A.I.V.

Autonomous Irrigation Vehicle

Senior Design 2 Final Document

SAMUEL M. RICHIE, PH.D

Sponsored by: Guard Dog Valves

Group Number 4 Members:
Joshua Betetta (EE)
Parameswari Chandrasekar (CpE)
Stuart Munich (CpE)
Roberto Santos (EE)

Table of Contents

Table of Figures

List of Tables

List of Formulas

1.Executive Summary

The Autonomous Irrigation Vehicle is a project that aims to change how modern homes irrigate their property. Originally, the idea of revolutionizing modern home irrigation systems was to restructure the valve layout and control of a typical irrigation system. Most modern homes have an irrigation system that have a valve that controls each sectioned zone on the property. The group has proposed to replace that system with an innovative solution of having one valve control all of the sections. This manner of water management and control would allow the consumer to save on water usage.

Once the group had met up with the sponsor and other groups that had petitioned to work with the sponsor, it had been decided to revolutionize the modern home irrigation system by creating an autonomous irrigation vehicle. The newly proposed autonomous irrigation vehicle was to be outfitted with a water tank, a home base, and a mesh network of moisture sensors. After meeting and discussing with the other groups, the autonomous irrigation vehicle had a few revisions. The main revisions being that the original water tank size was too large which had to be scaled down and then completely removed. When the water tank was scaled down to a smaller size, a tethered hose was proposed to be used in unison with the tank. At the same time, there was to be a solar panel installed on the home base but was later then removed. The final revision of the autonomous irrigation vehicle had the water tank completely removed, solar panels removed, and have a tethered hose from the home base.

As such, the final revision of the autonomous irrigation vehicle will be equipped with a tethered hose, be able to recharge its onboard battery at the home base and be able to communicate with the mesh network of moisture sensors. With the autonomous irrigation vehicle having a tethered hose, the vehicle itself will have an easier time traversing its given environment. The autonomous irrigation vehicle will have a sprinkler head which will be activated by a solenoid that is located at the home base. The activation of the solenoid will be triggered by the autonomous irrigation vehicle itself once it receives data from the mesh network of moisture sensors.

The setup of the mesh network of the moisture sensors is a complicated yet simple method. In essence, the operation of the mesh network behaves as a colony of information. That is to say that whatever information a moisture sensor has, it will relay its data and information to the nearest moisture sensor on the network. From there, the moisture sensor that receives the information will be able to relay the received data as well as transmit its own moisture data. This effect will branch out like a spiderweb of information until all of the moisture sensors on the mesh network have information from all of the other moisture sensors. This set up is crucial since the home base and the autonomous irrigation vehicle will also be communicating with the mesh network of moisture sensors that are already in place.

The home base operation will also be a huge impact on the autonomous irrigation vehicle operations. The home base will be the starting point of the autonomous irrigation vehicle. From here, the autonomous irrigation vehicle will venture and begin taking in data. While the autonomous irrigation vehicle is out and about, the home base will be monitoring the water pressure and as well as waiting for it receive a signal from the autonomous irrigation vehicle. The signal will activate a solenoid at the home base which will then send water to the autonomous irrigation vehicle to begin watering the designated area. While the autonomous irrigation vehicle is at the home base, it will be able to recharge its onboard battery.

The autonomous irrigation vehicle will also be working autonomously as the namesake says. In order to achieve autonomous operations for the autonomous irrigation vehicle, it is programmed with an algorithm and equipped with a few sensors. The sensors provide the autonomous irrigation vehicle with vision intelligence, the ability to analyze and collect data from its immediate surrounding environment. To process the information and data received from the sensors, a machine learning algorithm is incorporated to the autonomous irrigation vehicle's programming. Machine learning algorithms allow the autonomous irrigation vehicle to learn from its previous experiences as well as recognize patterns. With these sensors and algorithms, the autonomous irrigation vehicle will behave and operate autonomously while avoiding any obstacles in its direct path.

The project consists of 3 working groups. The first group is the mechanical engineering group which will handle the home base and water flow aspect of the autonomous irrigation vehicle. The second group is comprised of electrical and computer engineers and are in charge of the autonomous vehicle's autonomous operation. The third group is also comprised of electrical and computer engineers and are in charge of the mesh network for all devices to communicate with each other. The parent sponsor is Guard Dog Valves.

2.Project Narrative Description

While performing marketing analysis of modern home irrigation systems, the group found that the majority of competitive products are using the classical method of transporting water through irrigation piping. The team's original idea was to innovate the yard's valve layout. Most common home irrigation systems have a valve that controls a section of sprinklers; for example, one valve controls the backyard while another valve controls the front yard as seen in Figure 1.

In order to innovate the team proposed a modern approach which would have an individual sprinkler head control as seen in Figure 2. The control of the individual water flow would conserve water usage. After a couple of meetings with the sponsor the team suggested to possibly not use pipes, so in pursuance of revolutionizing the modern home irrigation system, the group proposes to use an autonomous mobile vehicle which carries a scalable quantity of water, as well as a hose attachment which would be used to irrigate the dry portions of the yard. Allocating the water with this kind of precision

and efficiency would save the consumer money by cutting down the costs of water usage through accurate data.

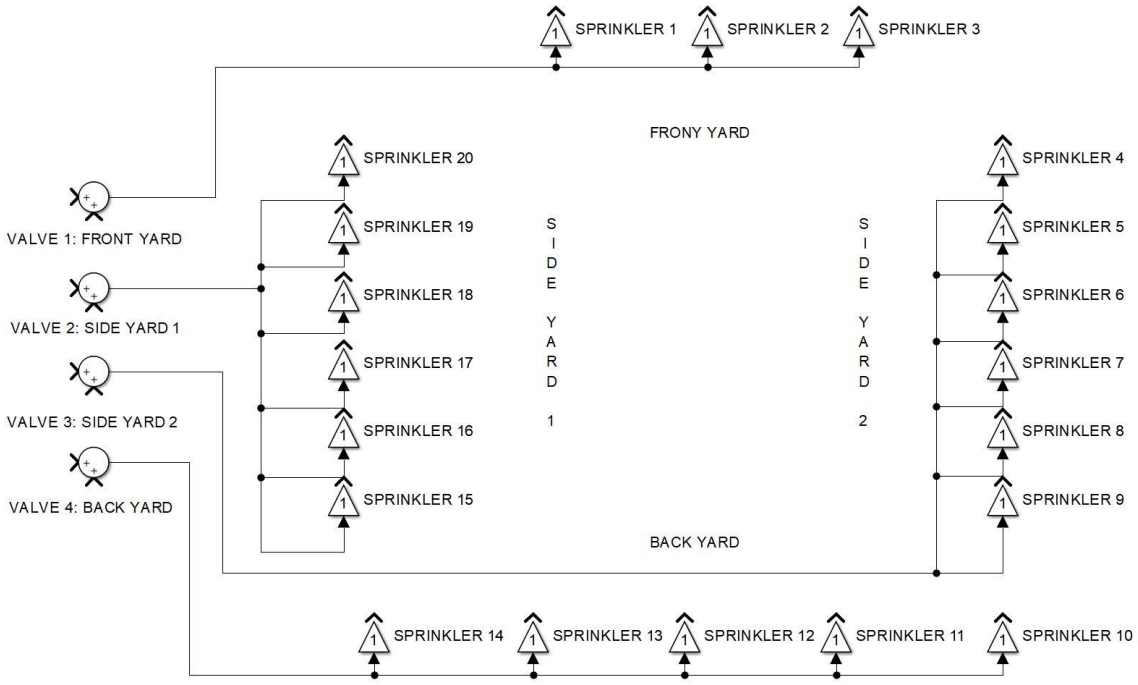


Figure 1: Sectional Valve Layout

Our autonomous vehicle operates using vision intelligence and machine learning algorithms that utilize data received from a Light Detection and Ranging sensor (LIDAR). Visual intelligence enables the vehicle to identify obstacles in the surrounding environment and avoid collisions. Machine learning algorithms allow the vehicle to learn and make predictions based on data stored from previous experiences by recognizing patterns in the visual signals. These algorithms allow the vehicle to adapt and traverse new environments.

The time and location of when and where the vehicle will water is dependent on a mesh network of soil moisture sensors. The mesh network will work with a central hub that is able to communicate with all of the moisture sensors within range to collect and return the collected data. The sensors within range of the central hub will relay the message to other neighboring sensors outside of the central hub's range, and so on to other neighbors, whereby the central hub will eventually obtain all of the information available from all of the sensors on the mesh network. Depending on the type of grass and moisture desired, the vehicle will be dispatched to water the location deprived of moisture. Another feature discussed in the meeting has been watering other plants other than grass in the yard. This will be accomplished by allowing the user to program the amount of water needed by that specific pant using a graphical user interface (GUI). Each plant will

have a specific Radio-frequency identification (RFID) so that the vehicle will know where the plants are and how much to water them based on the user input.

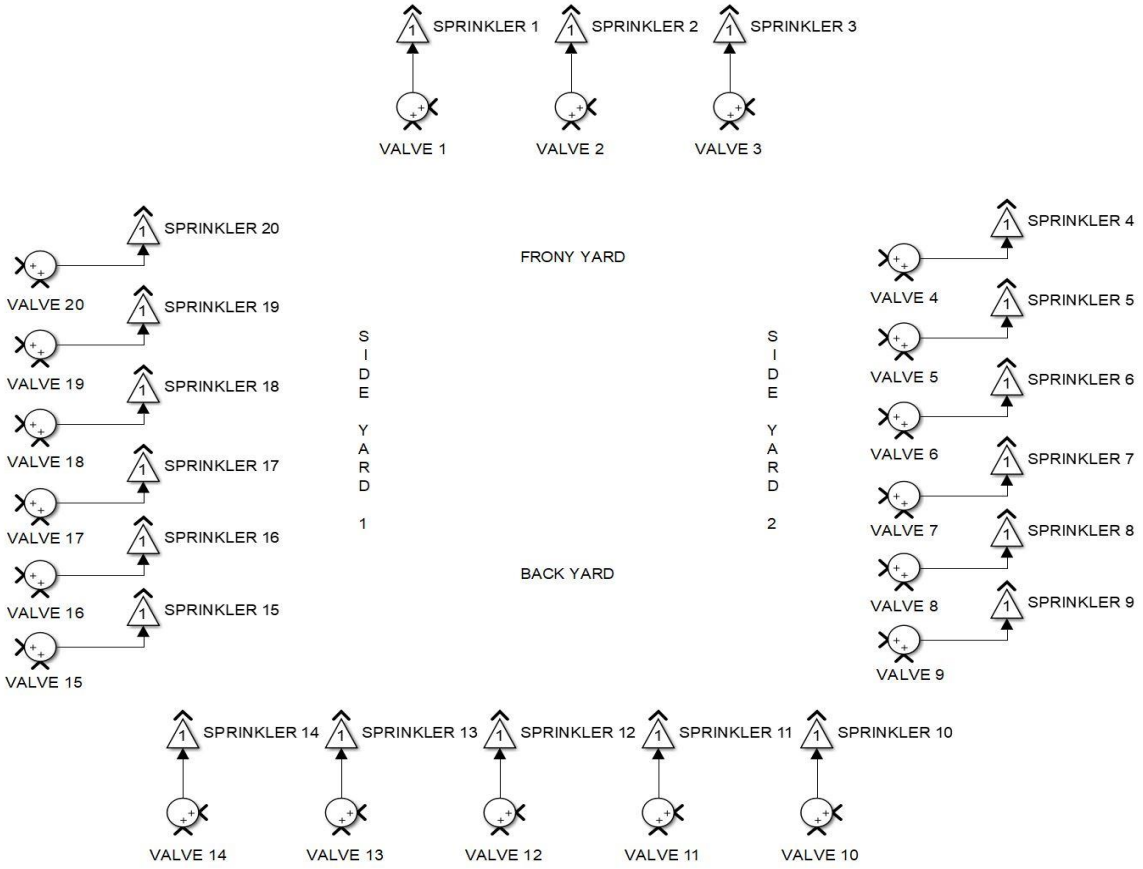


Figure 2: Individual Valve Layout

When the vehicle is not in use, it will be stationed at a home base until it is called upon to dispense water based on data from the sensors to water a designated area. The home base will feature a water reservoir and a charging station. The vehicle will be able to refill its watering tank from the water reservoir. The water reservoir will also be able to collect rainwater for reuse as well as have a garden hose attachment in case there is not a sufficient amount of water. For proof of concept, the team has decided to make a hybrid system which includes a small tank as mentioned above, but most of the water will come from a hose. The hose system will automatically retract into a spool providing an adequate amount of slack for the irrigation vehicle to traverse a yard. While at home base, the autonomous vehicle will be able to recharge its battery thereby powering its DC motors, the PCB it houses, and the sensors. Home base will also be powered by a solar panel to ensure a clean and renewable source of energy. For proof of concept, the team has decided to add a hose to the vehicle, an alternating current (AC) power cord will also be used to provide power most of the time. Through these proposed means, irrigation methods will be revolutionized. Pictured below in Figure 3 depicts the overall system diagram.

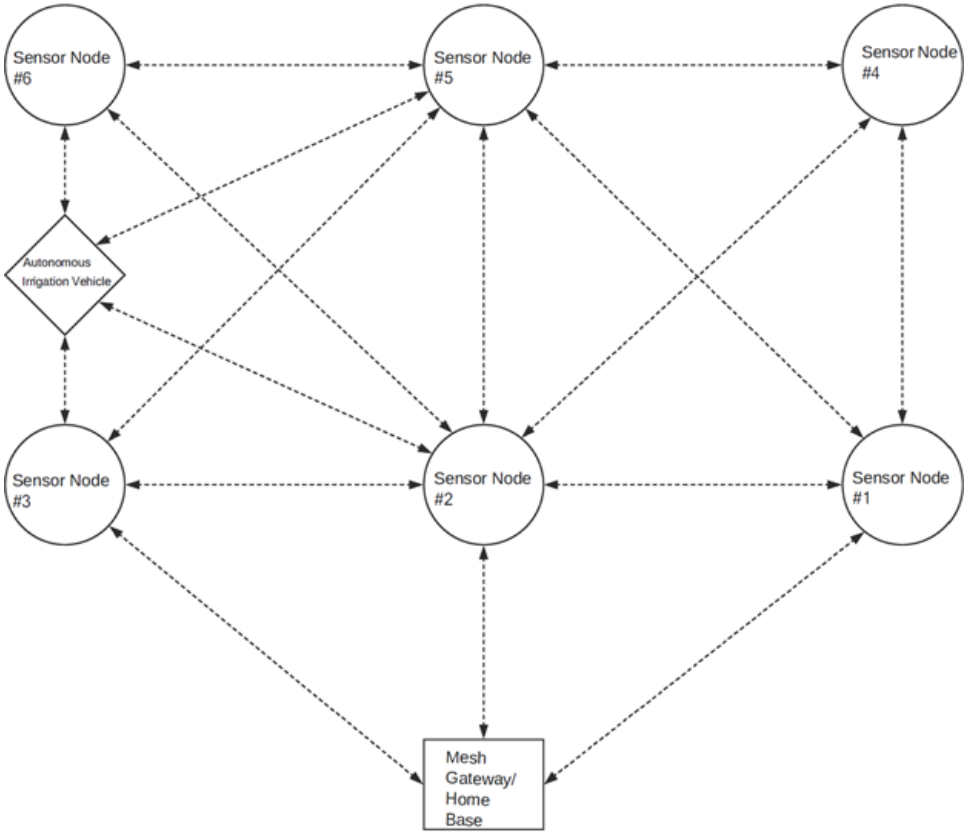


Figure 3: Overall System Schematic

The mesh network portion of the project will be researched and designed by another Electrical/Computer engineering team which is part of the bigger sponsor. The home base, muscle and skeleton of the vehicle, and the water flow elements will be researched and designed by the Mechanical team which is also part of the bigger Guard Dog Valves sponsored team.

2.1 Team Motivation

The team's motivation to choose this specific project comes from one of the first assignments completed for our Senior Design 1 class. The assignment was given before we had picked teams and had each student in the class come up with an idea for a project. Two of the members of our group came up with ideas that included renewable energy sources. The first idea was a portable electronic charger that could function using solar, wind, and hydro energy. The second was a farming robot that would autonomously plant the seeds and grow the plants. Once we began forming teams, we decided to make the farming robot, but power the robot using the three-renewable energy sources form the charger. As professor Samuel M. Richie presented the sponsored projects, we became interested in the Guard Dog Valves irrigation project. Since this project was about being

green and saving water the team was excited that something similar to our original idea was proposed, and we decided to take on the challenge.

Another factor motivating the team is that we all live and go to school and work in Florida. Water conservation is a big deal in the state of Florida because of the Floridan aquifer. The is the lifeline for all the great natural springs in Florida that tourists and locals alike enjoy all year long especially in the summer months. The Floridan aquifer is also used by all the citizens of the state as a water source. As water in the springs is depleted they turn into sinkholes. Our project could help preserve the springs.

2.2 Sponsor Inspired Motivation

The Guard Dog Valves sponsor spoke to us in the meeting that one of the reasons for wanting to produce this product was an article written by the University of Florida. The article states: "*Researcher Michael Dukes found that for three of four soil-moisture sensors tested, water savings ranged from 69 percent to 92 percent, compared to grass watered without the help of sensors*." (Anderson). The article also mentions "*Taking the human component out of the watering process certainly seems to help reduce overwatering, said Kathy Scott, section manager for conservation projects with the Southwest Florida Water Management District, which sets water policy for some 4.5 million residents*." (Anderson). In addition, it mentions that over watering the lawn is actually a bad thing because the grass does not grow long roots because it thinks that water will always be available. The short roots make the lawn prone to diseases and pests. In summary, overwatering not only costs more money but makes the grass weak. The product being developed hopes to use moisture sensors in conjunction with the mesh communication network, and the irrigation vehicle to take the human factor out of watering residential lawns and saving customers money and making their lawns healthy and strong.

Another motivation article shared by Guard Dog Valves is published by the EPA. The article is about how during the summer months the hotter weather causes more water to be needed, as much as two to four times the amount. Since most home irrigation systems are not smart and must be manually changed, a lot of homeowners forget to adjust them back after the summer is over. Not adjusting the irrigation system causes over watering during the colder months that do not require as much water. Even though summertime requires more water, a lot of overwatering occurs according to the article "*Depending on the region, homeowners use between 30 and 70 percent of their water outdoors." (When it's Hot)" and Experts estimate that "50 percent of the water we use outdoors goes to waste from evaporation, wind, or runoff due to overwatering." (When it's Hot).* The article also gives customers tips for saving money:

1. The first tip is knowing how much water the lawn needs and that it's better to water during early morning or late afternoon when the sun is not as hot.

