] = 255;
    Wire.write((byte*)sendArray, 2);
  }
  else { //Else transfer headingDegrees as first byte and a value of zero as the second byte over I2C bus
    sendArray[0] = headingDegrees;
    sendArray[1] = 0;
    Wire.write((byte*)sendArray, 2);
  }
}
```

Appendix D - Program for Digital Compass Module Arduino

```
/*
Digital Compass Module Arduino
by Aaron Akin

Requests 3-axis magnetic vector information from HMC5883L Digital Compass Module over I2C bus.
Calculates center pivot compass bearing in degrees from magnetic vector information.
Communicates center pivot compass bearing to Digital Compass Slave Sender Arduino over software
serial communication.

Required Libraries: Software Serial, Wire, Adafruit_Sensor, Adafruit_HMC5883_U

Last Edited: March 11, 2018
*/

#include <Wire.h>
#include <SoftwareSerial.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
SoftwareSerial sender(2, 3); //RX, TX for software serial communication


// Assign a unique ID to the sensor
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);

void displaySensorDetails(void)
{
  sensor_t sensor;
  mag.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" uT");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" uT");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" uT");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}

void setup(void)
{
  Serial.begin(9600);
  sender.begin(9600);
  Serial.println("HMC5883 Magnetometer Test");
  Serial.println("");
  if(!mag.begin())
  {
```

```
    Serial.println("No HMC5883 Detected");
    while(1);
  }
  displaySensorDetails(); //Display basic information about the sensor
}

void loop(void)
{
  //Request magnetic vector information from magnetometer
  sensors_event_t event;
  mag.getEvent(&event);
  //Display the results (magnetic vector values are in micro-Tesla (uT))
  Serial.print("X: "); Serial.print(event.magnetic.x); Serial.print("  ");
  Serial.print("Y: "); Serial.print(event.magnetic.y); Serial.print("  ");
  Serial.print("Z: "); Serial.print(event.magnetic.z); Serial.print("  ");Serial.println("uT");
  float heading = atan2(event.magnetic.y, event.magnetic.x);
  float declinationAngle = 0.05; //Magnetic declination in radians for Manhattan, KS
  heading += declinationAngle;
  //Correct for when signs are reversed
  if(heading < 0)
    heading += 2*PI;
  //Check for wrap due to addition of declination
  if(heading > 2*PI)
    heading -= 2*PI;
  // Convert heading from radians to degrees
  float headingDegrees = heading * 180/M_PI;
  Serial.print("Heading (degrees): ");
  Serial.println(headingDegrees);
  //Split headingDegrees into high and low bytes
  int reading = headingDegrees;
  byte h = highByte(reading);
  byte l = lowByte(reading);
  //Write high and low bytes to Digital Compass Slave Sender Arduino over software serial
communication
  sender.write(h);
  sender.write(l);
  delay(1000);
}
```

Appendix E - Detached In-Canopy Sprinklers Trials

Table 5. Correct SMS text message response received from system for detached in-canopy sprinkler.

Trial No.	Sprinkler Detached	Correct SMS Response Received?
1	1	Yes
2	2	Yes
3	3	Yes
4	4	Yes
5	5	Yes
6	6	Yes
7	7	Yes
8	8	Yes
9	1	Yes
10	2	Yes
11	3	Yes
12	4	Yes
13	5	Yes
14	6	Yes
15	7	Yes
16	8	Yes
17	1	Yes
18	2	Yes
19	3	Yes
20	4	Yes
21	5	Yes
22	6	Yes
23	7	Yes
24	8	Yes

Table 6. Correct SMS text message response from system for randomly detached in-canopy sprinkler.

Trial No.	Sprinkler Detached	Correct SMS Response Received?
1	3	Yes
2	6	Yes
3	7	Yes
4	6	Yes
5	5	Yes
6	3	Yes
7	7	Yes
8	1	Yes
9	2	Yes
10	3	Yes
11	1	Yes
12	8	Yes
13	6	Yes
14	1	Yes
15	8	Yes
16	4	Yes
17	2	Yes
18	7	Yes
19	4	Yes
20	2	Yes
21	8	Yes
22	6	Yes
23	2	Yes
24	1	Yes

Appendix F - Detached Drop Hose Trials

Table 7. Correct SMS text message response received from system for detached drop hose.

Trial No.	Drop Hose Detached	Correct SMS Response Received?
1	1	Yes
2	2	Yes
3	3	Yes
4	4	Yes
5	5	Yes
6	6	Yes
7	7	Yes
8	8	Yes
9	1	Yes
10	2	Yes
11	3	Yes
12	4	Yes
13	5	Yes
14	6	Yes
15	7	Yes
16	8	Yes
17	1	Yes
18	2	Yes
19	3	Yes
20	4	Yes
21	5	Yes
22	6	Yes
23	7	Yes
24	8	Yes

Table 8. Correct SMS text message response received from system for randomly detached drop hose.