2. The second tip is upgrading to a smarter home irrigation system with local weather data.

3. Another tip is for residents to be aware of the different zones in their irrigation systems and adjust the zones depending on the type of plants and of much time they spend in the sun or in the shade.

The product we designed allows the consumer not to have to remember to change the setting after summers are over. The autonomous irrigation vehicle automatically knows how much water each zone of the lawn needs, so the consumer will not have to worry about that or how much time the zones are in the sun or the shade. The autonomous irrigation vehicle saves money by water during the hours that are most beneficial for the lawn. This one-time purchase covers all the recommendations made in the article allowing the consumer to have the ease of mind that they are saving money and the yard is properly hydrated. All of these factors make the autonomous irrigation vehicle a great product for Guard Dog Valve to market.

2.3 Motivation Goals

The Goals for the project are for the autonomous irrigation vehicle to provide the consumer with an irrigation system that does not over water the grass causing shallow roots that make the lawn more susceptible to pests and diseases and cost more money, not just in the water bill but also the extra money spent on curing the diseases and getting rid of the pests. While properly irrigating the lawn will not get rid of diseases and pests it will allow the grass to grow long deep roots that making it strong and therefore less susceptible to the pests and diseases. These goals will all be achieved by removing the human factor from the whole process.

The lawn caretaker will no longer have to remember to change the automatic timers as the seasons change. They will also not need to be aware of how much water the lawn needs or if it has rained recently. How much sun or shade a certain zone gets during the weather changes will not need to be something the consumer has to think about anymore. These factors are great selling point for the autonomous irrigation vehicle.

The first time the consumer sets up the autonomous irrigation vehicle they will have to input the days and times the local government allows them to water their yard so that autonomous irrigation vehicle does not cause them to pay a fine for watering on restricted times or days. As well as install the proper amount of mesh network moisture sensors. The amount of mesh network moisture sensors will depend on the size of the lawn. Once that is complete, they can be worry free about the yard irrigation. With a single investment, the consumer will have more free time to do more important things while

knowing they will have a properly irrigated lawn that is healthy, strong, and grows deep roots all while saving money.

2.4 Objectives

The objectives of the autonomous irrigation vehicle are simple yet at the same time, are of great significance. As a group, we have discussed about what the autonomous irrigation vehicle's main objectives should be in order to satisfy customer needs as well as market needs. With that being said, determining the objectives for the autonomous irrigation vehicle was no easy task as the vehicle itself has gone through a few revisions since its conception. As the autonomous irrigation vehicle evolved, the objectives evolved alongside the vehicles technological development. Along with the evolution of the autonomous irrigation vehicle, there were considerations as to what would make the project itself feasible and also have objectives that pertained to the vehicle's process of operations. As the autonomous irrigation vehicle evolved, some of the technologies and objectives have been changed or removed but these decisions were made in order for the vehicle to succeed given its surroundings.

The Main Objectives:

- Effectively irrigate a 30x20 ft plot of grass
- Effortlessly avoid obstacles
- Traverse terrain with minimum error
- Reduce water consumption
- Effective communication with mesh network of sensors
- Sense battery capacity
- Control rate of water flow

As mentioned before, the list of main objectives above is what remains after technological growth of the autonomous irrigation vehicle. The 30 by 20 feet plot of land is the average size of a given backyard that is assigned from a typical neighborhood in which homes are built with relatively the same home design and layout. As such, the autonomous irrigation vehicle function will be dispensing water in the most uniform fashion to provide the backyard with a sufficient amount of water. In order for the autonomous irrigation vehicle to dispense the water in a uniform fashion, the vehicle itself is able to avoid and travel around whatever obstacles may be in its way in any given backyard. As the autonomous irrigation vehicle itself is a machine, it is prone to making errors while it travels. As this is an expected outcome, the group will do its best to mitigate errors whilst the autonomous irrigation vehicle is traveling through machine learning algorithms. As the autonomous irrigation vehicle traverses the plot of land more and more, it will adapt and learn its surroundings thus reducing errors while traveling as well avoiding obstacles. While the autonomous irrigation vehicle is traveling and dispensing water to sections of the backyard that have signaled that the area's soil moisture is low, the vehicle will be effectively delivering water to aforementioned designated areas. Water consumption is

an elephantine problem most consumers encounter whilst irrigating and maintaining their lawn so the autonomous irrigation vehicle aims to cut down water consumption by only delivering the sufficient amount of water that the designated area requires. To accomplish this objective, the autonomous irrigation vehicle will have a water flow sensor that will be able to determine how much water has been dispensed and then proceed to shut off the water valve in order to minimize water consumption. For the autonomous irrigation vehicle to properly dispense the sufficient amount of water that is needed by the designated area, it must be able to effectively communicate with the mesh network of sensor. The mesh network of sensors will be providing the autonomous irrigation vehicle with real time updates and soil moisture data which the vehicle will use to determine whether or not a designated area will require watering. Whilst the autonomous irrigation vehicle is on a watering venture, there will be times where the vehicle's on board battery will start to have low reserves of energy. The autonomous irrigation vehicle will need to determine whether or not to continue to the next designated watering area or to return to the home base in order for the vehicle to recharge its battery.

As previously mentioned, the autonomous irrigation vehicle has been revisioned a couple of times before deciding on the final product's direction, architecture, and functions. With the first revision, the initial project idea and suggestion called to make a mesh network for all of the sensors in the ground to be able to communicate with each other and provide other sensors in the designated region, the data from each individual sensor. Each sensor was to collect a variety of data that would be uploaded into the mesh network and sent to the other sensors on the network to communicate with the user or whatever device that was also connected to the mesh network. The initial project was dropped and then became an autonomous irrigation vehicle that carried a water tank in order to dispense water in designated areas based on the data received from the sensors in the mesh network. The autonomous irrigation vehicle also had a proposed solar panel installed on the vehicle itself to charge its onboard battery supply. The autonomous irrigation vehicle would be connected to the mesh network as well to be able to analyze and determine where to water and how much water was needed in that designated area. Due the mountainous encumbrance of the water tank and the added weight from the water, the idea of the water tank on the vehicle was reduced from a 300 gallon tank to a mere 3 gallon tank. Also, the proposed solar panel on the autonomous irrigation vehicle was to be installed on the home base where the vehicle would be able to charge and refill its water tank. In the autonomous irrigation vehicle's last revision, the water tank had been completely scrapped to lower power requirements and allow the consumer a user friendly product. As such, the autonomous irrigation vehicle's previous objectives were practically the same but with additional but now unemployable objectives.

The Previous Objectives

- Effectively irrigate a 30x20 ft plot of grass
- Effortlessly avoid obstacles
- On-the-go battery charging with Solar Panel
- Traverse terrain with minimum error
- Reduce water consumption
- Carry sufficient quantity of water

- Effective communication with mesh network of sensors
- Control rate of water flow

As previously mentioned, the autonomous irrigation vehicle was to be installed with a solar panel to charge the vehicle's onboard battery as it traversed its given terrain. Along with the once proposed solar panel, a water tank was also envisioned to be installed on the autonomous irrigation vehicle in order for the vehicle to irrigate any designated area that required watering. A solar panel was also proposed to be installed but was then redacted to allow more power to be used by the autonomous irrigation vehicle. After careful consideration, the autonomous irrigation vehicle is now in its final stage of research and development.

2.5 Requirement Specifications

In regard to the specifications of the given projects, there are a few sensors that will be utilized by the autonomous irrigation vehicle in order to traverse whatever terrain the vehicle is in as well as determining proper amount of water to be dispensed. Aside from sensors, the vehicle has a printed circuit board and microcontroller working in unison in order to communicate with the vehicle's sensors as well as the moisture sensors installed in the ground. The vehicle's battery life has been designed to output a sufficient amount of power that the vehicle requires as it traverses the terrain with a varying amount of water in the water tank or the varying rate of water flow while irrigating designated areas and plants. Power management is a critical asset for the autonomous irrigation vehicle to achieve a successful irrigation pattern and effective object detection.
Sensors such as LIDAR, power management, and water level will allow the vehicle to safely traverse its surrounding terrain, monitor battery charge and the water tank's level in order to determine whether or not to return to the home base to recharge and refill its battery and water tank. The LIDAR sensor has an approximate range from 2cm up to 6m with a 180 degree view to detect and avoid obstacles in its immediate frontal surroundings. A battery management sensor will be installed in order to monitor the autonomous irrigation vehicles level of battery charge. When the level of charge reaches a certain point, a signal will be transmitted to the vehicle indicating the vehicle to return to its home base. In order for the vehicle to determine where to dispense water, it has a wireless receiver antenna and transmitter to communicate with the measurement sensors that are on the mesh network. As the vehicle will be dispensing water to designated areas, its water level will drop so a water level sensor will also be implemented in the vehicles design. As with the battery management sensor, a signal is sent at a certain threshold of water indicates that the vehicle needs to return to the home base to refill its water tank.

In order for the vehicle to make the decisions of where to dispense water and when to return to home base and for that end, a printed circuit board and microcontroller are installed on the vehicle. A printed circuit board and microcontroller are designed in such

a manner that they are small enough for both of the components to fit comfortably on the autonomous irrigation vehicle. The printed circuit board functions as an intermediary between the sensor data and data processing of the microcontroller. Signals that are generated by the on board sensors are communicated with the microcontroller in order to determine where water will be dispense, if there are any obstacles in the vehicle's path, and whether or not the vehicle needs to return to home base to refill its water tank or recharge its battery. The microcontroller handles all of the sensor data processing as well as movement and obstacle detection per its installed motors and LIDAR sensor, respectively. With the autonomous irrigation vehicle being capable of movement, the vehicle operates at a certain velocity so that the vehicle can safely traverse the terrain that it is in so as to minimize jerking motions.

In terms of power demand, the autonomous irrigation vehicle requires a substantial amount of power in order to traverse the terrain with the water tank. Depending on how much water the tank will be carrying, will dictate how much power will be needed in order for the vehicle to travel with the given weight. As such, a battery unit with a sufficient amount of voltage and amperage will be needed. Furthermore, as the vehicle itself operates out in the open, the battery will either need to be housed to prevent damage from the elements, have a specific amount of water resistance, or have a high resilience to outdoor activity. The power generated by the battery is then distributed accordingly in regard to the demands of the sensors, motors, and water dispensing mechanism.

In regard to dispensing water at the designated watering location, the autonomous irrigation vehicle has an additional method of transporting water. The vehicle will have a water tank attached as well as a tethered hose. The purpose of the hose is to minimize power consumption of the vehicle's on board power supply as the water tank can add additional weight which requires a higher power output. With the hose attached, the water tank can be scaled down in size to minimize the weight, size, and power consumption of the vehicle itself. With the help of the sensors and microcontroller, the vehicle will be able to detect if the hose has been entangled with an obstacle. If the case arises of an entangled hose, the vehicle will be able to self disconnect the hose and continue to water with its own water supply. Once the hose has been detached from the vehicle itself, it will begin to retract itself to the home base or where ever the base of the hose is located.

The original project was a sensor based smart technology which focused on controlling the flow of water to sprinkler heads. Each of the sprinkler head would have had a sensor attached to it which would take in readings for soil moisture, ambient temperature, and online weather data for rain. When the data is read and processed, a microcontroller would have processed the data and would have determined whether or not the designated sprinkler head would dispense water to the designated area. In order to power the sensors on each of the sprinkler heads, a small but efficient solar panel would have been implemented to have provided power to the unit. With each of the sprinkler head sensors collecting data from the sensors, a machine learning algorithm could have been implemented to autonomously optimize watering schedules. In order to for the system to optimize the watering schedules, each of the sensors would have to been connected or linked to each other in to what is called a mesh network. In the mesh network, each of

the sensors would be communicating and relaying data with each other with would provide the algorithm with enough data to begin to learn how to optimize watering schedule. Along with the proposed mesh network of sensors, mobile device application would have been designed to accompany the propose sensor technology. With the application on the mobile device, it would have been able to give the end user the ability to alter and directly be able to water any designated area with the sprinkler that was in the area. By giving the user, or customer, the ability to directly control the watering schedule, the system could also learn and develop routines based on the user's inputs for the watering schedule. Along with providing the customer with the opportunity to control, designate, and set watering schedules, the mobile device application would have also provided the customer with valuable data such as moisture levels of the soil of a specific region, errors that may have occurred, and provide the user with current an up to date watering routines. Below is Table 1 which lays out the targeted specification goals.

	Measures
Vehicle Size	21.10" x 24.13" x 5.55" (L x W x H)
Obstacle Detection Distance	2 cm to 6 m
Time to water designated area	≤1 hour per zone
Targeted rate of Water Flow	24 Gallons Per Minute
Average Lawn Size	$10890 ft^2$ (.25 acre)
Battery Life	≤1 hour
printed circuit board size	6" x 5" (L x W)

Table 1: Project Specifications

As previously mentioned, the list has a desired goal of specifications that the team would prefer to achieve in order to implement a successful working autonomous irrigation

vehicle. The size of the autonomous irrigation vehicle has been decided to be an efficient size to allow safe and uninhibited travel through a grassy terrain. Obstacle detection is an enormous hurdle that will be dealt with accordingly and successfully. Water deployment will also play a crucial role in an successful performance for the autonomous irrigation vehicle.

2.6 House of Quality

The House of quality shown in Figure 4 represents inner relationship between the marketing requirement and engineering requirements as well as the correlation matrix shows how the engineering requirement impact each other. Each requirement is associated with positive or negative polarity.

The marking requirements are focused on what the customer need and expects. Cost is one of the main constraint in marking requirement. The cost constraint represents how much lower the customers want the price to be and it also has negative polarity which shows that the lower the cost, higher the demand for the product. Likewise, Power consumption is a measure of how much power the autonomous irrigation vehicle would consume to complete the target tasks.

User friendliness is another crucial part of marketing requirement because it defines how easily the autonomous irrigation vehicle can be operated by the end user. As well as the function of the vehicle should to easy to maintain. When customers buy new product (vehicle) first they check how reliable the product is. The reliability represents how well the autonomous irrigation vehicle will perform for this specific task. Increasing the user friendliness and reliability of the product will make the product more desirable among buyers, so both requirements represented in positive polarity.

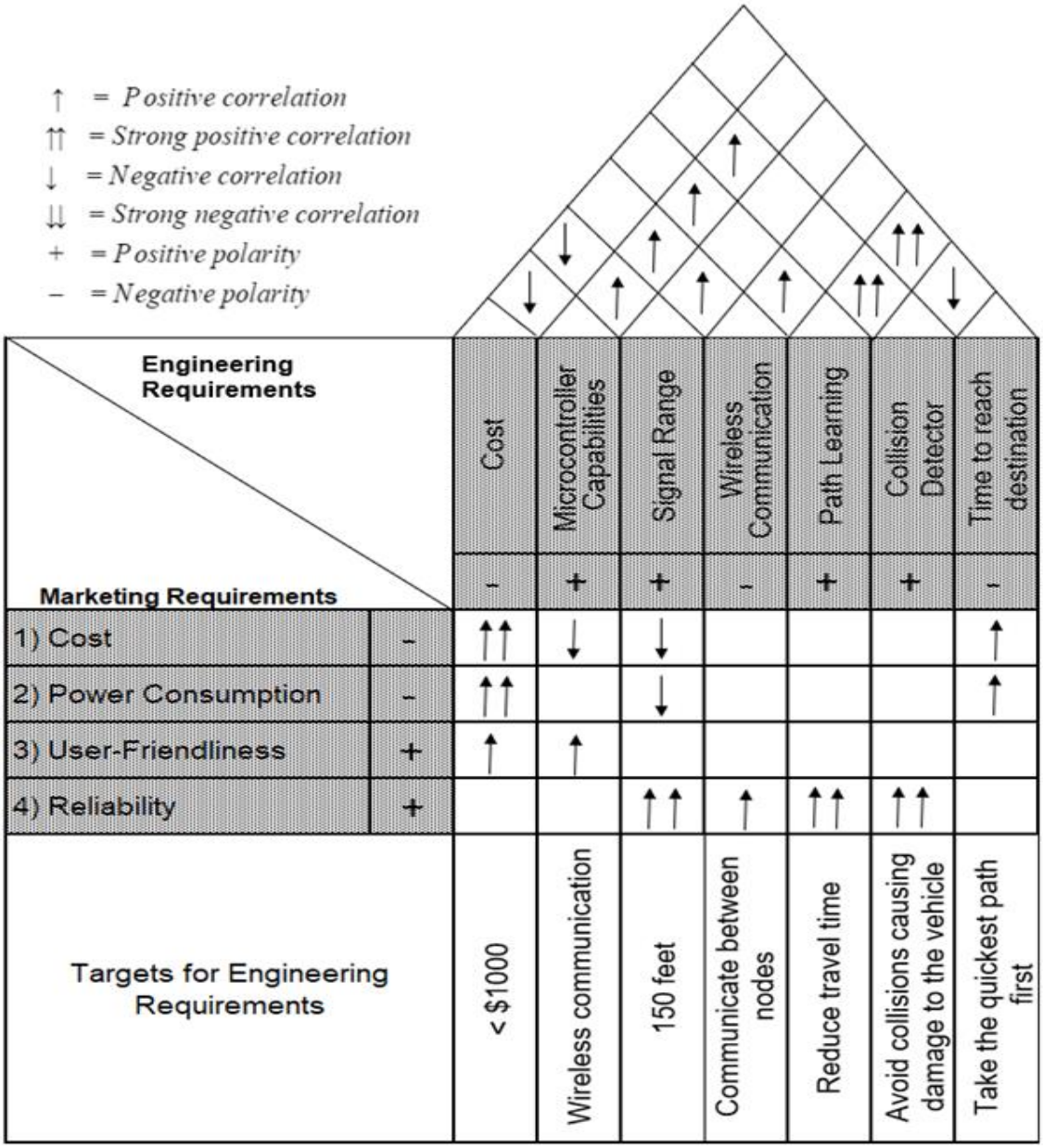


Figure 4: House of Quality

The engineering requirements represents how the engineers could design the product which will meet customer needs. As we talked about in marking requirement the lower the cost is better the demand. Thus, the cost is representing in negative polarity. Since Microcontroller is the brain of the autonomous irrigation vehicle, microcontroller enable engineers to interface mesh network and autonomous irrigation vehicle. Having a high capabilities microcontroller will allow engineers to reach the target more efficiently. Evidently the signal range is another significant part of engineering requirement. Each sensor node must communicate with microcontroller. Thus, the microcontroller capabilities and signal range are represented in positive polarity. Path learning will help

the vehicle to reach the target node in the quickest way as well as it helps the autonomous irrigation vehicle to avoid the collision. Evidently Increase the collision detection and path learning will make the vehicle more desirable.