Trial No.	Drop Hose Detached	Correct SMS Response Received?
1	6	Yes
2	2	Yes
3	4	Yes
4	3	Yes
5	5	Yes
6	8	Yes
7	3	Yes
8	5	Yes
9	8	Yes
10	5	Yes
11	4	Yes
12	8	Yes
13	4	Yes
14	2	Yes
15	7	Yes
16	8	Yes
17	6	Yes
18	4	Yes
19	2	Yes
20	4	Yes
21	7	Yes
22	4	Yes
23	6	Yes
24	8	Yes

Appendix G - Disconnected Sprinkler Node Trials

Table 9. Correct SMS text message response received from system for disconnected sprinkler node.

Trial No.	Sprinkler Node Disconnected	Correct SMS Response Received?
1	1	Yes
2	2	Yes
3	3	Yes
4	4	Yes
5	5	Yes
6	6	Yes
7	7	Yes
8	8	Yes
9	1	Yes
10	2	Yes
11	3	Yes
12	4	Yes
13	5	Yes
14	6	Yes
15	7	Yes
16	8	Yes
17	1	Yes
18	2	Yes
19	3	Yes
20	4	Yes
21	5	Yes
22	6	Yes
23	7	Yes
24	8	Yes

Table 10. Correct SMS text message response received from system for randomly disconnected sprinkler node.

Trial No.	Sprinkler Node Disconnected	Correct SMS Response Received?
1	1	Yes
2	4	Yes
3	3	Yes
4	4	Yes
5	7	Yes
6	5	Yes
7	3	Yes
8	5	Yes
9	2	Yes
10	4	Yes
11	5	Yes
12	3	Yes
13	4	Yes
14	5	Yes
15	7	Yes
16	5	Yes
17	3	Yes
18	7	Yes
19	1	Yes
20	5	Yes
21	8	Yes
22	4	Yes
23	6	Yes
24	3	Yes

Appendix H - User Interface Trials

Table 11. Correct SMS text message response received from system for message sent to system.

Trial No.	Message Sent	Correct SMS Response Received?
1	Help	Yes
2	Sprinkler	Yes
3	Compass	Yes
4	One	Yes
5	Five	Yes
6	Help	Yes
7	Sprinkler	Yes
8	Compass	Yes
9	One	Yes
10	Five	Yes
11	Help	Yes
12	Sprinkler	Yes
13	Compass	Yes
14	One	Yes
15	Five	Yes

Table 12. Correct SMS text message response received from system for randomly selected message sent to system.

Trial No.	Message Sent	Correct SMS Response Received?
1	Sprinkler	Yes
2	Compass	Yes
3	Help	Yes
4	Compass	Yes
5	Five	Yes
6	One	Yes
7	Sprinkler	Yes
8	Compass	Yes
9	Sprinkler	Yes
10	Five	Yes
11	Compass	Yes
12	Five	Yes
13	Compass	Yes
14	Help	Yes
15	One	Yes

Appendix I - Tables of System Generated vs. Calculated Coordinates of Detached In-Canopy Sprinklers

Table 13. System generated and calculated coordinates of detached in-canopy sprinklers for sensor bearing of 150 degrees.

	Degrees	Radians
Sensor Bearing:	150	2.62
Measured Bearing:	153	2.67

Pivot Point Longitude:	-96.582840
Pivot Point Latitude:	39.189826

		System Generated Coordinates Using Sensor Bearing		**Calculated Coordinates Using Measured Bearing**	
Sprinkler No.	*Radius (m)*	*Longitude*	*Latitude*	*Longitude*	*Latitude*
1	1.55	-96.582834	39.189816	-96.582834	39.189815
2	3.06	-95.852828	39.189806	-96.582829	39.189805
3	3.82	-96.582825	39.189800	-96.582826	39.189799
4	4.61	-96.582822	39.189795	-96.582824	39.189794
5	6.06	-96.582817	39.189785	-96.582818	39.189784
6	7.58	-96.582811	39.189775	-96.582813	39.189773
7	8.34	-96.582808	39.189770	-96.582810	39.189768
8	9.11	-96.582805	39.189764	-96.582807	39.189762

Table 14. System generated and calculated coordinates of detached in-canopy sprinklers for sensor bearing of 156 degrees.

	Degrees	Radians
Sensor Bearing:	156	2.72
Measured Bearing:	159	2.78

Pivot Point Longitude:	-96.582840
Pivot Point Latitude:	39.189826

		System Generated Coordinates Using Sensor Bearing		**Calculated Coordinates Using Measured Bearing**	
Sprinkler No.	*Radius (m)*	*Longitude*	*Latitude*	*Longitude*	*Latitude*
1	1.55	-96.582836	39.189815	-96.582836	39.189815
2	3.06	-96.582831	39.189804	-96.582831	39.189804
3	3.82	-96.582828	39.189799	-96.582829	39.189798
4	4.61	-96.582826	39.189793	-96.582827	39.189792
5	6.06	-96.582820	39.189783	-96.582823	39.189781
6	7.58	-96.582816	39.189772	-96.582819	39.189770
7	8.34	-96.582814	39.189767	-96.582816	39.189765
8	9.11	-96.582811	39.189761	-96.582814	39.189759

Table 15. System generated and calculated coordinates of detached in-canopy sprinklers for sensor bearing of 169 degrees.

	Degrees	Radians
Sensor Bearing:	169	2.95
Measured Bearing:	172	3.00

Pivot Point Longitude:	-96.582840
Pivot Point Latitude:	39.189826

		System Generated Coordinates Using Sensor Bearing		**Calculated Coordinates Using Measured Bearing**	
Sprinkler No.	*Radius (m)*	*Longitude*	*Latitude*	*Longitude*	*Latitude*
1	1.55	-96.582838	39.189815	-96.582838	39.189814
2	3.06	-96.582835	39.189803	-96.582837	39.189802
3	3.82	-96.582834	39.189797	-96.582836	39.189796
4	4.61	-96.582833	39.189791	-96.582835	39.189790
5	6.06	-96.582831	39.189780	-96.582833	39.189779
6	7.58	-96.582828	39.189768	-96.582832	39.189767
7	8.34	-96.582827	39.189762	-96.582831	39.189761
8	9.11	-96.582826	39.189756	-96.582830	39.189755

Table 16. System generated and calculated coordinates of detached in-canopy sprinklers for sensor bearing of 174 degrees.

	Degrees	Radians
Sensor Bearing:	174	3.04
Measured Bearing:	175	3.05

Pivot Point Longitude:	-96.582840
Pivot Point Latitude:	39.189826

		System Generated Coordinates Using Sensor Bearing		**Calculated Coordinates Using Measured Bearing**	
Sprinkler No.	*Radius (m)*	*Longitude*	*Latitude*	*Longitude*	*Latitude*
1	1.55	-96.582839	39.189814	-96.582839	39.189814
2	3.06	-96.582838	39.189803	-96.582838	39.189802
3	3.82	-96.582837	39.189797	-96.582837	39.189796
4	4.61	-96.582837	39.189790	-96.582837	39.189790
5	6.06	-96.582836	39.189779	-96.582836	39.189778
6	7.58	-96.582834	39.189767	-96.582835	39.189767
7	8.34	-96.582834	39.189761	-96.582834	39.189761
8	9.11	-96.582833	39.189755	-96.582834	39.189755

Table 17. System generated and calculated coordinates of detached in-canopy sprinklers for sensor bearing of 185 degrees.

	Degrees	Radians
Sensor Bearing:	185	3.23
Measured Bearing:	186	3.25

Pivot Point Longitude:	96.582840
Pivot Point Latitude:	39.189826

		System Generated Coordinates Using Sensor Bearing		**Calculated Coordinates Using Measured Bearing**	
Sprinkler No.	*Radius (m)*	*Longitude*	*Latitude*	*Longitude*	*Latitude*
1	1.55	-96.582841	39.189814	-96.582841	39.189814
2	3.06	-96.582842	39.189803	-96.582843	39.189802
3	3.82	-96.582842	39.189797	-96.582843	39.189796
4	4.61	-96.582843	39.189790	-96.582844	39.189790
5	6.06	-96.582844	39.189779	-96.582845	39.189779
6	7.58	-96.582845	39.189767	-96.582846	39.189767
7	8.34	-96.582845	39.189761	-96.582847	39.189761
8	9.11	-96.582846	39.189755	-96.582847	39.189755

Table 18. System generated and calculated coordinates of detached in-canopy sprinklers for sensor bearing of 189 degrees.