2.6.1 Signal Range

As the autonomous irrigation vehicle communicates with the mesh network of sensors that re placed in the given terrain of a customer, signal ranging is a determining factor. The autonomous irrigation vehicle has the ability to effectively communicate with the sensors in the mesh network and in order to do so, it has a suitable and sufficient amount of signal range. The signal range of the autonomous irrigation vehicle is determined by a few factors. The first factor is based on the power draw of the wireless communication sensor that will be implemented in order to allow proper communication between the sensors. The second determining factor is the distance between the location of the sensor node and the present location of the autonomous irrigation vehicle.

IEEE 802.15 operates as a radio wave which is an electromagnetic field. Through an electromagnetic field, information can be sent and received by 2 or more components. The electromagnetic field range is given by a combination of Maxwell's equations taking the form of a electromagnetic wave in a volume of free space. These fields are impacted not only by time but also by electric potential as well as the distance between 2 points. When both are combined, they generate an electromagnetic field in which components can communicate with each other.

Effective communication will play a critical role in the successful execution of the autonomous irrigation vehicle delivery of water to certain designated areas. Accomplishing the communication is determined by the effectiveness and strength of the autonomous irrigation vehicle's onboard wireless communication device. As such, there are multiple wireless signals that can be used. For instance, based on the Institute of Electrical and Electronics Engineers (IEEE for short) association standards, the 802.15 radio frequency, a wireless personal area network, operates on the ISM (Industrial, Scientific, and Medical) band and is commonly used for 2 pieces of hardware to communicate with each other. The IEEE 802.15 comes in many different variations but due to the autonomous irrigation vehicle's needs for wireless communication, only four different types of wireless personal area network will be considered.

IEEE 802.15.1 is a task group that is based on Bluetooth technology. That is to say that it defines physical layer and media access control specifications within a personal operating space that allows wireless connection of fixed, portable, and moving devices. IEEE 802.15.4 is a low data rate WPAN which promotes a very long battery life as well as providing a very low level of complexity. IEEE 802.15.4a is an amendment for the IEEE 802.15.4 which specifies additional physical layers thus providing higher precision, throughput, a longer range, lower power consumption as well as cost. IEEE 802.15.4b is an improvement to the IEEE 802.15.4a that reduced complexity and increased flexibility

among security key usage. Through careful research and consideration, an acceptable device has been chosen.

2.6.2 Wireless Communication

An effective method of communication must be established in order for the autonomous irrigation vehicle to be able to dispense water accordingly and efficiently to certain designated areas. The mesh network of sensors that are placed within the autonomous irrigation vehicle's given terrain, communicates with the vehicle as it traverses the given terrain. The autonomous irrigation vehicle will be connected to the mesh network of sensors in order to determine which designated areas requiring water and proceed to the location relayed by the sensors. In order to accomplish an effective and efficient method of communication between the autonomous irrigation vehicle and the mesh network of sensors, three types of wireless personal area networks have been considered.

With IEEE 802.15, a wireless personal area network is established which allows the interconnect ability of multiple devices within a certain designated area. IEEE 802.15 operates within the 2.4 gigahertz frequency band. Another perk of the IEEE 802.15 standard is that it allows any two or more wireless personal area network equipped devices to be able to communicate with each other, a mesh network.

2.6.3 Path Learning

Path learning is a crucial component of engineering requirement. The autonomous irrigation system is able to reach the destination midpoint in specific time. The impact of path learning in marketing requirement is reliable autonomous irrigation system takes some time to learn the path at the beginning while it travels to the destination midpoints. When autonomous irrigation vehicle travel to the destination midpoints, it uses the computer vision obstacle detection system to detect any obstacles such as any hard objects, big stone, small tree, lawn status and etc. Once it detects those obstacles, it updates the map. And whenever the autonomous irrigation vehicle travel to the midpoints it keeps update the map constantly. Having this updated map data wound help the autonomous vehicle to reach the target points quickly. Besides, the goal of using path learning in this project is to reach the destination midpoint or midpoints in the quickest possible time. Having this updated map data and effective path algorithm would help to reach the goal this component. In addition, path learning algorithm which is the nearest neighbor algorithm (NNA) generates the shortest path to reach the target node and make sure that the vehicle water the lawn effectively. Also, path learning has a strong correlation with reliability. The better the autonomous irrigation vehicle learns the path the higher the reliability of the product. Evidently, increasing the path learning makes the vehicle more desirable.

2.6.4 Collision Detector

Collision detector is another significant component of autonomous irrigation vehicle. During the learning phase the autonomous irrigation system takes some time to learn the path while it travel to the destination midpoints as well as it uses the computer vision obstacle detection system to detect any obstacles such as hard objects, big stone, small tree, lawn status and flipping over pond. Once the computer vision obstacle detection system detects those obstacles, it updates the map. And whenever the autonomous irrigation vehicle travel to the midpoints it keeps updating the map constantly.

Having this updated map data wound able to help the autonomous vehicle to reach the target points without being stuck by any obstacles. In addition, all of this data is necessary for the computer vision obstacle detection system to safely travel around the lawn to deliver the needed water. Since we want the collision detector to have more capability to detect the obstacles, the collision detector is represented in positive polarity in the house of quality. Also, it has strong positive correlation with one of the marketing requirement of the reliability. In marketing perspective, the goal of the collision detector is to increase the reliability of the autonomous irrigation vehicle. Evidently, Increase the collision detection will make the vehicle more desirable.

2.6.5 Time to Reach Destination

Time to reach destination is important factor of engineering requirement. The autonomous irrigation vehicle has to meet the destination midpoint within the specific time. The mesh network provides the sensor details such as which sensors need water and where is located in the two dimensional array. Reaching the midpoint on quickest time is mostly depend on the how effectively the path finding algorithm works. Nearest neighbor algorithm (NNA) is chosen as a path finding algorithm because it starts from a location and chooses the nearest unvisited destination by comparing the lowest distance to travel from its current location to all other available destination. And the algorithm repeats the same process from the next destination. Also, this nearest neighbor algorithm generates several paths, in case if all the neighbors are equally close to each other then it takes the shortest path to reach the destination. Having the algorithm to do this processes, it is guaranteed that the autonomous irrigation vehicle reaches the destination on time. In addition, in the house of quality time to reach destination constraint is represent in negative polarity because the goal is to reduce the autonomous irrigation vehicle travel time. Also, it has positive correlation with cost and power consumption which is marketing requirement components. The faster the autonomous vehicle moves the higher the price would be on market. Evidently, decreasing the travel time will make the vehicle more desirable

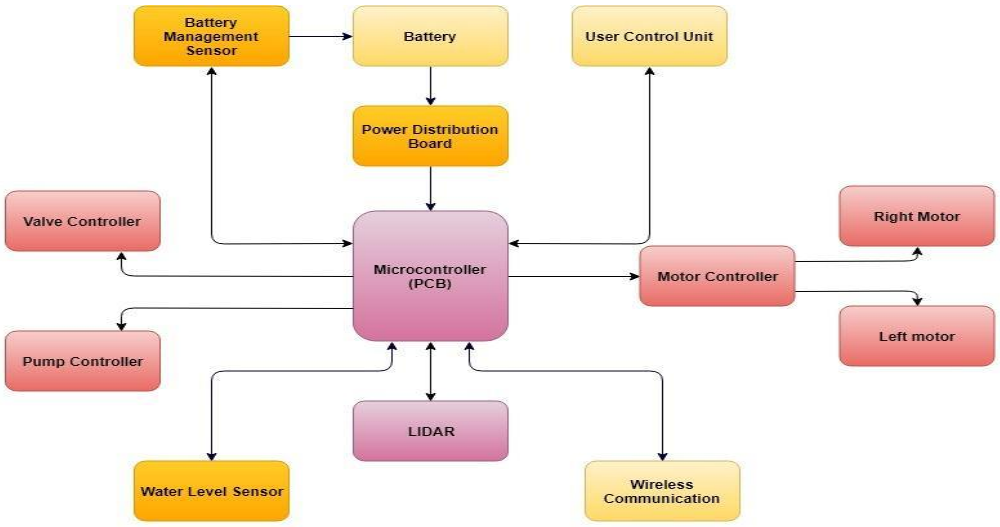
3. Hardware Diagram

The hardware diagram seen in Figure 5 starts with the power which is provided by a battery that can withstand the outdoor elements. The battery is connected to a power distribution board as seen in Figure 5. The power distribution board will be responsible for regulating the voltage and delivering the proper amount of power to the microcontroller and other elements that might not be powered through the printed circuit board that the microcontroller is mounted on. Once the printed circuit board provides power to the microcontroller, which will act as the brains of the autonomous irrigation vehicle, the various sensors will be able to send and receive information to/from the microcontroller through the printed circuit board.

One sensor was responsible for monitoring how much charge is left in the battery. The software will be programmed to use the battery management sensor as seen in Figure 5 so that the autonomous irrigation vehicle always has enough charge in the battery to make it back to home base where it can refill its water tank and recharge its battery. The battery management sensor could have been designed onto our custom printed circuit board or a pre-fabricated silicon chip sensor may have been purchased. The battery management sensor was not implemented.

Another monitoring sensor was the water level sensor as seen in Figure 5. The purpose of this sensor is to monitor the amount of water in the tank so that the autonomous irrigation vehicle knows when to return to home base and refill its tank. These sensors come in various types. The pressure level sensor works by detecting the amount of pressure when the tank is full and as the level changes so does the pressure. The ultrasonic and radar variants work in a similar fashion by sending a signal and recording the amount of time taken to reach the water level. The capacitance sensor as a rod in the tank that sends an RF signal and measures the capacitance between the rod and the tank. Which of these types will used will have to be further researched to come to a solid conclusion. The design was changed and this sensor was not implemented.

The autonomous irrigation vehicle receives user input through a user control unit. It will be attached to the microcontroller so that the user can turn off, reset, and override the moisture sensors to send the autonomous irrigation vehicle to water any specific areas desired. In the finale design the user control unit ended up becoming one button that launched the main algorithm that controls the vehicle. The microcontroller will also have wireless communication capabilities and use WiFi, Bluetooth, or a certain radio frequency to send and receive information from the mesh network of moisture sensors. The final design used Bluetooth.



Block Description	Block Satus
Battery Management Sensor	Research / Not Acquired
Battery	Research / Not Acquired
User Control Unit	Research / Not Acquired
Power Distribution Board	Research / Not Acquired
Microcontroller (PCB)	Research / Not Acquired
LIDAR	Research / Not Acquired
Wireless Communications	Research / Not Acquired
Water Level Sensor	Research / Not Acquired

Figure 5: Hardware Diagram

The vehicles vision component was handled by the LIDAR as seen in Figure 5. It uses optical sensing to measure the distance to an object by illuminating it with light. The LIDAR works by using a rotating mirror to sweep the light beam 360 degrees, and a receiver records the time taken by the light beam to return to calculate the distance to different objects. The motor controller, as well as left and right motors are managed by a microcontroller to move the autonomous irrigation vehicle through the yard. The valve controller and pump controller will be turned on and off by the microcontroller to regulate the flow of water. final design did not incorporate a pump controller or a valve controller.

3.1 Revised Hardware Diagram

After multiple meetings with the mechanical engineering students and other electrical and computer engineering students that compose the Guard Dog Valves team, the project went through several major design changes and a revised hardware diagram has been developed and can be seen in Figure 6. From the top down the first major change is that the autonomous irrigation vehicle will no longer be similar to a fire truck with a tank of water but will instead have a hose attachment for water delivery. The water tank was removed due to weight because carrying enough water to irrigate a 20x30 plot of grass proved to be significantly heavy. Since the water tank has been removed there will be no need to have a pump controller since there will be no pump and all the pressure will be coming from the hose attached to the customer's home.

Another portion that has been removed because there will no longer be a tank is the water level sensor, since there is no need to detect the level of water left in the tank. The valve controller has also been removed from the design because the microcontroller printed circuit board will not be driving the solenoid that controls the water flow coming from the hose. The valve controller that controls the water flow has been moved to the home base. The reason being that if it were to be on the autonomous irrigation vehicle the built-up pressure when not watering would weight it down. The extra weight would need more power to shortening the battery life time of the autonomous irrigation vehicle. The alternating current power cable that was talked about in the project description which was going to be tethered to the autonomous irrigation vehicle has also been removed making battery management and conservation an even more important aspect of the project.

The wireless communication that was going to be used for the mesh node network team of electrical and computer engineers has chosen the E01-ML01IPX which enables the autonomous irrigation vehicle to communicate with the mesh network of sensors. Its uses the 802.15 standard and sets up a wireless personal area network with the all the nodes and the autonomous irrigation vehicle that does not interfere with the customer's WiFi. It is also low cost and more energy efficient than regular WiFi. The final design did not incorporate this chip and was replaced by Bluetooth.

As a backup obstacle avoidance system, we used ultrasonic range finders which work similar to the LIDAR but instead of light they use a specific frequency of sound waves to calculate the distance from the autonomous irrigation vehicle to the obstacle in its path. A LIDAR servo has been added to the hardware diagram to make a 180-degree LIDAR that is water proof. The details are explained in section LIDAR Setup. An Arduino microcontroller has been added as part of the autonomous irrigation vehicle's final design.

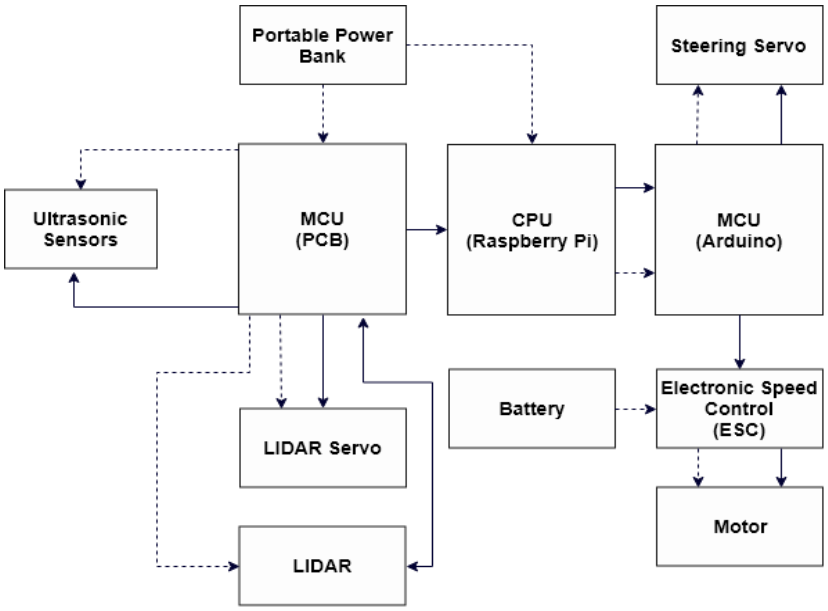


Figure 6: Revised Hardware Diagram

3.2 Existing Related Projects

While conducting research, our group could not find any projects that related to autonomous irrigation vehicles. We did find a couple of projects that, when combined, have similar concepts to our overall project. The senior design project we researched is called Fundamental Agriculture Resource Monitor (F.A.R.M.). F.A.R.M is closely related to our team's original idea of innovating the yard's valve layout and adding mesh network of sensors to efficiently and autonomously water the grass. The F.A.R.M project proposed using an array of sensors including, solar, water, humidity, and temperature. Our idea was to use weather, moisture, flow, rain, and temperature sensors. One of the main ideas we borrowed from the F.A.R.M project was researching more about the ZigBee Radio Transceiver for the mesh network and other related alternative hardware

technologies for the communication between the autonomous irrigation vehicle and the mesh network of sensors.

Once our team decided to focus on the autonomous irrigation vehicle, new research had to be conducted. Searching for related projects our team came across a project done in 2013 called The Manscaper Autonomous Lawn Mower. While it does not have anything to do with irrigation one of the main tasks of The Manscaper is to navigate the lawn and avoid objects using robot vision autonomously. Then Manscaper proposed to use an ultrasonic PING sensor in combination with a webcam to avoid objects in the lawn. The ultrasonic PING sends out an ultrasonic signal, and if the signal does not bounce back from an object, the lawn mower continues to move forward. The webcam is used in conjunction with an edge detection algorithm. Our autonomous irrigation vehicle proposed to replace the ultrasonic PING and webcam with a LIDAR to serve the same purpose. The LIDAR works in a similar fashion to the ultrasonic PING sensor except that the signal is light-based instead of sound. The other difference is the LIDAR sweeps 360 degrees. The LIDAR detects the shape of the object using the reflected light, similar to the webcam function on The Manscape. In the final design an ultrasonic PING sensor was added to aid the LIDAR in object avoidance.

3.2.1 Existing Related Products

While performing research on related products in the marketplace we came across two products that are similar to the autonomous irrigation vehicle our team is designing:

The first is called Droplet. According to the Droplet website "*Droplet is the world's first smart sprinkler system that combines the latest technology in robotics, cloud computing and connected services to transform the way sprinkler systems function. Droplet keeps your plants healthy without wasting water by drawing upon a vast system of data to intelligently determine how best to care for your plants. By being smart, precise and frugal, Droplet can lower your sprinkler water consumption by up to 90%; on average Droplet saves $263 a year!*" (Droplet) While the description of their product seems extremely similar, the implementation is quite different. The first difference is that the Droplet requires the user to use a smart device to enter information about the location and type of vegetation so that the droplet knows when and where it needs to water. The autonomous irrigation vehicle that we are designing uses a mesh network of sensors in combination with RF chips to autonomously determine when and where to water. The RF chips where not implemented in the final design. In order to precisely deliver water, the Droplet uses live data from weather stations as well as a database of soil samples to know how much to water. Our autonomous irrigation vehicle uses soil moisture sensors designed by a mechanical engineering group in 2017. The project was called Integrated Water Monitoring System and was also sponsored by Guard Dog Valves. The Integrated Water Monitoring System is connected to the mesh network to accomplish this function. While the Droplet has an easier installation and setup – since it does not require a mesh

network of sensors – our irrigation provides more efficient water allocation because our data is coming from the consumer's lawn and not a database.