	Degrees	Radians
Sensor Bearing:	189	3.30
Measured Bearing:	188	3.28

Pivot Point Longitude:	-96.582840
Pivot Point Latitude:	39.189826

		System Generated Coordinates Using Sensor Bearing		**Calculated Coordinates Using Measured Bearing**	
Sprinkler No.	*Radius (m)*	*Longitude*	*Latitude*	*Longitude*	*Latitude*
1	1.55	-96.582842	39.189814	-96.582842	39.189814
2	3.06	-96.582844	39.189803	-96.582843	39.189802
3	3.82	-96.582845	39.189797	-96.582844	39.189796
4	4.61	-96.582846	39.189791	-96.582845	39.189790
5	6.06	-96.582848	39.189780	-96.582847	39.189779
6	7.58	-96.582850	39.189768	-96.582848	39.189767
7	8.34	-96.582851	39.189762	-96.582849	39.189761
8	9.11	-96.582852	39.189756	-96.582850	39.189755

Appendix J - Maps of System Generated vs. Calculated Coordinates of Detached In-Canopy Sprinklers

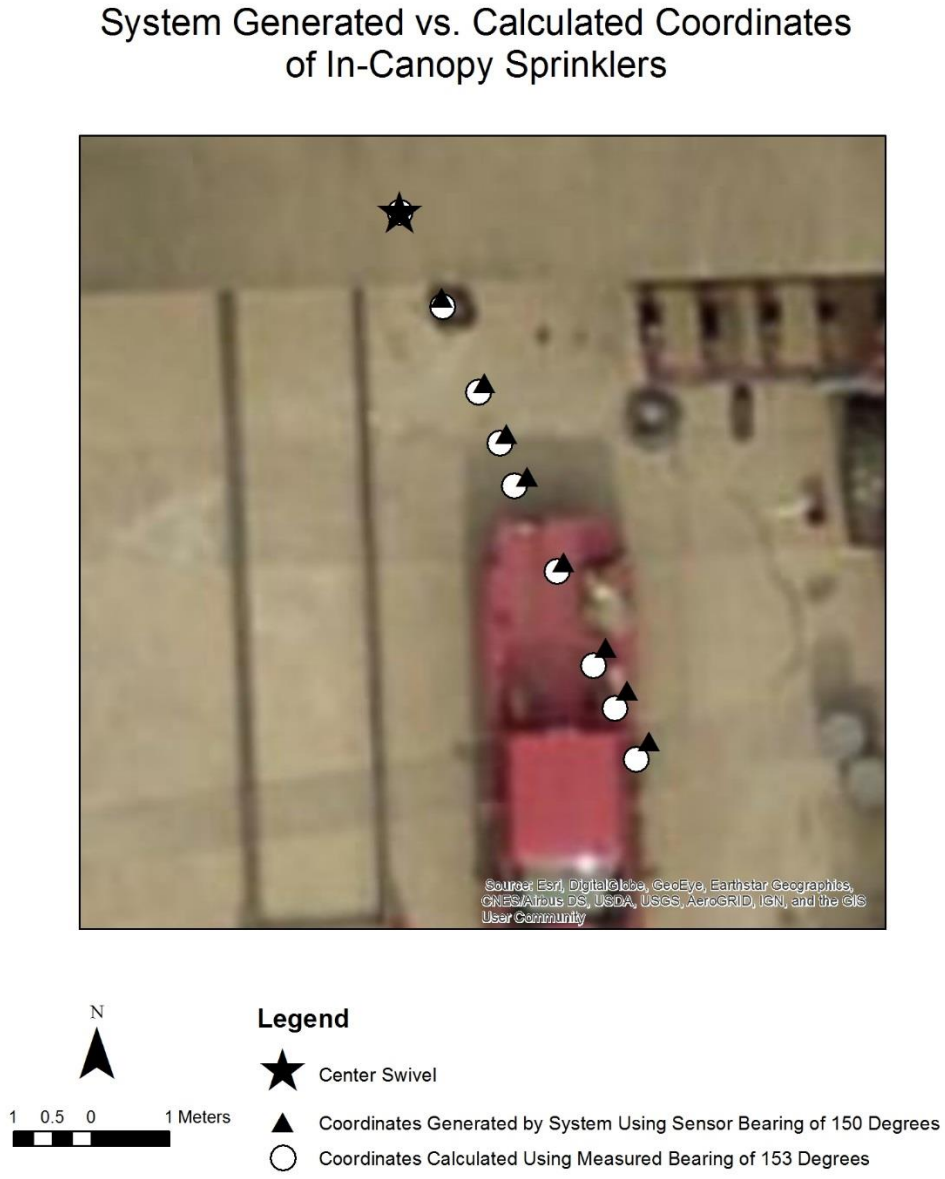

Figure 20. System generated vs. calculated coordinates of detached in-canopy sprinklers for sensor bearing of 150 degrees.

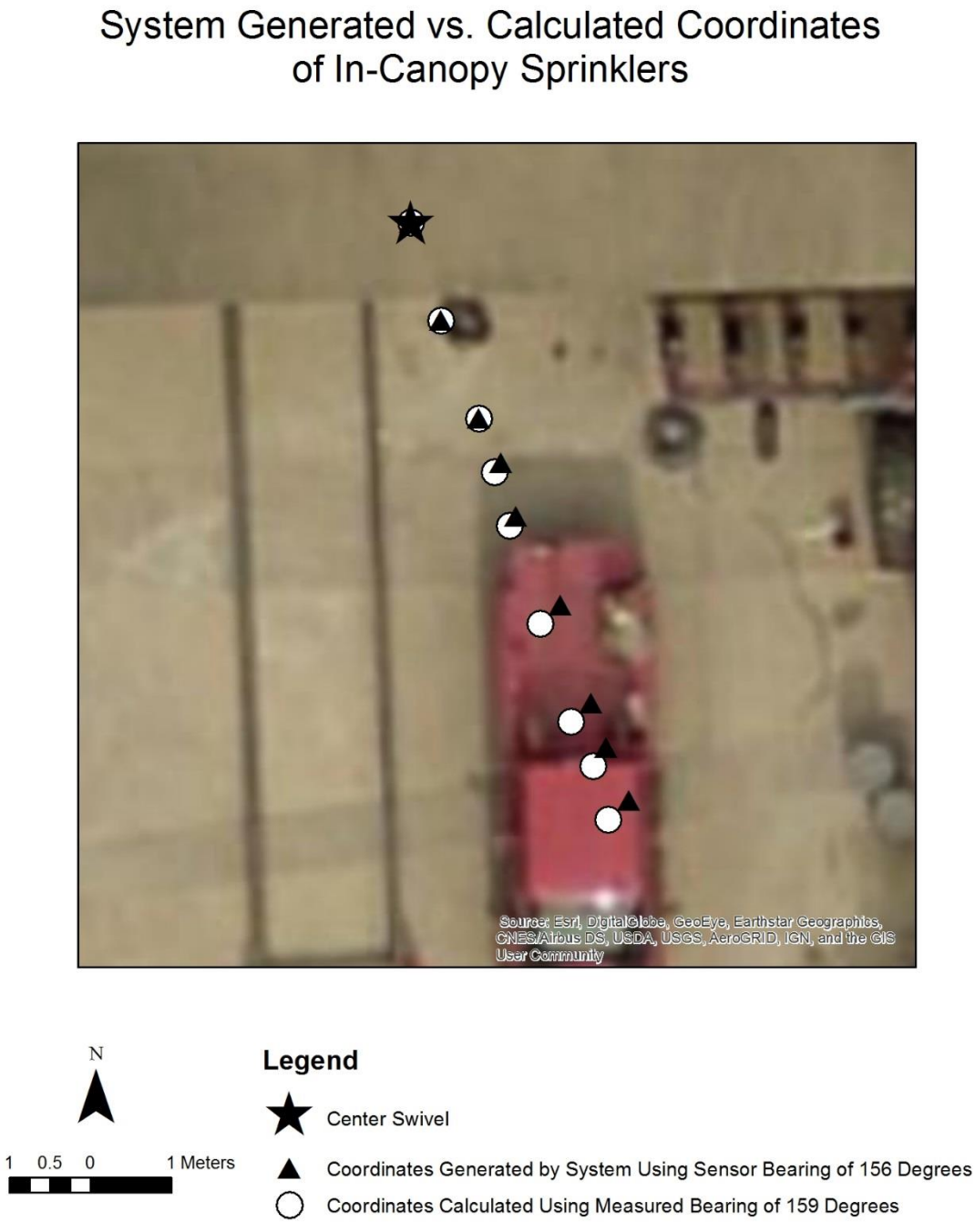

Figure 21. System generated vs. calculated coordinates of detached in-canopy sprinklers for sensor bearing of 156 degrees.

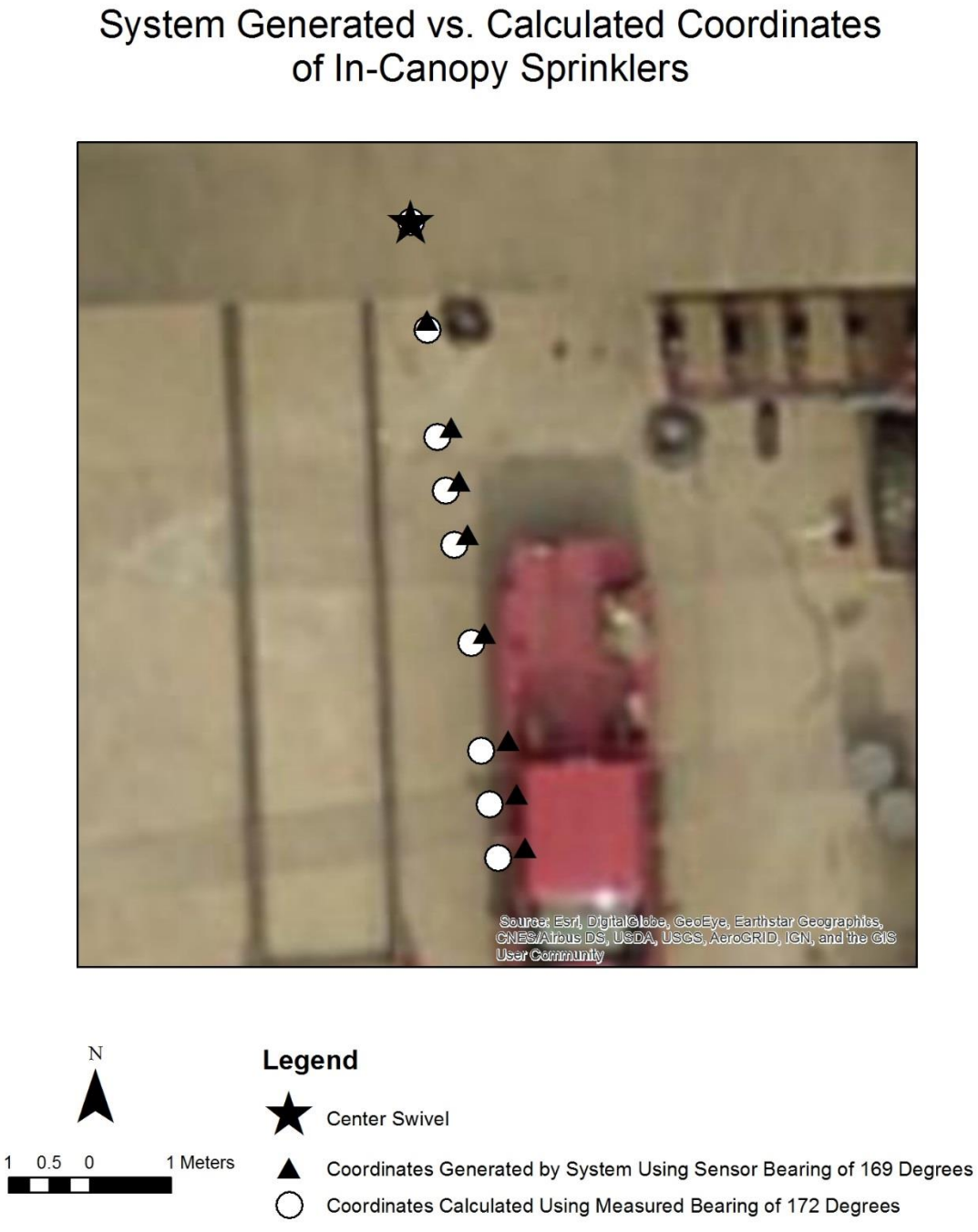