Another product out on the market that has inspired our autonomous irrigation vehicle is the Roomba robotic vacuum. While it may not have anything to do with irrigation, the main concept of our autonomous irrigation vehicle is applied to an indoor autonomous robot. The Roomba tasks are to vacuum multiple rooms in a house while maneuvering around objects on the house floor. Another major feature that the Roomba has that our autonomous irrigation vehicle will have is what the Roomba website calls Recharge and Resume which means the Roomba "*Automatically recharges itself and resumes cleaning until the job is done*" (Roomba). The autonomous irrigation vehicle will does not have the capability of irrigating the lawn and if it runs out of battery during the irrigation go back to home base to recharge and finish watering the rest of the lawn.

The robot vision and object detection for the Roomba is handled by a camera that points forty degrees up in front of the Roomba. The data received from the camera is processed using a Vision Simultaneous Localization and Mapping (VSLAM) program. This program works by taking pictures received from the camera and looks for distinguishing patterns in the pixels of the pictures. With these patterns, it builds a map of its environment allowing it to learn and remember where objects are located. The Roomba does this indefinitely as objects in a house are moved around and continues updating its map of the house.

Our autonomous irrigation vehicle uses the LIDAR to detect objects instead of the camera. The LIDAR sweeps its laser a hundred and s eighty degrees and calculates how much time the light from the laser takes to come back to the LIDAR; the time is used to determine how far away an object is and to take the proper steps to avoid the object. The mapping portion of the autonomous irrigation vehicle uses the mesh network and sets up a two-dimensional array map. As the autonomous irrigation vehicle runs into objects or obstacles it updates the two-dimensional array in order to remember their positions. Once the map is updated the next time the autonomous irrigation vehicle is summoned, it will takes the fastest route while avoiding the obstacles.

The Roomba uses Wi-Fi to connect the users to an app which allows the user to set up a schedule for when it is convenient for the user for The Roomba to vacuum the house. The autonomous irrigation vehicle uses Bluetooth to establish a mesh communication network that the vehicle uses to build a map and navigate the yard as well as communicate with the moisture sensors and decide which areas need watering. The autonomous irrigation vehicle also was going to have a user control unit on the vehicle and a GUI for the mesh network. This was going to allow the user to override the autonomous irrigation so that the vehicle may water whatever yard zone the user desires. The option to schedule when the autonomous irrigating vehicle is allowed to water is actually a must because depending on what state and county you live in the government does not allow the lawn owner to water whenever they want. The watering schedule permitted depends on a lot of factors. The number of days depends on whether it's daylight savings time or not. Second, the time of the day you are allowed to water restricts the hours so that residents do not water when the sun is hottest because if they do, the

water evaporates before the lawn can properly absorb the water. Another restriction is that each zone can only be watered a maximum of one hour during the allowed days and hours. These restrictions may not have to be followed or may vary depending on where the user's property is located. While the Roomba's Wi-Fi enabled scheduling, feature is there for the convenience of the customer the autonomous irrigation vehicle scheduling feature is mandatory due to government regulations as well as the convenience of the customer. Also, the Bluetooth communication for the autonomous irrigation vehicle is a core feature for the basic function of the autonomy, while the Roomba does not require the Wi-Fi for its autonomy.

3.3 Strategic Components

The autonomous irrigation vehicle requires a certain list of components that help ensure its successful design, construction, and proper execution of all of its desired assignments. Each component has been carefully and extensively researched to make certain that the autonomous irrigation vehicle executes its desired goals. As the autonomous irrigation vehicle has gone through a couple of revisions, there have been a handful of components that have been discontinued or modified to attain the desired results.

3.3.1 Sensors

Sensors play an integral role in the autonomous irrigation vehicle's responsibility of traversing terrain, detecting objects, proper dispensation of water and monitoring its current battery charge. Each sensor was specifically chosen to cater to the given requirements of the autonomous irrigation vehicle. The autonomous irrigation vehicle is outfitted with sensors that promote its success when it is out in the field while performing its specified tasks. Although a few sensors have been dropped from the autonomous irrigation vehicle's final design, the main sensors have stayed and been implemented.

3.3.1.1 LIDAR

The light detection and ranging sensor, or LIDAR for short, is an imperative component for the autonomous irrigation vehicle. The LIDAR enables the autonomous irrigation vehicle to see its surrounding environment. By providing the autonomous irrigation vehicle with a sense of sight, the vehicle is able to detect obstacles that can possibly obstruct its path. With the autonomous irrigation vehicle equipped with the LIDAR sensor,

it is able to traverse to the designated area to commence proper watering within suitable time through obstacle avoidance.

3.3.1.2 Power Management Sensor

The Power Management Sensor was going to be necessary because the autonomous irrigation vehicle was going have the capability of returning to home base in order to recharge its battery. The power management sensor was going allow the microcontroller to determine when the autonomous irrigation vehicle needs to return by polling the battery every so often and when a set battery life is reached the autonomous irrigation vehicle will automatically know to return to home base. The power management sensor was going to allow the autonomous irrigation vehicle to know how efficient it should run in order to get back to the changing station.

3.4 Power Distribution Board

The power distribution board was going to provide an organized way of connecting the LIDAR, ultrasonic sonic range finders, servo, and radio frequency module to the battery of the autonomous irrigation vehicle. A typical power distribution board has negative labeled pads or terminals that are all connected. Similarly, power distribution boards have positive labeled terminals or pads that are all connected. The fact that they are labeled and are all connected make it easy to solder the black wires to the negative terminals of the power distribution board and all the red wires to the positive terminals or pads of the power distribution board. Once this is done, assuming the battery is connected to the power distribution, all of the components will have power. Some power distribution boards include additional battery eliminator circuits which is a fancy way of saying voltage regulator. The voltage regulator circuit on the power distribution power is responsible for regulating the batteries voltage to a specific voltage required by the devices. A power distribution board without a voltage regulator must have all components solder to it run at the same voltage. The reason voltage regulators are called battery eliminator circuits is because if the power distribution board has a voltage regulator you may attach components with different voltages to the same power distribution board eliminating the need for a different voltage battery. When deciding which power distribution board was needed the team first found how much current was passing through it which meant find out what the maximum current of each of the components attached requires. The team had still not decided if they were going to use more than one battery or not. The reason we were still unsure was that we believed that the battery that came with the drive train was able to provide power to all the components specially because it does not run at full throttle since the purpose is to avoid obstacles while watering a yard or lawn. The concern with using the battery from the drive train was the potential of damaging the battery trying to splice the wires. Now that the team knows exactly what battery came with the drive train the team added a second battery for all the electronics.

3.5 User Control Unit

The user control unit was going to be a set of buttons that gave the user options to override the autonomous functions of the vehicle. The first button was going to turn on the autonomous irrigation vehicle. The second button was going to turn off the autonomous irrigation vehicle. In case the autonomous irrigation vehicle is malfunctioning or not performing up to customers satisfaction there was going to be a reset button that was going to reset all microcontroller and Raspberry PI. Another user feature that was going to be controlled by a button on the user control unit is a manual override to water a specific node. This feature is a great for marketing purposes because if a home owner plants a new patch of grass that requires a little more water than usual even if the moisture sensor is not notifying the autonomous irrigation vehicle the customer can override the moisture sensors and water that specific area of grass. The user control unit could also of hade a LED screen that displays the battery life of the autonomous irrigation vehicle. The same LED screen could have also displayed the node area it is traveling to go water. All these functions could have been operated by push buttons or switches for the functions that only have one choice. For the override to go water a specific node feature, which would have required more than one choice, a rotary switch or a dial was going to be used. All the buttons were going to have two wires. One of the wires was going to run to ground, so all the ground wires of the buttons would be daisy changed together and attached to a ground on the printed circuit board. The other wire of the buttons would be attached to a pull up resistors. The ground and pull up resistors would have to be designed directly on the printed circuit board. Another option would have been to have the buttons attached to the Raspberry PI which depended on which model we used and could have programable pullup/pulldown resistors. In the end the user control unit ended up being one button activated the main path finding algorithm.

3.6 Printed Circuit Board (PCB)

The printed circuit board was going to be mounted inside of the enclosed autonomous irrigation vehicle frame pictured in Figure 8.

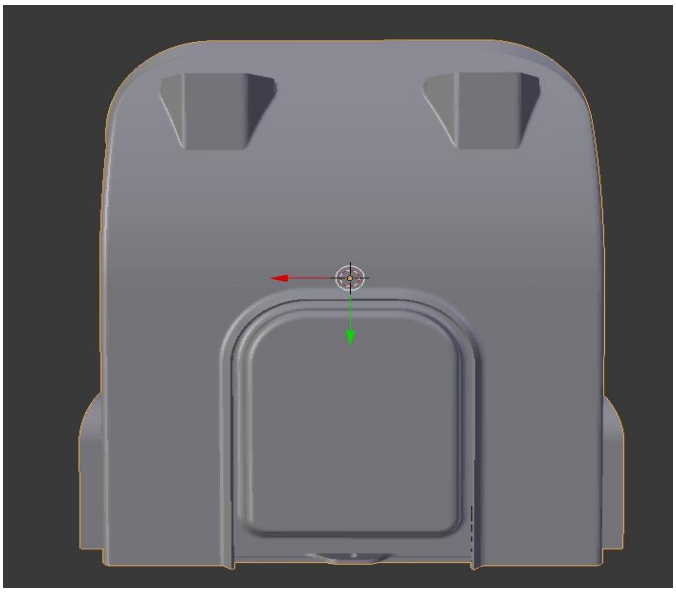


Figure 7: Autonomous Irrigation Vehicle Prototype Frame

The enclosure featured a sliding door on the back of the frame pictured in Figure 9 for easy installation and maintenance of the printed circuit board and other enclosed electrical components. The custom printing of our circuit board guarantee that all the required input and outputs for power, motors, and sensors were met. The custom printed circuit board also guarantee it will be small enough to fit inside the enclosure and would be properly protected from minor collisions and outdoor weather. Another benefit from a custom printed circuit board is that it guaranteed ability to modify it according to design changes. The finale design did not include a enclosed frame.

The custom printed circuit board was fitted with a ATmega328 made by ATMEL for all our projects processing needs. The ATmega328 8-bit processor was going to provided sufficient processing power for the controlling the LIDAR, the motor control unit, battery monitoring unit, valve controller, pump controller, water level sensor, and the wireless communication device. In the finale design the ATmega328 8-bit processor provided sufficient processing power for the controlling the LIDAR, Servo and ultra sonic range finder. Basic on, off, and reset buttons on the user control unit were going to be connected to the printed circuit board. In the finale design the Raspberry Pi ended up having one button that activated the main path finding algorithm and the PCB did not have any buttons.

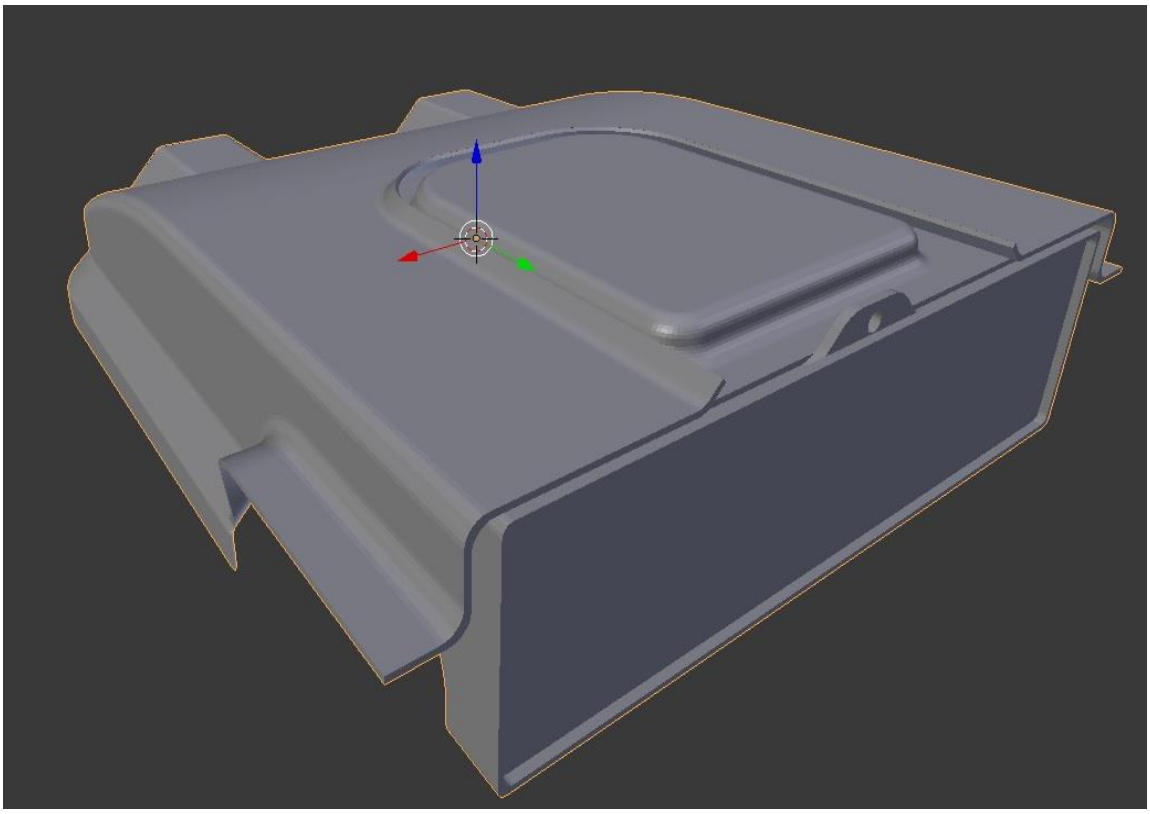


Figure 8: Autonomous Irrigation Vehicle Prototype Sliding Door

The autonomous irrigation vehicle operates as effectively and efficiently as possible as what the printed circuit board would allow. In order for any kind of process to begin, for a servo to move, or to transmit a data instruction set, there will need to be a source of power. In order for each of the sensors and electronics to work together, there must be a source of power but also, an appropriate route in which the power is directed towards. Here's where the printed circuit board design comes into play. The printed circuit board will be in charge of how each of the pieces received power, the intermediary between sensors and electronics communication as the signals will pass through the circuit board.

3.6.1 Specification

In terms of the specifications of the printed circuit board, there are only a couple that are of utmost concern in regards to the autonomous irrigation vehicle's success and operational abilities. The specifications for the printed circuit are simple in that the printed circuit board needs to be able to draw as little power as possible. On top of low power draw of the autonomous irrigation vehicle's printed circuit board, the printed circuit board also needs to be a small size in order for it to fit inside the autonomous irrigation vehicle's chassis.

3.6.2 Schematic

As seen in the schematic below, there are multiple connections that must be made in order for the printed circuit board to operate as needed and to be properly used by the autonomous irrigation vehicle. The microcontroller itself is soldered to the various sensors on board the autonomous irrigation vehicle. The servos that are used to give the autonomous irrigation vehicle a forward driving motion are operated by the motor controller unit which is also connected to the microcontroller. There is also be another servo motor that is used to control the direction the LIDAR is facing. With the movement of the autonomous irrigation vehicle taken care of, the connections of the sensors are now to placed strategically. Below is a rough draft, Figure 10, of the proposed schematic for the printed circuit board for the autonomous irrigation vehicle.

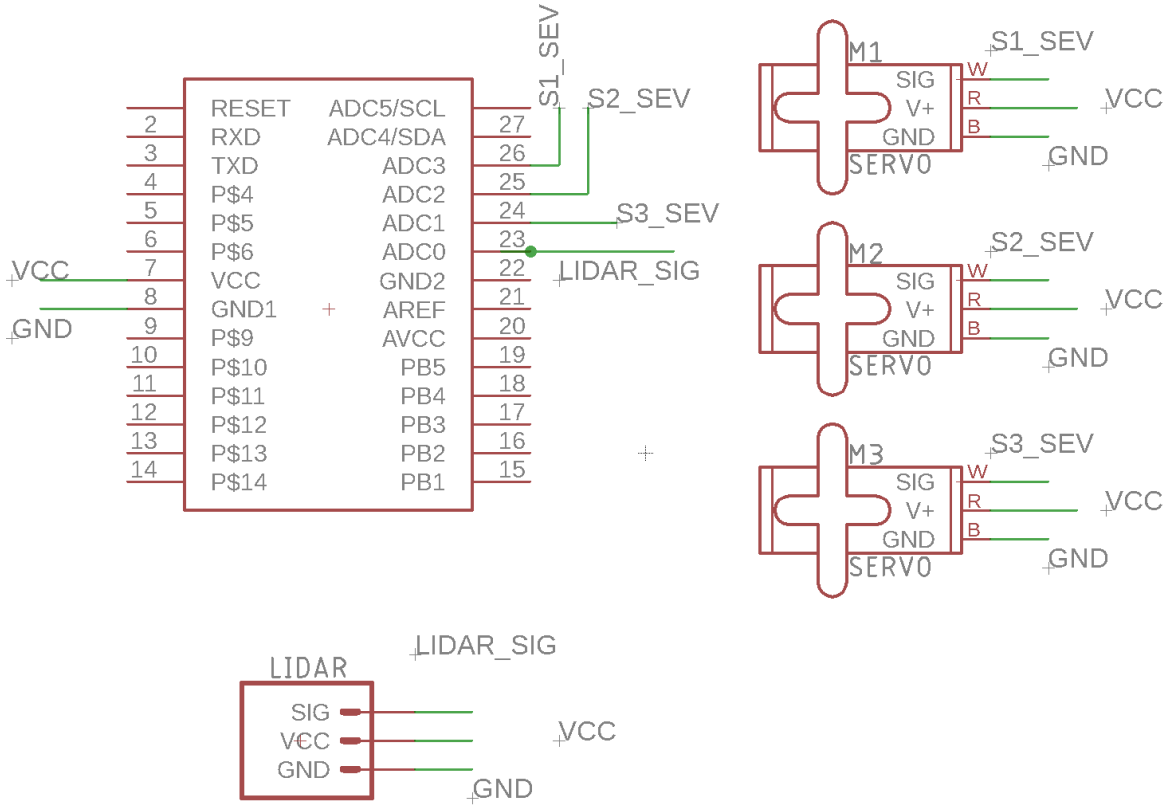


Figure 9: Printed Circuit Board Rough Draft

Placement of the sensors are placed in the most optimal positions that can be allowed on the printed circuit for the autonomous irrigation vehicle. As it currently stands, the autonomous irrigation vehicle is using two sensors to navigate through any given terrain. The sensors mentioned are the LIDAR and the ultrasonic detection. The LIDAR behaves as a set of eyes for the autonomous irrigation vehicle in which it maps out the objects that are directly in a 60 degree field of view in front of the autonomous irrigation vehicle.