Figure 22. System generated vs. calculated coordinates of detached in-canopy sprinklers for sensor bearing of 169 degrees.

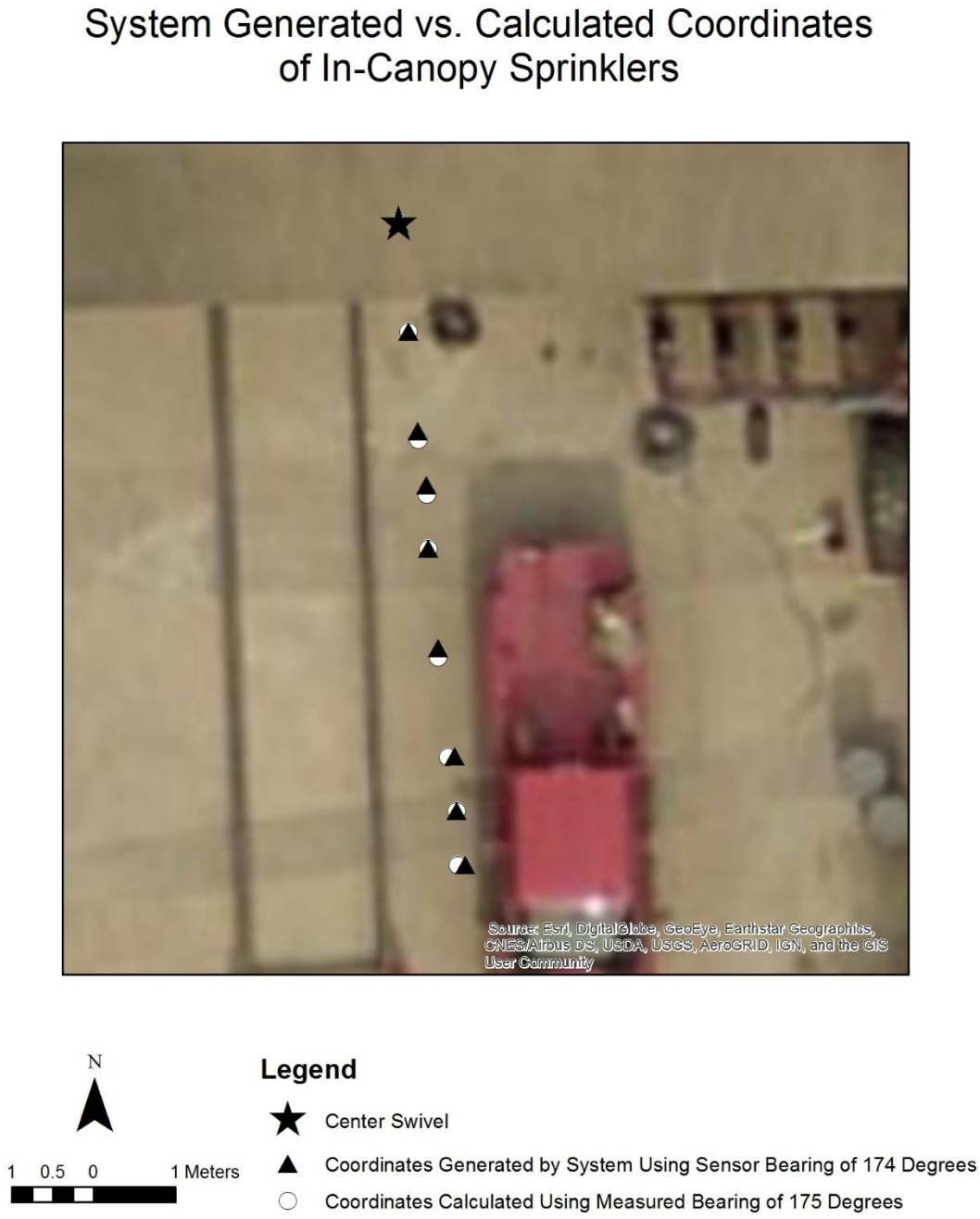

Figure 23. System generated vs. calculated coordinates of detached