The ultrasonic detection sensor will be providing support to the LIDAR by supplementing the detection abilities already provided by the LIDAR. The battery management sensor will be in charge of monitoring the charge of the on board battery of the autonomous irrigation vehicle. Below is the final PCB schematic that was used for the autonomous irrigation vehicle Figure 10.

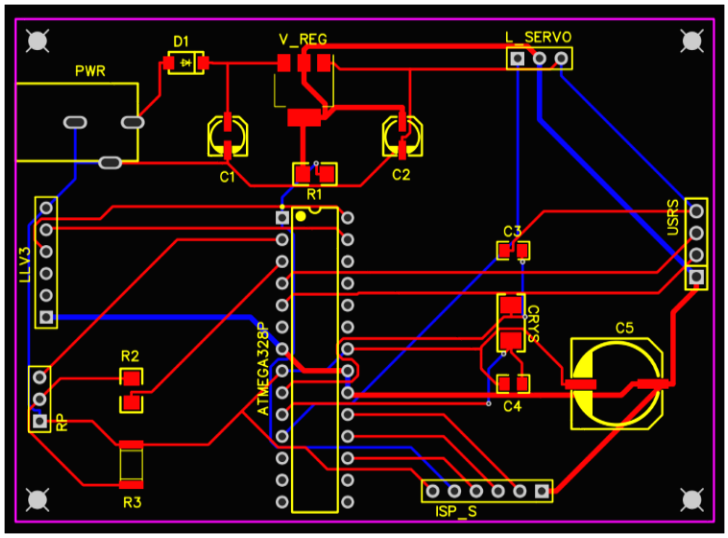


Figure 10 – Final PCB Schematic

3.7 Motor Controller

In order for the autonomous irrigation vehicle to traverse any given terrain, it requires motors to drive the wheels. Although the motors are operational, there are additional controls and regulations that need to be implemented so as the motors of the wheels do not draw an excess amount of power. In order to regulate and minimize the power draw of the motors, a motor controller is implemented. On top of regulating the power draw of the motors, the motor controller is also be in charge of the speed and direction of rotation of the motors by manipulating the voltage sent to the motors. Along with regulating speed, direction, and power draw, the motor controller plays another critical role by preventing an initial high power surge which could compromise the integrity of the wheel motors.

3.7.1 Software

Programming of the microcontroller unit is relatively straight forward. The microcontroller requires no intense or complex algorithm in order for the unit itself to operate as intended. The microcontroller's programming code is efficient and sufficient enough so that all of the operations that it's carrying out, are executed without any kind of flaw. As such, there

is a need for an integrated development environment software which allows the microcontroller to be programmed. The programming code of the microcontroller itself needs to meet exact requirements and specifications that will be needed by each of the sensors and onboard electronics.

The software that is programmed into the microcontroller unit for the autonomous irrigation vehicle is short and concise yet efficient and reliable enough for the vehicle to operate as desired. For the autonomous irrigation vehicle to accomplish its given task, the programming code incorporates all of the sensors on the vehicle itself. The programming code not only has the onboard computer processor interacting with the sensors but also has each sensor interacting with any other desired sensor or electronic device. Since the autonomous irrigation vehicle is using its onboard sensors to navigate through the terrain, it needs to be constantly updating and maneuvering its sensors and servos as needed.

The autonomous irrigation vehicle doesn't require any mobile application. The environment used to develop in is Visual Studio or Eclipse which are an Integrated Development Environments (IDE). We chose to use Integrated Development Environment (IDE) because it is easy to maintain different classes, variables and functions as well as it has the IntelliSense which is a useful feature which will correct any spelling error in the code while writing. In addition, using Integrated Development Environment will make it easier to debug the code. The debug feature is able to stop the program at any specific point also just to run one function at a time to check where the errors are occurring and have it avoid any spelling mistakes. The Integrated Development Environment would be easy to collaborate and manage the project. Another advantage of using Integrated Development Environment is that it will increase efficiency of the program. The high-level language will be used for path learning algorithm such as C++. C++ was chosen because it is one of the powerful language and it provide more controls compared to other languages. Besides, it increases the performance and compare to other programming language C++ considered to be the faster one. Nowadays C++ is used in most of the industries, so it is a much-known programming language for developer. Since C++ has more functionality, it would be the more reliable programming language to use for robotics.

The path learning algorithm is the nearest neighbor algorithm (NNA). The nearest neighbor algorithm is chosen for this autonomous irrigation vehicle project because nearest neighbor algorithm starts from a location and chooses the nearest unvisited destination by comparing the costs to travel from its current location to all other available destinations and selects the one with the lowest cost for it to move to and repeats the process from that next position until all destination midpoints have been visited. This will generate at least as many paths as there are destinations, more if at any point there are multiple neighbors who are equally close to each other and the path needs to split. From all the paths generated, the shortest distance path, or the path with the lowest cost if dealing with other constraints, is chosen. Also, the high level language will be used for microcontroller part as well as the open source libraries will be used.
Software requirements

- The autonomous irrigation vehicle should receive the sensor data from mesh network
- The software shall be able to calculate the nearest midpoint using NNA algorithm.
- The software shall be able to detect the obstacles and avoid collision
- The software shall be able to update the two dimensional matrix array every time it travel to the midpoints.
- The software shall be able to send a feedback to sensor to indicate the dryness level

3.7.2 Energy Consumption

As the autonomous irrigation vehicle carries out its given objectives and tasks, it will be consuming a certain amount of power. Power draw is a serious concern for any given system. With predetermined tasks of the autonomous irrigation vehicle, power draw of the microcontroller has been brought down to a minimum. The lower the power consumption of the microcontroller, the more power can be allocated to the other components of the autonomous irrigation vehicle. As such, the microcontroller has a minimum power requirement in order for the microcontroller unit to operate.

A benefit of a microcontroller, or any microcontroller for that matter, is that a typical microcontroller's power consumption is a relatively small value. That is to say that on average, a microcontroller unit's current draw is roughly about 500 nanoamps at 2 volts when it is on standby mode. When the microcontroller is being operated and carrying out its programmed tasks, the current draw jumps to 15 microamps at 2 volts and 32 kilohertz and 170 microamps at 2 volts and 4 megahertz. With the microcontroller operating at such a low voltage and having a low current draw, it's power consumption is also be low. With the microcontroller's power consumption being on the low spectrum, it allows the autonomous irrigation vehicle to have a higher amount of power to route to the other sensors and electronics that require power.

3.7.3 Comparison Between MCUs

The autonomous irrigation vehicle's success depends on the operational efficiency that the microcontroller can produce. Efficiency is an important topic for the autonomous irrigation vehicle as it will be dispensing water and tending the lawn. The microcontroller runs on as little power possible as well as handling an instruction set from programming code. As such, there have been multiple microcontroller units that have already been considered such as the PIC16F87A -I/SP depicted in Figure 11.

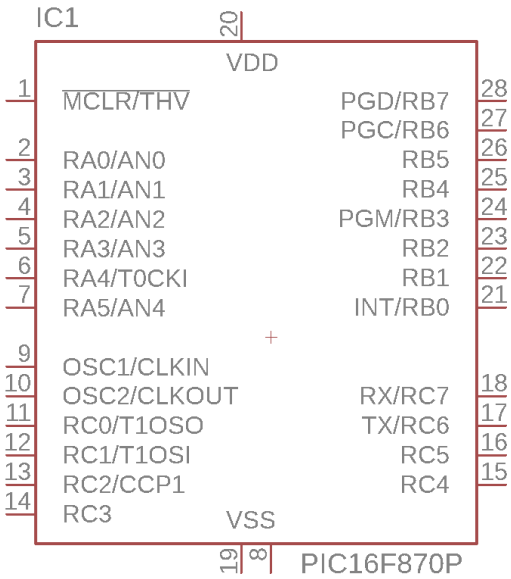


Figure 11: PIC16F87A-I/SP microcontroller

First candidate consideration for the autonomous irrigation vehicle's microcontroller unit is the PIC16F87A-I/SP microcontroller as pictured above. The PIC16F87A-I/SP microcontroller itself is a high performance reduced instruction set computer central processing unit. With an operating speed set at 20 megahertz and 200 nanoseconds, the PIC16F87A-I/SP is able to process 35 single-word instructions. On top of that, the PIC16F87A-I/SP boasts a flash memory size of 14.3 kilobytes or 8192 words. Most importantly is that the PIC16F87A-I/SP itself has 22 I/O pins and 3 I/O ports. The PIC16F87A-I/SP also provides, a variety of timers from 8-bit timer/counter with an 8-bit prescaler and a 16-bit timer/counter with prescaler. There is also a synchronous serial port which operates under two modes, SPI Master and I2C Master & Slave.

The next candidate to be the autonomous irrigation vehicle's microcontroller is the PIC18F258-I/SP microcontroller as pictured below in Figure 12. As with the last microcontroller, the PIC16F87A-I/SP, this candidate's performance surpasses the previous microcontroller. The operating speed of the PIC18F258-I/SP is at 40 megahertz and up to ten million instructions per second. Along with its speed, the PIC18F258-I/SP's flash memory has been upgraded from 14.3 kilobytes to 32 kilobytes which is equivalent of 16,384 words. The PIC18F258-I/SP also comes with added features such a power saving sleep mode, a selectable oscillator options from 4x phase lock loop of primary oscillator and a secondary oscillator clock input which operates at 32 kilohertz. As before, the PIC18F258-I/SP also has 22 I/O pins as well as 3 I/O ports. There is also a high current sink and source of 25 miliamps. The PIC18F258-I/SP also comes with 4 timers. The PIC18F258-I/SP also comes with a controller area network communication protocol which allows the node with the higher address number to yields the bus to the lower addressed node. This could help the autonomous irrigation vehicle communicate effectively between all of the other sensors and electronics on board the vehicle.

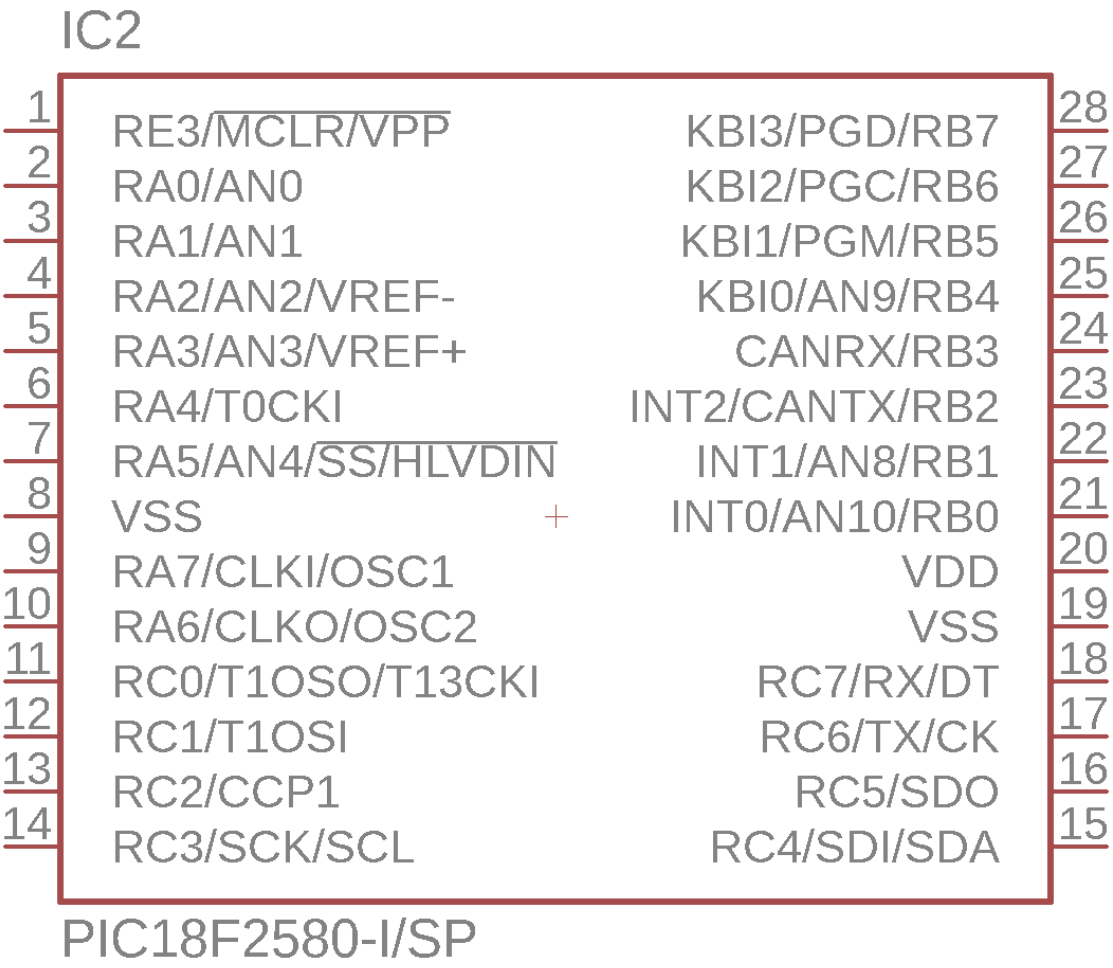


Figure 12: PIC18F258-I/SP 28 DIP

The ATMEGA328P-PU microchip is also being considered because it's a low cost and low power chip allowing the team to save on money and power consumption. The ATMEGA328P-PU features 32 registers and a robust instruction set. The Arithmetic Logic Unit and 32 registers are directly connected to each other. This feature allows two registers to independently be accessed in one instruction set and both be executed in one clock cycle. These architectures feature mentioned above are code efficient which leads to faster throughput.

As such, the ATMEGA328P-PU with an advanced reduced instruction set computer architecture. Alongside that feature, offers high endurance non-volatile memory segments. The ATMEGA328P-PU carries 23 kilobytes of in-system self-programmable flash program memory, data retention of less than 1 PPM over 20 years at 85 degrees Celsius and then at 100 years at 25 degrees Celsius. This memory retention is an ideal feature as the autonomous irrigation vehicle will certainly be operating at degrees of 100 degrees Fahrenheit thus keeping any memory loss during the vehicle's life cycle.

The ATMEGA328P-PU peripheral features should also not be underestimated. The ATMEGA328P-PU comes with two 8-bit timer and counter with separate prescaler and compare mode. Alongside the two timers, there is a 16-bit timer and counter with separate prescaler, compare mode, and a capture mode. The ATMEGA328P-PU also carries a real time counter with a separate oscillator, 6 pulse width modulation channels, six of which are 10-bit ADCs with a temperature measurement. As seen in Figure 13, the ATMEGA328P-PU contains 23 programmable I/O pins out of the total 28 pins.

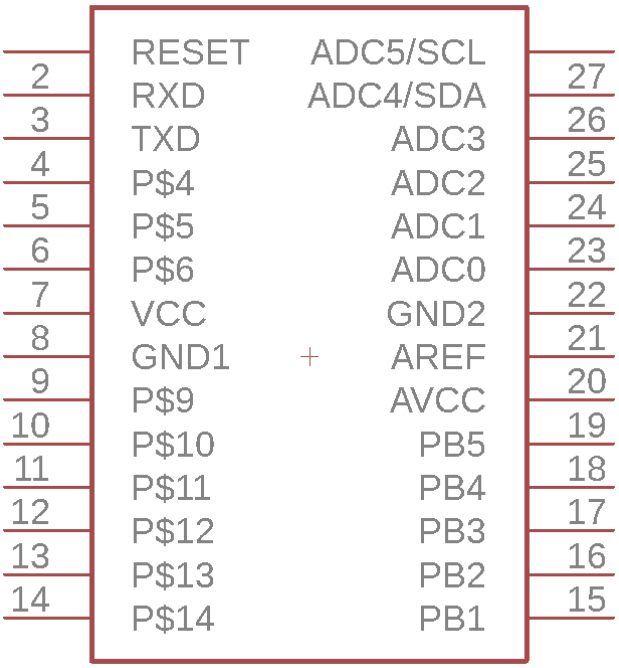


Figure 13: ATMEGA328P-PU

The ATMEGA328P-PU also comes with a programmable serial universal synchronous and asynchronous receiver, a master and slave serial peripheral interface, and a byte orientated 2 wire serial interface which is compatible with Philips I2C. Finally, for the peripherals features is the programmable watchdog timer with a separate on-chip oscillator, an on-chip comparator, and an interrupt and wake up on pin change.

The ATMEGA328P-PU also comes with a host of special features that separate it from the ATMEGA family of chips. Of the special features, the ATMEGA328P-PU comes with a power-on reset, a programmable brown-out detection, and an internally calibrated oscillator. There is an additional feature of the ATMEGA328P-PU which is the external and internal interrupt sources. This feature allows the ATMEGA328P-PU microcontroller unit to be put into a low power state whilst waiting for an interrupt, from either an external or internal interrupt source, before executing a programmed instruction set and to conserve power. Last but not least, is that the ATMEGA328P-PU comes with six sleep

modes. The six sleep modes the ATMEGA328P-PU comes with are the idle, ADC noise reduction, power-save, power-down, standby, and an extended standby mode.