in-canopy sprinklers for sensor bearing of 174 degrees.

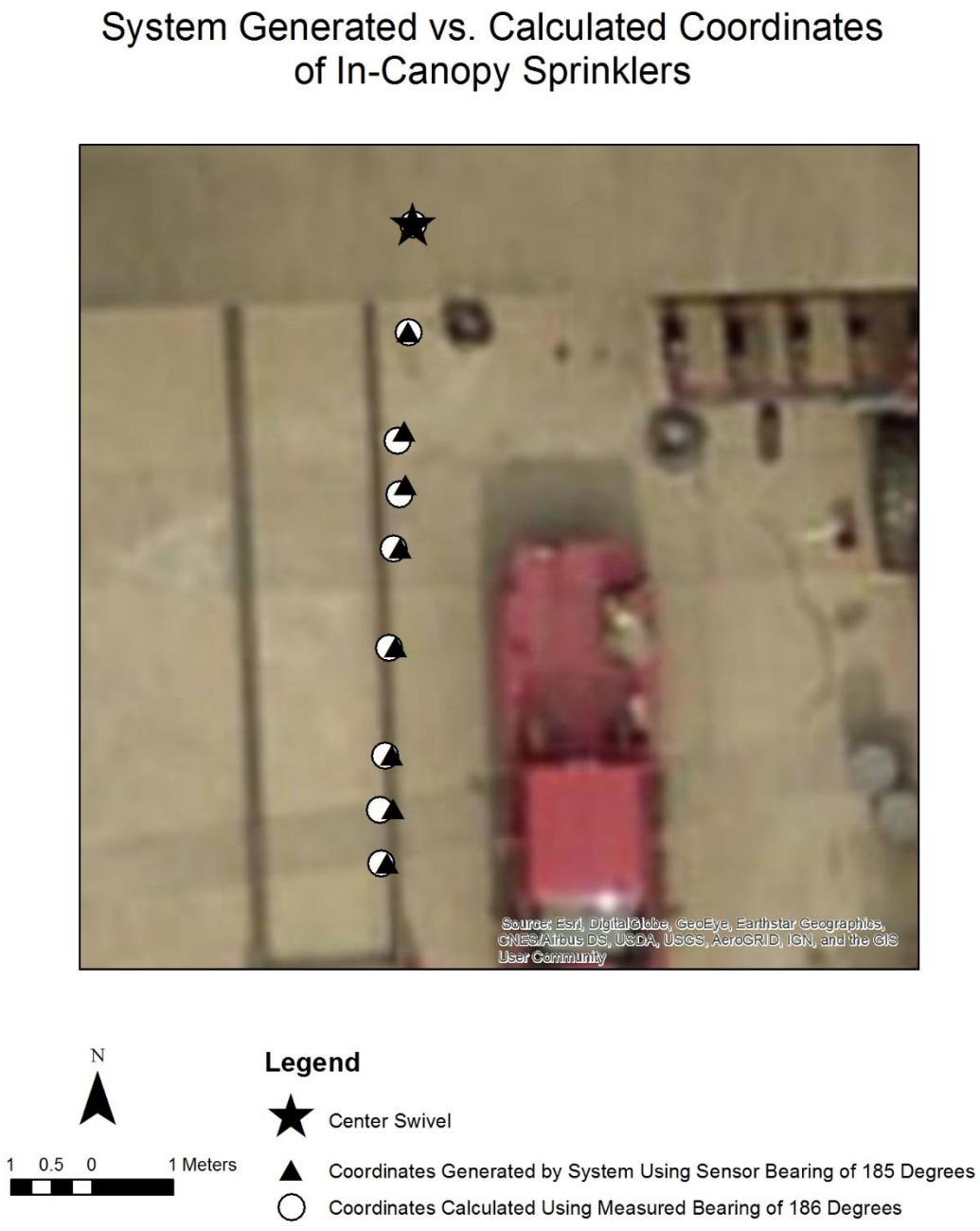

Figure 24. System generated vs. calculated coordinates of detached in-canopy sprinklers for sensor bearing of 185 degrees.