The ATMEGA328P-PU comes with a multitude of features that can greatly benefit the operation processing of the autonomous irrigation vehicle. As such, the ATMEGA328P-PU also has varying speeds at which it operates at as it processes instruction sets. The ATMEGA328P-PU can operate at a speed of 3 megahertz from 1.8 volts to 5.5 volts, 10 megahertz from 2.7 volts to 5.5 volts, and 20 megahertz at 4.5 volts to 5.5 volts. One of the most attractive feature of the ATMEGA328P-PU is its low power consumption as previously stated. In the ATMEGA328P-PU active mode, the current draw is .2 milliamps at 1 megahertz, 1.8 volts at 25 degrees Celsius. Whilst the ATMEGA328P-PU is in power down mode, the current is .1 microamps at 1 megahertz, 1.8 volts at 25 degrees Celsius. The power save mode of the ATMEGA328P-PU has a current draw of .75 microamps, which includes a 32 kilohertz RTC, at 1 megahertz, 1.8 volts at 25 degrees Celsius.

As the aforementioned details have stated, there is a multitude of microcontrollers that can accomplish and achieve the needs of the autonomous irrigation vehicle. While the performance of the microcontroller will affect the performance of the autonomous irrigation vehicle, the microcontroller will need to be of a rugged profile. That is that the ATMEGA328P-PU can endure a large load, continuous operation in a very warm environment, and has a low power usage which promotes a long life time. As such, the proposed microcontrollers has a decisive candidate but is still subject to change during the construction of the autonomous irrigation vehicle. The autonomous irrigation vehicle will operate as needed and desired regardless of the selection of the microcontroller unit since the duties of the microcontroller unit will only be limited by how much energy is provided to it and how durable it is in regards to its surrounding environment

3.7 Wireless Module

The autonomous irrigation vehicle travels about its given terrain and as such, does its best to avoid any obstacles to reduce the chances of collision. The autonomous irrigation vehicle avoids whatever obstacles are in its path in order to reach the designated area to begin dispensing water. Although the autonomous irrigation vehicle is able to move forward and avoid any obstacles within its path, the autonomous irrigation vehicle needs will need the additional data provided by the mesh network of sensors. The additional data provided by the mesh network of sensors guide of the autonomous irrigation vehicle by providing the vehicle with a dryness reading.

Since the data that is provided by the mesh network of sensors, it only makes sense that the autonomous irrigation vehicle is also connected and plugged into the mesh network itself. That is that the autonomous irrigation vehicle is apart of the mesh network of sensors and is able to communicate with the sensors on the mesh network . For the autonomous irrigation vehicle to gain the ability to communicate with all of the sensors on the mesh network, a radio frequency module will be needed. The radio frequency module

has met the few certain requirements so that the autonomous irrigation vehicle can successfully accomplish any given task that is at hand.

3.7.1 Comparing Different Models

In order for the autonomous irrigation vehicle to achieve its success, its wireless module will need to meet a few requirements. The first of which is that it will need to be operating with the IEEE 802.15.4 standard. Another requirement will be a low power consumption to maximize the autonomous irrigation vehicle's time out navigating its given terrain on a given task. Since the module will need to fit inside of the autonomous irrigation vehicle's chassis, the size of the radio frequency module will be required to be small in dimensions. As such requirements are needed for the radio frequency module, a number of candidates have been selected.

First of the candidates to be the autonomous irrigation vehicle's main radio frequency module, is the MRF24J40 transceiver. The MRF24J40 transceiver operates using the IEEE 802.15.4 standard. The MRF24J40 transceiver has multiple models, Model A, Model B, and Module C, with different specifications. Model A of the MRF24J40 transceiver offers a current draw of 23 milliamps for transmission, 19 milliamps for reception, and 2 microamps while in sleep mode. Model B of the MRF24J40 transceiver operates with a longer range and thus has an active current draw of 130 milliamps for transmission, 25 milliamps for reception, and 5 microamps in sleep mode. The MRF24J40 transceiver's Model C current draw is 120 milliamps for transmission, 25 milliamps for reception and 12 microamps for it is in sleep mode.

The second candidate to be considered to be the autonomous irrigation vehicle's main radio frequency module is the E01-ML01IPX transceiver. The E01-ML01IPX transceiver itself operates at a low power consumption. The current draw of the E01-ML01IPX transceiver while it is transmitting is 12 milliamps, 11.5 milliamps for reception, and a 1 microamp while in sleep mode. Another added benefit of the E01-ML01IPX transceiver is its small size. The E01-ML01IPX transceiver comes in a 12 x 19 millimeter squared. As seen in Figure 14, the E01-ML01IPX transceiver has multiple holes to allow different kinds of data to be received and transmitted.

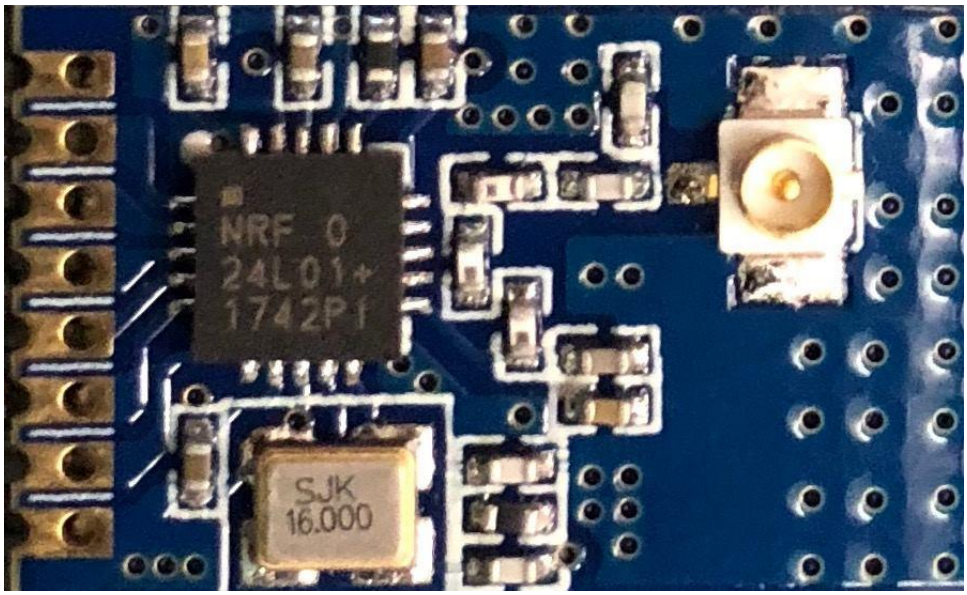


Figure 14: E01-ML01IPX Transceiver

With the E01-ML01IPX transceiver being able to receive and transmit signal data to and from the mesh network of sensors, the range will also be a factor. Although the autonomous irrigation vehicle location of operation will be within a 20 x 30-foot area, the range will be covering more than enough spacing to ensure effective communication. The 100m range of the E01-ML01IPX transceiver is a gratuitous amount needed for the autonomous irrigation vehicle's immediate surroundings but implies that it is possible to scale up the number of sensors for the mesh network.

Another candidate that is currently being considered to play the role of main radio frequency module for the autonomous irrigation vehicle is the ATZB-24-B0. The E01-ML01IPX transceiver has a current draw of 18 milliamps while transmitting and 19 milliamps for reception. The dimensions of the ATZB-24-B0 are relatively small as they are 18.8 mm x 13.5 mm x 2 mm thus allowing the module is easily fit inside of the autonomous irrigation vehicle's chassis. As the autonomous irrigation vehicle will most likely be operating in the outdoors because of the tasks it needs to accomplish, the inside of the chassis will warm up greatly. As heat could possibly be a problem, the ATZB-24-B0 can operate safely up to a temperature of 85 degrees Celsius. All these features provided by the ATZB-24-BO can possibly play a critical role within the autonomous irrigation vehicle's signal relay system. Due to some communication and design restrictions, the module was not implemented in the final design of the autonomous irrigation vehicle.

4. Related Standards

The standards we are adhering to for this project are received from Institute of Electrical and Electronics (IEEE). These IEEE standards are used to create specifications and procedures that guarantee the reliability of things people use in everyday life such as services, methods, products, and materials. Standards allow for interchangeability of parts which leads to mass production which saves money for all the companies in the related industry. Interchangeability in electrical components is achieved through standard protocols that assure compatibility and functionality for products. Another important factor that standards improve is consumer safety and public health. Listed below are the standards for software, sensors, microprocessor, and wireless communication.

1. The first standard related to our autonomous irrigation vehicle deals with software portion of our design, therefore, the collision-detection and pathfinding algorithm. It is called IEEE Std 1044-2009 which is a revision of an older standard called IEEE Std 1044-1993) - IEEE Standard Classification for Software Anomalies the description on the IEEE website states “This standard provides a uniform approach to the classification of software anomalies, regardless of when they originate or when they are encountered within the project, product, or system lifecycle. Classification data can be used for a variety of purposes, including defect causal analysis, project management, and software process improvement (e.g., to reduce the likelihood of defect insertion and/or increase the likelihood of early defect detection).” (IEEE).

2. The seconded standard that associated with the autonomous irrigation vehicle relates to the uses of sensors in our project. It is named 1554-2005 - IEEE Recommended Practice for Inertial Sensor Test Equipment, Instrumentation, Data Acquisition, and Analysis and is abstracted on the IEEE as “Test equipment, data acquisition equipment, instrumentation, test facilities, and data analysis techniques used in inertial sensor testing are described in this recommended practice.” (IEEE) Another standard the relates to sensors we must adhere to is the 2700-2014 - IEEE Standard for Sensor Performance Parameter Definitions the purpose of this standard according to the IEEE is “This standard presents a standard methodology for defining sensor performance parameters in order to ease system integration burden and accelerate time to market (TTM). This standard fulfills the need for a common methodology for specifying sensor performance that will ease the non-scalable integration challenges. This standard defines a minimum set of performance parameters, with required units, conditions, and distributions for each sensor. Note that these performance parameters shall be included with all other industry-accepted performance parameters.” (IEEE).

3. On the hardware side of the autonomous irrigation vehicle there is a standard related to microprocessor architecture called IEEE 1754-1994 - IEEE Standard for a 32-bit Microprocessor Architecture which is described on the IEEE website as “A 32-bit microprocessor architecture, available to a wide variety of manufacturers and users, is defined. The standard includes the definition of the instruction set, register model, data types, instruction op-codes, and coprocessor interface.” (IEEE).

4. For the communication with the wireless mesh network the autonomous irrigation vehicle must follow the IEEE Std 802.11-2016 for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications which is a revision IEEE Std 802.11-2012. The scope of the standard according to IEEE is "The scope of this standard is to define one medium access control (MAC) and several physical layer (PHY) specifications for wireless connectivity for fixed, portable, and moving stations (STAs) within a local area." (IEEE). The purpose of the standard as stated on the IEEE website is "The purpose of this standard is to provide wireless connectivity for fixed, portable, and moving stations within a local area. This standard also offers regulatory bodies a means of standardizing access to one or more frequency bands for the purpose of local area communication.". (IEEE) As the design of the autonomous irrigation vehicle develops the standards the relate will change and our team will add and remove the standards related to the project accordingly.

5. Realistic Design and Constraints

Many times over, there have also been restrictions and constraints of any given project that has been developed. The reasoning for any team to develop and realize their own project's design constraints is to realistically construct said project. As such, there have been many considerations when developing and building and kind of project in regards to economic, environmental, social, and other constraints that can impact the development of a group's project. During the research process, the group encountered many hurdles that have impacted the design of the autonomous irrigation vehicle. Of the many hurdles, the most prevalent of which determined the size and the autonomous vehicle's ability to traverse its surrounding terrain.

5.1 Economic and Time Constraints

As a group, we have set up personal deadlines for our group. By doing so, this allows our group to work efficiently and also account for any mishaps that may occur during development and research. Our group has also scheduled weekly meetings to confirm and reinforce a consistent rate of progression. With the group deciding to set a deadline before the designated deadline, the time constraints should be able to be handled with ease as well as minimize the consequences of random occurrences.

Aside from the group's personal timing constraints, there will also be timing constraints in regards to the irrigation of the designated areas. The suggested time to being irrigation of the designated area is projected to commence at 2:00 AM to 5:00 AM. The reasoning behind this suggested watering time window is so that the designated area does not receive more water than is required. When the irrigation has stopped, water has seeped into the designated area and will sustain the area itself. When the sun rises, water will begin to evaporate during the day which will help prevent a much unwanted complication with home lawns; lawn fungal diseases.

The group has selected Lake Nona, Florida as the autonomous irrigation vehicle's test location. As such, there are watering restrictions in place that will inevitably affect the proposed watering scheduling of the autonomous irrigation vehicle. Originally, a three day watering period was proposed alongside the watering time window of 2:00 AM to 5:00 AM. As the city of Lake Nona is located within orange county in the state of Florida, the proposed three days a week watering schedule has been reduced to a one day a week watering schedule. The new watering schedule of once a week is only applicable from the first Sunday of November until the second Sunday of March then switches to a schedule of twice a week during Daylight Savings Time from the second Sunday of March to the first Sunday of November. Below is a descriptive list and table of Orange county's watering restrictions courtesy of Orlando Utilities Commission. As per Orlando Utilities Commission's informative list and Table 2, watering scheduling will be affected during certain times of the year but it should have little impact in regards to efficient irrigation practices. On top of impacting irrigation operation, it will also end up having the consumer saving money due to the low frequency of lawn irrigation.

- Outdoor irrigation is limited to one day a week during Eastern Standard Time (from the first Sunday in November until the second Sunday in March) and two days a week during Daylight Saving Time (from the second Sunday in March until the first Sunday in November).
- Water only if necessary and not between 10:00 a.m. and 4:00 p.m.
- Water for no more than one hour per zone.

Time of year	**Homes with odd-numbered or no addresses**	**Homes with even-numbered addresses**	**Nonresidential properties**
Daylight Saving Time	Wednesday/ Saturday	Thursday/ Sunday	Tuesday/ Friday

Eastern Standard Time	Saturday	Sunday	Tuesday

Table 2: Orange County Florida Watering Restrictions. Reprinted with permission from OUC

Another major constraint is the amount of funding that was determined by the scope of the project. Funding was heavily affected by how many sensors were to be implemented, the design and fabrication of the autonomous irrigation vehicle's casing, and water consumption during water dispensing. The quantity of sensors required by the autonomous irrigation sensors have affected the overall cost of the vehicle by running an estimated price of $300. As for the fabrication of the autonomous irrigation vehicle's casing, it is also estimated to run for approximately $50 due to the allocation of needed materials. As for the water consumption rate, the group aims to dispense a relatively low amount of water when compared to the typical method of irrigation which would reduce a customer's water bill by a considerable as the rate of irrigation water runs at $1.655 per gallon for the first 19,000 gallons consumed, $3.094 per gallon for the next 11,000 gallons consumed, and then $5.79 per gallon after 30,000 gallons have been consumed for irrigation purposes.

5.2 Environmental, Social, and Political Constraints

In regards to the environmental constraints for the autonomous irrigation vehicle, there are not many to take into consideration. First, as the autonomous irrigation vehicle will be mainly be operating with water, careful consideration has been taken into account to protect the internal hardware. The autonomous irrigation vehicle's case will be housing the electronics and shield the hardware from any foreseeable water damage. Another environmental concern that has been considered is the proper disposal of the autonomous irrigation vehicle's battery. Users cannot simply throw away a battery as harmful chemicals and toxins can leak after extended use. There are no social constraints regarding the autonomous irrigation vehicle's operation. As for political constraints, there are virtually none as the autonomous irrigation vehicle will not be working in conjunction with human beings and solely with water and its given terrain.

5.3 Ethical, Health, and Safety Constraints

As the autonomous irrigation vehicle is a machine that will be conducting all operations outside with no human interaction, there are no health constraints whatsoever that impact the vehicle's design. Ethical constraints are also nonexistent due to the nature of the autonomous irrigation vehicle as it has no way of being able to out right offend any one person in particular. Safety constraints may come through the autonomous irrigation vehicle's tether as human beings can overlook and possibly trip over the tether itself. Although the possibility of a human being tripping over aforementioned tether of the autonomous irrigation vehicle has been considered, it is highly unlikely that it could ever occur as the hours of operation of the vehicle will be pre dawn thus cutting down the likelihood of such an event happening.

5.4 Manufacturability and Sustainability Constraints

The autonomous irrigation vehicle is heavily influenced by manufacturability constraints as it has affected the vehicle's overall design. A contribution to the manufacturability constraint is the feasibility of constructing the autonomous irrigation vehicle's outer casing as it will be housing the electronics that conduct its overall operation. Another manufacturability constraint is that the autonomous irrigation vehicle must be built as a light weight vehicle so as to not damage the consumer's terrain. Along with minimizing damage to the consumer's terrain, a light weight design also provides other benefits such as lower power draw due to its weight since the motors need to adjust their speed according to how much resistive force is presented. An additional manufacturability constraint to consider is the type of hose that will be tethered to the autonomous irrigation vehicle. A light weight hose is the preferable choice when considering the manufacturability constraint as it will allow the autonomous irrigation vehicle to travel with less resistive force while the vehicle is not dispensing water.

An additional constraint to consider is the sustainability of the autonomous irrigation vehicle. The autonomous irrigation vehicle operates on an onboard battery that supplies all of its power needs. As such, the autonomous irrigation vehicle's battery must be able to persist through all of the vehicle's operation including being able to communicate with the mesh network of sensors as it traverses the given terrain. Given that each zone will only be allotted a watering window of about one hour, it is imperative that the battery last for more than hour in order to effectively water each zone within the time limit. The goal here is to reach all of the designated destinations within the time and power limit that has been put on the autonomous irrigation vehicle.

6. Project Hardware Design

As time has passed, there were many revisions and changes that the autonomous irrigation vehicle had since its conception. Since conception, the overall design of the autonomous irrigation vehicle has changed dramatically. When the autonomous irrigation vehicle was first conceived, there were suggestions of having an onboard water tank for the vehicle itself to haul around. At the same time, there were also solar panels that were proposed to be built on the autonomous irrigation vehicle. Due to practicability, power, and financial restraints have caused the autonomous irrigation vehicle to drop the proposed water tank and solar panels all together.

As ideas and suggestions have dropped from the predesign of the autonomous irrigation vehicle, so did the overall hardware design. When the water tank was proposed, there were plans of incorporating water pressure and a water level sensor to monitor the water tank so that the autonomous irrigation vehicle was aware of its current water supply were dropped. With the new proposal of a tethered hose at the disposal of the autonomous irrigation vehicle, two design restraints were lifted which was the power requirement as well as the size. Although the sensors were scrapped for the water tank portion of the autonomous irrigation vehicle, the vehicle itself could also move more freely about any given terrain.