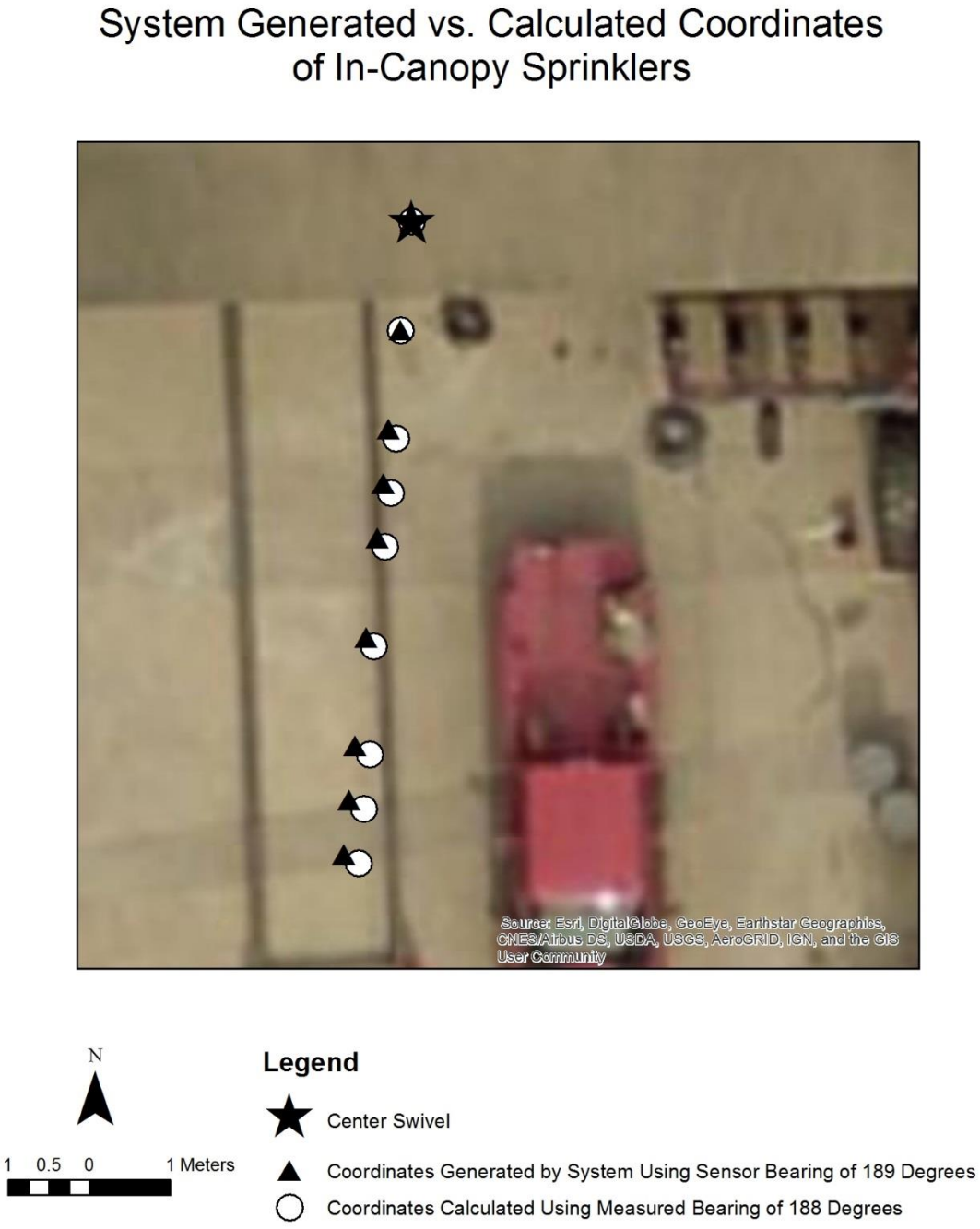

Figure 25. System generated vs. calculated coordinates of detached in-canopy sprinklers for sensor bearing of 189 degrees.