Another previous design element that was apart of the initial design but then dropped was the solar panel. The solar panel was meant to be the sustainable effort in regards of having the autonomous irrigation vehicle to be self-reliant and be untethered allowing free movement in its given terrain. With the recent revision of the autonomous irrigation vehicle, the proposed solar panel was dropped. In place of the solar panels the autonomous irrigation vehicle will have an onboard battery. In order for the autonomous irrigation vehicle's onboard battery to recharge, the vehicle itself would travel back to a home base.

6.2 PCB Design

The proposed design of the printed circuit board incorporates all of the sensors, servos, and electronics that the autonomous irrigation vehicle will be using, In order for the autonomous irrigation vehicle to be able to use all of the sensors, electronics, and servos, each of the pieces are carefully selected and strategically placed on the printed circuit board. By picking the smallest yet most effective sensors, electronics, and servos allowable, the printed circuit board has all of the necessary connections to maximize free space. With the maximization of space on the printed circuit board design, this will allow for future components to be easily added.

The electrical design starts with the printed circuit board (PCB). The PCB is responsible for managing the obstacle avoidance sensors. The PCB starts with a 9-volt direct current jack that is attached to the anode of a diode which is there to prevent any damage from occurring to the PCB if the power supply is connected incorrectly. The cathode of the

diode is attached to a capacitor that is in parallel with the input of the voltage regulator to control phase shifts. The voltage regulator puts out 1 amp at 5 volts. Another capacitor is attached in parallel with the output of the voltage regulator for stability. The 5 volts form the regulator are attached to three pins of the microcontroller required for normal operation. A 16 MHz crystal with two capacitors in series is connected to the proper pins to allow the microcontroller to keep track of time. The capacitors on the crystal are there to ensure proper oscillation. The PCB also has an in-system programming socket which is connected to the proper pins on the microcontroller. A resister is attached to the reset pin on the microcontroller and holds the pin high, so the program continuously runs. The PCB also has sockets for communication with the Raspberry Pi.

The socket has a voltage divider to prevent damage to the Raspberry Pi since its operating voltage is 3.3 volts and the PCB operates at 5 volts. The PCB also has a socket for a servo which is connected to power, ground and a pulse width modulation pin (PWM). The LIDAR also has a socket connected to power, ground, and two pins for I^2C communication protocol. The power and ground connection for the LIDAR has a capacitor in parallel to smooth out the power signal and guaranty accurate distance readings. The last socket on the PCB is for the waterproof ultra-sonic range finder which is connected to power ground and a PWM pin and a digital pin on the microcontroller.

6.2.1 PCB Assembly

The assembly of the printed circuit board is of the utmost importance in regards to the success of the autonomous irrigation vehicle. The layout of the printed circuit board also plays a role in how the autonomous irrigation vehicle is able to access and operate the sensors and other onboard electronics. With the layout of the printed circuit board properly mapped out, the component sizing also plays another role in terms of assembly. That is that the size of the selected components is either a difficult or a relatively easy step to carry out. Regardless of what circuit element or electronic component that is designed into the printed circuit board schematic, the assembly must be done with little to no error.

6.2.2 Manufacturer Selection

In order for the assembly of the printed circuit board to begin, there must first be an circuit board that has been printed out with the correct layout. Although the possibility of the group making their own printed circuit board is a reality, it would cut too deep into the scheduling of creating the autonomous irrigation vehicle. As such, there was a need to have the printed circuit board created within a reasonable timeframe for testing purposes. In order to cover the window of time that the group has to complete the autonomous irrigation vehicle, there was need of selection of manufacturers to create the proposed printed circuit board.

The first of manufacturers to be considered is a manufacturer that goes by the name of PCBWay. PCBWay, as a company, has been around for roughly a decade in the business. PCBWay is being considered for their quick turn over time in order to allow quick redesign if needed be. There could also be issues that could arise such as the printed circuit board may not have been printed properly so a quick turn around time could play a critical role.

The second candidate in consideration for the responsibility of ensuring the circuit board is printed correctly is JLCPCB. Just as PCBWay, JLCPCB has also been in the printed circuit board manufacturing industry for over a decade. JLCPCB also provides the speedy turn over that PCBWay offers but also manufactures multi-layer print circuit boards. If by chance the autonomous irrigation vehicle needed a multi-layered printed circuit board, JLCPCB would be able to provide a service in regards to this situation.

6.3 Microcontroller Design

Regarding the design of the microcontroller unit, it has to be a programming code that offers the best allocation of memory and communication. The allocation of the memory of the microcontroller unit is an essential part since it can help execute instructions in a much speedier manner. The lines of code that are responsible for the communication aspect of the microcontroller unit need to be quite effective to ensure the signals are sent to the correct device. Depending on the microcontroller that is chosen, it is programmed in an integrated development environment.

6.3.1 Assigning Input & Output Pins

There is a number of sensors that are attached to the microcontroller itself as it acts as the central hub of signals. As such, there is a limited number of pins that the microcontroller unit would have free so the essential pins must be outlined. In regards to assigning the input and output pins, there is an input pin from the battery management sensor, an input and output from the LIDAR, input from the ultrasonic sensor, and then an input and output for the computer processor. On top of the sensors and compute processor, there is an additional servo attached to the LIDAR to allow it to have a swiping view.

6.3.2 Connecting All Sensors

Connections of all the sensors that the autonomous irrigation vehicle uses is a simple feat. The only possible issue is where and how to place all of the sensors and which sensor communicates with which electronic device. As it currently stands, the LIDAR and ultrasonic sensors are connected to the microcontroller. There needs to be a way for the autonomous irrigation vehicle to return to the home base when its power supply is running low. In such a case, a battery monitoring sensor will be connected to the microcontroller so that he autonomous irrigation vehicle can remain aware of its current power supply level. there will be a battery monitoring sensor that will be connected to the either the microcontroller unit or to the computer processor unit.

6.3.3 Connecting to Motor Controller & Servos

As the perils lie ahead of the autonomous irrigation vehicle, the vehicle itself is able to detect a sudden obstacles in its chosen path. Thus in order for the vehicle to maneuver itself safely and securely, it has control over its wheels. Accomplishing the motor control of the autonomous irrigation vehicle has over its own set of wheels is imperative. As such, each wheel on the autonomous irrigation vehicle will be connected to its own servo. Each servo will be in charge of rotating its wheel. The only problem with the servos themselves is that they are activated the moment they receive current. With an uncontrollable electric force, the servos themselves could malfunction due to a current surge and have no way of stopping its actions. This is where a motor controller unit will be a convenient and practical solution to this problem.

The motor controller can adjust the speed of the servo itself by controlling how much power is sent to the servo. Not only can it control the power a specific servo is receiving, it can also make the servo or wheel spin in the opposite direction giving the wheel a backwards motion of rotation. With the wheel itself being able to rotate counter and counter clockwise, it gives the autonomous irrigation vehicle the ability to traverse its environment in a backwards motion. The final design of the autonomous irrigation vehicle has a single servo controlling the steering.

6.4 Drive Hardware

The drive hardware portion of the autonomous irrigation vehicle is also an indispensable design for the success of the vehicle. The autonomous irrigation vehicle’s base will be a Traxxas RC model. From there, there are servos and motor controllers in place so that the onboard computer processing unit can control the movements for autonomous activity. The servos be directly connected to the wheels to control speed and direction of wheel rotation. The motor controller will be connected to the servos to prevent surge damages and provide further control over servos.

Due miscommunication and vendor issues, an Arduino Uno Microcontroller was taken up to control the driving and steering of the autonomous irrigation vehicle. The Arduino Uno microcontroller is a powerful yet simple piece of electronic used by many people. As such, there is a multitude of information and a community behind the product itself. Thus with that in mind, the driving portion is controlled by the Arduino microcontroller which is then controlled by the computer processing unit.

An Arduino was implemented into the design to control the steering servo and brushed engine of the remote-control car base. To achieve this the radio frequency signals of the remote control had to be duplicated digitally. For the steering servo, this was accomplished by connecting the servo wires directly to the Arduino. Using the built-in servo libraries, testing of different angles until the proper angles for turning right and left at various intensities were discovered.
The engine receives its signals from an electronic speed controller (ESC). Finding what signals the ESC sends to the brushed engine required disconnecting the radio frequency receiver from the ESC and connecting the radio frequency receiver directly to the PWM pins on the Arduino. The remote control was then used to calibrate the ESC as per the manufacturer's manual. While the calibration was taking place the radio signals were read by the Arduino. The calibration was then replicated using PWM signals with the Arduino.

Once the values for neutral, forward and reverse had been calibrated into the ESC, the mechanics of the remote control had to also be duplicated. Loops were used to simulate the act of gradually pressing on the remote-control trigger. One loop was created for moving the autonomous vehicle and another for moving in reverse. The loop for making the autonomous vehicle move forward was simple and only required the value of neutral to increment to the desired speed gradually. To go in reverse, we needed to slow the autonomous vehicle down to neutral, incrementally reach the maximum reverse value, return up to the neutral value and then repeat back down to the reverse value to rotate the motors in the opposite direction. The reason the reverse loop worked in that manner was a safety feature the ESC had that prevented the user from instantly switching from forward to reverse to protect the transmission gears of the vehicle.

6.5.1 Servos Selection

The autonomous irrigation vehicle will be dealing with quite a large load of water. Since there is a considerable amount of water, weatherproofing these servos would be most beneficial for the autonomous irrigation vehicle. With the consideration of weatherproofing the servos, there is also an additional cost associated with such a feature. There are a number of ways to create a do-it-yourself weatherproofing of these servos or simply buying out right already weatherproofed servos.

6.5 Wireless Module Hardware

The autonomous irrigation vehicle needs to have a method of communication between itself and the mesh network of sensors. It has already been established that the IEEE 802.15 will be the main choice of communication options available. The hardware itself behaves as a transmitter and a receiver for the autonomous irrigation vehicle. The module itself is able to receive, analyze, and process data to and from specific devise and sensors. At the same time, it can transmit data back to the sensors or to the user for any current information.

6.5.1 Wireless Module Setup

Setting up the wireless module should not be a daunting task. It requires the module to be connected to a computer in some manner in order to program the wireless module. Once connected, the group used an integrated development environment to write the programming code for the wireless module. It was a relatively simple task of programming the transmitting and receiving operations of the wireless module. It is put into a sleep mode of some sort to conserve power during the autonomous irrigation vehicle's operation.

7.0 Ultrasonic Range Finder

The ultrasonic range finder was used as a backup obstacle avoidance sensor. The ultrasonic range finder works by sending out a specific high frequency sound wave signal through the transmitter and waiting for the signal to come back to the receiver. The transmitter acts like a speaker and the receiver acts like a microphone. The time taken for the sound wave to come back is recorded and since the speed of sound in air is known, the distance to the object that the sound wave bounced form can be measured. The reason ultrasonic range finders were used as back up obstacle avoidance sensors was because ultrasonic range finders have a widespread detection range that was easily affected by obstacles that were not necessarily in the path of the autonomous irrigation vehicle. The false reading caused the autonomous irrigation vehicle to take longer while navigating the lawn or yard. The combination of the LIDAR and ultrasonic range finders provided adequate obstacle avoidance allowing the autonomous irrigation vehicle to effortlessly traverse any size yard or lawn. Between two to four ultrasonic sensors were going to be placed on the autonomous irrigation vehicle. The team begin testing with one and deem it not necessary to add any more ultrasonic range finder to the finale design.

7.1 360 LIDAR

LIDAR is a short form for light detection and ranging which uses light in the form of a pulsed laser to accurately detect objects. The light detection and ranging played a crucial role in the autonomous irrigation vehicle's responsibility of traversing terrain and detecting objects. The light detection and ranging was placed on the autonomous irrigation vehicle to detect objects and avoid collision. When the autonomous irrigation vehicle received a signal from the mesh network to water a location or locations, it loaded a map with all currently known terrain data with coordinates of each sensor from the mesh network.

The shortest path to reach each destination midpoint was calculated using path finding algorithm which created a list of coordinates to reach each target location that need to be watered. When it received it coordinate lists, the autonomous irrigation vehicle stared from its home station. While the autonomous vehicle travel to the target midpoint, the LIDAR sends out pulse laser to identify if there are any objects on its way in the terrain, and waits for pulse to return. Once the pulse return back to LIDAR, based on the return pulse it accurately calculated how far the obstacles are from the autonomous irrigation vehicle and it updated the two-dimensional array map.

After it has been found the obstacle on its way to the target point, the autonomous irrigation vehicle recalculated the travel path again using the nearest neighbour algorithm. Figure 15 LIDAR flowchart shows the efficient process of finding the obstacle and updating the two-dimensional array map.

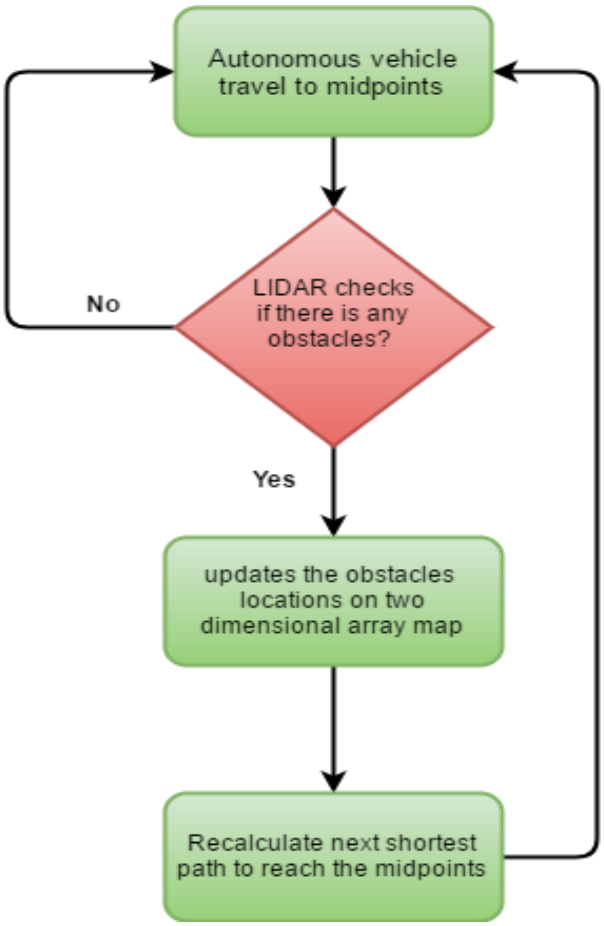


Figure 15: LIDAR Flowchart

There were several types of LIDAR available in markets. For the autonomous irrigation vehicle project, we have chosen RPLidar A1M8 model because it has the perfect specs which will help us to accomplish our goal such as efficiently update the data in two-dimensional array map. The autonomous irrigation vehicle will be used in the outdoor and this model works perfectly for outdoor environment. Moreover, the Table 3 shows the RPLidar A1M8 model specifications such as distance range, angular field of view, laser wavelength, motor power supply, and scanning rate. Also, Figure 16 shows the RPLidar A1M8 model picture.

Even Though the autonomous irrigation vehicle needed to scan 180 degree, this light detection and ranging scans up to 360 degree from the autonomous vehicle surrounding. Also, it has maximum laser wavelength of 795 nanometer. Furthermore, The RPLidar will be taking in data from the vehicle's surrounding area which will be used in connection with the vehicle's computer vision algorithms that will allow the vehicle to move autonomously. In the flowing document the LIDAR is referenced as computer vision obstacle detection system. After meeting with our sponsor during senior design 2 class, we decided not to implement the 360 degree LIDAR in our final project design instead we used 180 degree LIDAR.

Figure 16: RPLidar A1M8 Model. Reprinted with Permission from RobotShop

	Minimum	Maximum
Distance range	0.2 m	6 m
Field of view	0°	360°
Laser wavelength	775 nm	795 nm
Motor power supply	5 v	10 v
Sample frequency	n/a	2010 Hz
Scan rate	1 Hz	10 Hz
Price	$199	
Vender	RobotShop	

Table 3: LIDAR - RPLidar A1M8 Model Specs

7.1.1 Weather Proofing LIDAR

Originally the team had considered a 360 LIDAR as the main obstacle avoidance sensor but has since reconsidered because of the fact that all the electrical components must be waterproof or at the minimum weather proof. While researching how to waterproof/weatherproof a LIDAR we came across two different options.

The first option was to manufacture a custom case out of an infrared transparent material which allows the specific wavelength that the LIDAR uses to pass through the material with minimum reflection so that information readings from the LIDAR are accurate. An example of this material is a clear polycarbonate plastic that is easily molded to fit around the LIDAR. Another benefit of using a clear polycarbonate plastic is how inexpensive it is and that ninety percent of the light beam used by the LIDAR passes through it. While researching our team found one way to improve the infrared transparent case which was to add a surface coating to the case such as an anti-reflective coating. The anti-reflective coating would help reduce the amount of signal that is reflected as its passes through the infrared transparent case. Even when using an infrared transparent material there are few things that must be considered.

1. The case would have to be properly sealed form water and rain.

2. The thickness of the case would affect how much light form the LIDAR would be allowed to pass though, the thicker the material the more light would be blocked form the LIDAR. The thickness could also bend the light beam from the LIDAR causing the readings to not be accurate.

3. Any scratches the infrared transparent case may receive would cause errors in the measurements.

4. Any dust that would collect on the infrared transparent case over time would also disturb the signals form the LIDAR.

5. The final reason this method of water proofing the LIDAR was not chosen was because even with the infrared transparent material the emitter and transceiver of the LIDAR would have to be as perpendicular as possible with the surface of the material in order to get accurate reading form the LIDAR. The 360 LIDAR we had previously chosen had a cover over the transmitter and emitter that made this method difficult without having to take apart the LIDAR and possibly damage the functionally in the process. Since we can't afford to attempt this type of modification we had to continue researching how to waterproof/weather proof the 360 LIDAR.

The second method of waterproofing the LIDAR that our team came across while doing research was buying a waterproof case such as a dry box. The waterproof case would of course have to be modified for the LIDAR. The modification would require the use to a Dremel tool to cut out holes for the LIDAR's emitter and receiver. The emitter and receiver would be sticking out of the case. Using some silicone rubber compound and gaskets the area around the emitter and receiver would be sealed off so that no water would be able to leak in through the hole made by the Dremel tool.

Another benefit to this method was that by purchasing a bigger water proof case our team could water proof other components such as the custom printed circuit board, Raspberry

PI, battery, power distribution board, ultrasonic range finders, radio frequency module and any other components that may be added during the design process. Again, this waterproofing method would not be reasonable choice for a 360 LIDAR because the emitter and receiver would have to sticking out of the waterproof case. As discussed in the previous waterproofing method, our team does not have the luxury of trying to modify the 360 LIDAR and possibly damage the expensive sensor. Since both methods researched where not suited for a 360 LIDAR we researched a solution using an unidirectional LIDAR discussed in section LIDAR Setup. The finale design was a proof of concept and was operated indoors and the sprinkler head was replaced by a flashing blue LED. No waterproofing was implemented in the final design.

7.2 Weather Proofing Servo

Now that we have added a servo for the LIDAR it too had to be weather/water proof. While our team did some reach into weather/water proofing we came across a popular method used by remote control vehicle enthusiasts. The first step requires disassembling the servo and adding lubrication to the base of the spline which sticks out of the case of the servo. For lubrication, gear grease is used so that when water tries to seep into the servo through the hole of the spline it is repelled by the grease and does not reach the water sensitive electronics of the servo. The second step is to seal all the seams around the servo case.

This can be accomplished by using silicon sealant, liquid electrical tape, or air-dry rubber spray to seal all the seams around the servo. Tape must be used to make sure the spline portion of the servo does not get covered by any of the sealants as this part must be allowed to easily move. Water proof servos are manufactured and sold, the only problem is they cost on average about thirty dollars more for a servo with the same specifications. The waterproofing method discussed above was not tested on older servos owned by one of our team members and was not proven to not work properly. A waterproof servo was purchased for the 180-degree LIDAR setup on the autonomous irrigation vehicle. In the final design a non weather proof servo was used because it had less electromagnetic field interference than the weather proof servo. The electromagnetic field interference was causing false readings for the ultrasonic range finder.

7.3 Weather Proofing Ultrasonic Range Finders

The team had learned a lesson form the LIDAR and instead of researching and picking out which ultrasonic range finders we would be using on the autonomous irrigation vehicle we preformed research on how to water proof them first. As soon as we started the research we found that weatherproof ones were available on the market. The entire ultrasonic range finder is not water proof. What they do is separate the printed circuit

board part of the ultrasonic range finder form the probe which serves as the transmitter and receiver. The transmitter/receiver probe is weather proof and has a long cable, so the printed circuit board could be placed with the other electronics in safe weather proof box. While the weatherproof ultrasonic range finders are about fourteen dollars more the team has decided that this will be the best option for the autonomous irrigation vehicle. In the final design a non weather proof ultrasonic range finder was used because it proved to be more accurate than the weather proof one specially with the electromagnetic field interference form the servos.

7.4 LIDAR Setup

While performing research on non-360 LIDARs we came across the Garmin LIDAR-LITE v3. The LIDAR-LITE v3 is a high-performance optical distance measurement sensor which is perfect for autonomous vehicle applications such as our autonomous irrigation vehicle. Some of the other appealing features are how compact it is with a height of 40 millimeters, width of 48 millimeters, and a depth of 20 millimeters which is 1.6 inches in height, 1.9 inches in width, and 0.8 inches in depth. LIDAR-LITE v3 is also light weight coming in at only 22 grams or 0.7 ounces. The LIDAR-LITE v3 also has low power consumption only consuming 130 milliamps operating at 5 volts direct current with an ideal current of 105 milliamps. The LIDAR-LITE v3 also only has a single digital signal processing chip allowing for easy communication with our custom printed circuit board though I-squared-C or pulse width modulation. The easy communication allows our team to adjust the accuracy, operating range, and measurement time of the LIDAR to suit our needs perfectly. This LIDAR is also suitable for both waterproofing methods discussed in the section Weather Proofing LIDAR.

The only issue with the LIDAR-LITE v3 is that it could only range find in one direction. To solve this issue, we came across a solution which requires a few extra parts to transform the LIDAR-LITE v3 into a 180-degree LIDAR which suits our needs perfectly. Since the autonomous irrigation vehicle will now constantly be tethered to the hose we will only need 180-degree obstacle detection maybe even less. The setup requires a servo motor attached to a bracket so that it can be mounted on the autonomous irrigation vehicle. The LIDAR also requires its own bracket so that it can be mounted on the servo motor. The servo motor and the LIDAR will be programed with the microcontroller on the custom printed circuit board in order to transform the LIDAR into a 180 degree LIDAR. A picture of a test setup the team performed is pictured on Figure 18.

Figure 17: Test setup of 180º LIDAR performed by the team

7.4.1 LIDAR Code

The programming of the LIDAR sensor began by setting up the serial communication between our printed circuit board and the LIDAR using the I-squared-C communication protocol. I-squared-C is typically used for short communication between interconnected circuit peripherals such as the LIDAR in our autonomous irrigation vehicle. Similar to a universal asynchronous receiver-transmitter (UART) I-squared-C only requires two wires for communication. The communication only requires two signals. One is a clock signal called SCL and the other is a data signal called SDA. It is also supports multiple masters for example, microcontrollers and processors which in our autonomous irrigation vehicle are the printed circuit board and Raspberry PI. In addition, it also supports multiple slaves which would be the LIDAR, ultrasonic range finders, and servos in our autonomous irrigation vehicle. Once the I-squared-C was properly setup the microcontroller will began communication with the LIDAR printed circuit board. The flow chart for programing can be seen in Figure 19.

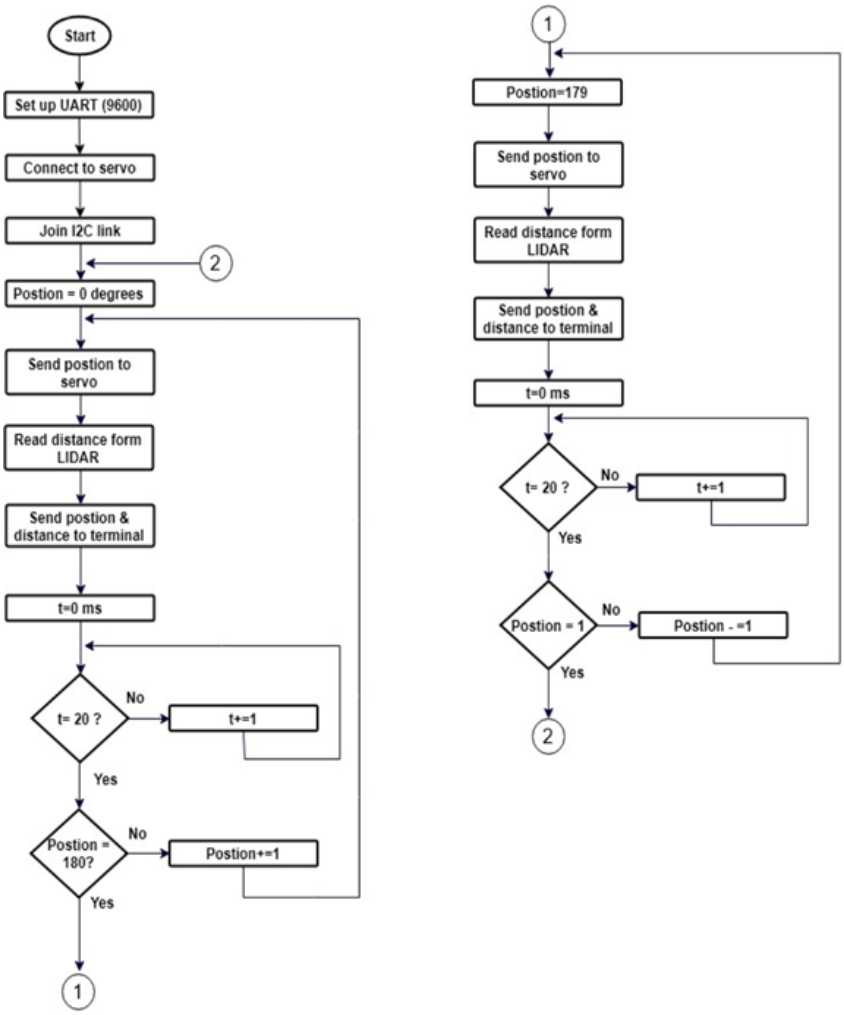


Figure 18: LIDAR Programing Flowchart

The microcontroller lets the LIDAR's printed circuit board know what values to store in the register pointers. The program includes a delay to allow sufficient time for the pointer registers to load. For the first round of testing we included a function in the program that prints the results of the reading from the LIDAR in degrees for the position of the servo and in centimeters for the distance to the object in ASCII characters to a terminal on a computer. Later in development this function was changed to serial communication between the printed circuit board and the Raspberry PI.

After the initial hardware configuration and function definitions the main body of the program begins. The program was composed of two "for" loops. The first loop is responsible for moving the servo from 0 to 180 degrees in increments of one degree and taking measurement readings at every degree. As it takes its readings it will also send the position and distance data to a terminal.

The next loop was responsible for moving the servo from 180 to 0 degrees again in one-degree increments as it takes readings from the LIDAR at every angular position and sends the positions and distance information to the terminal or Raspberry PI. The

program continues to switch back forth between the two “for” loops indefinitely. The programs turned a unidirectional LIDAR into a 180-degree LIDAR allowing the team to get the function needed for obstacle avoidance and saved money over the 360 LIDAR that was originally being considered as the main sensor for obstacle avoidance in the autonomous irrigation vehicle.

7.5 Ultrasonic Range Finder Code

The programing for the ultrasonic range finder began by setting the echo pin as an input and setting the trigger pin as an output. The echo pin is then set to high. The main program was a loop that set the trigger pin to low and set a delay for two microseconds. Next the trigger pin was set to high for ten microseconds. Then the trigger pin is back to low. This setting of the trigger pin to low and high was creating the sound pulse form the ultrasonic range finder. The next portion of the code records how long the sound pulse took to return to the high echo pin and calculated the distance.

The final part of the code created a delay of 50 milliseconds before looping back around. For testing purposes, the next part of the code printed out the distance of the object in centimeters to a terminal on a computer. After testing this portion of the code was changed to serial communication between the printed circuit board and the Raspberry PI. This code could have been easily adjusted to multiple ultrasonic range finders by activating their proper trigger and echo pins in the same fashion as the above program explanation. The code was never adjusted because only one ultrasonic range finder as used in the finale design. A flow chart of this program can be seen in Figure 20.

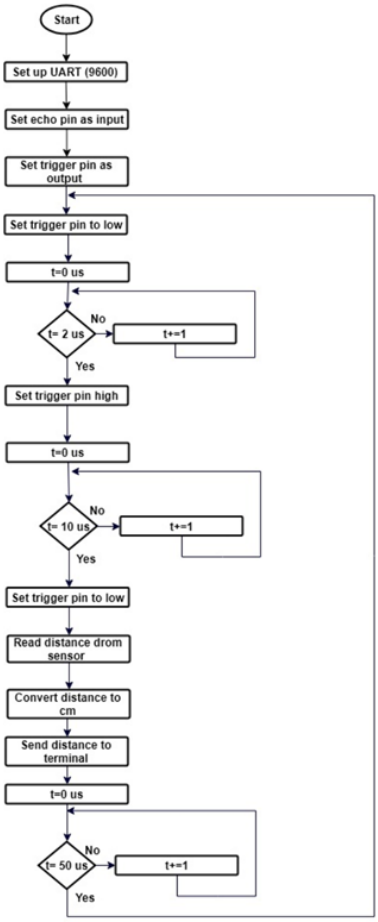


Figure 19: Ultrasonic Range Finder Programing Flowchart

7.6 Hardware Programing Libraries

The Arduino library wire.h was used in the programing of the LIDAR. The first function used is named Wire.begin and is responsible for joining the I-squared-C bus as a master or slave. In our case the microcontroller is joining as a master. The second function used is called Wire.beginTransmission this function begins a transmission to a I-squared-C slave device with the address typed in the function. In our case it was the LIDAR. The third function is named Wire.write and is responsible for writing data between the slave and master of the I-squared-C protocol. The fourth ends the transmission started by Wire.beginTransmission. The fifth is called Wire.reqestFrom is a function used by the master to get information form a slave device. The sixth is called Wire.read and is used after Wire.reqestFrom and reads the information sent from the slave to the master.

The second library used is called Servo.h and was used to program the servo attached to the LIDAR. The first function used from this library is called myservo.attach and is responsible for attaching the servo variable to a specific pin. The second function used is named myservo.write and is responsible for writing bytes to the servo motor. The

bytes writing to the servo motor control in our program set the angle of the shaft in degrees.

Both the LIDAR and Ultrasonic sensors used the Arduino.h library. The first function they both used is Serial.begin which sets the baud rate for serial communication. The second is pinMode which configures the pin as an input or an output. The third used is digitalWrite which, once the pins are configured, it allows to set them to high or low. The fourth is delayMicroseconds which delays the program for the value entered in microseconds. The fifth reads the state of a pin whether it's high or low and is called pulseIn. The last is Serial.print which prints to a terminal in our case.

7.7 Drive System

The Traxxas summit 1/10th Scale was going to be used as the base for the autonomous irrigation vehicle and was chosen by the mechanical engineering students of the Guard Dog Valves team. It features a four-wheel drive system with wheels that have a large radius and off-road tread for traction. These features are important because the autonomous irrigation vehicle needed to effortlessly navigate a customer's lawn or yard filled with grass, dirt, weeds, mulch, and rocks. The Traxxas summit 1/10th Scale is meant to be driven off-road so the tread on the wheels will allow the autonomous irrigation vehicle to maneuver any yard or lawn in any type of condition.

Since the wheels are designed for off-road driving the tread on them will not allow the autonomous irrigation vehicle to slip on the lawn or yards even when wet. They will also support the autonomous irrigation vehicle when it experiences steep inclines allowing it to traverse up hills and down slopes. The prototype testing will be performed in Florida and elevation change is highly unlikely but if marketed in different states or countries the autonomous irrigation vehicle will be ready.

The mechanical engineering students had performed research on other drive systems that had tank tracks instead of wheels and decided to go with wheels because the tank tracks can be unforgiving on loose landscape. For consumer satisfaction Guard Dog Valves would not want the autonomous irrigation vehicle to destroy or worsen any loose patches in the yard or lawn. The Traxxas summit 1/10th Scale base is approximately two feet by two feet and is the appropriate size for tackling the task of watering a 20x30plot of grass. The compact size of two feet by two feet is also favorable from a marketing stand point because the autonomous irrigation vehicle can be easily stored in a shed or garage without taking up valuable storage real estate.

A 3D draft of the layers of the shell that will go on top of the Traxxas summit 1/10th Scale can be seen in Figure 22 with the exact dimensions and was designed by one of the mechanical engineering students of the Guard Dog Valves team. An overall system

diagram of the mechanical components and a few electrical components can be seen if Figure 23 that the mechanical engineering students of the Guard Dog Valves team made.

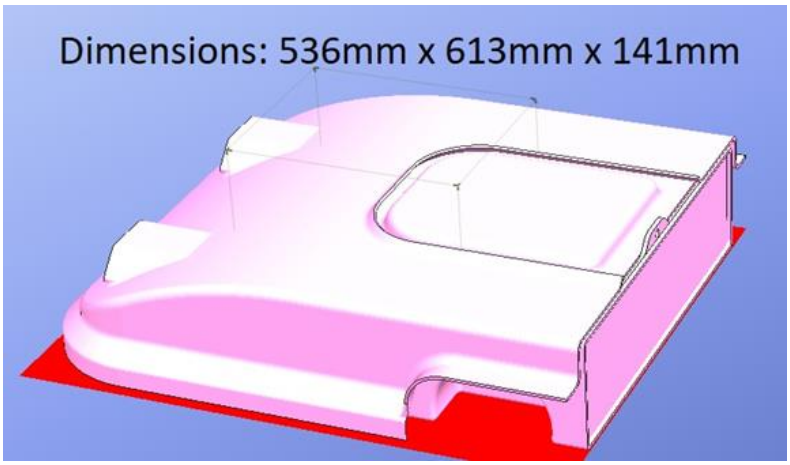


Figure 20: Prototype Dimensions for Autonomous Irrigation Vehicle

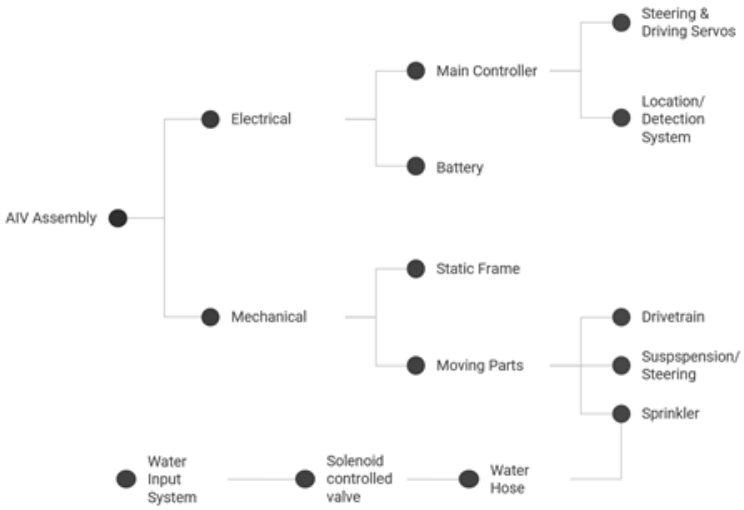


Figure 21: Mechanical System Design Diagram

Some other benefits of The Traxxas summit 1/10th Scale as a base model is a durable suspension deigned specially for off-roading. The Traxxas summit 1/10th Scale was also designed with the remote-control car enthusiasts in mind so all the parts are modular which means performing modifications should be pretty easy. This feature is extremely desirable for the Guard Dog Valves team as electrical and mechanical modifications must be made to build the autonomous irrigation vehicle. The Traxxas summit 1/10th Scale is also relatively low weight coming in at 12.7 pounds which almost any home owner would be able to lift if required. The Traxxas summit 1/10th Scale also comes with a reliable rechargeable battery that has been designed to be outside and the battery and all of the other electrical components such as the motor and transmission and are all water proof.

This is a great feature for the autonomous irrigation vehicle as it will spend most of its time outside and will have a sprinkler head attached that will be dispensing water throughout the yard or lawn. In the final design a 1/16th Scale Traxxas summit was used instead to the Traxxas summit 1/10th Scale due to miscommunication between mechanical engineering students of the Guard Dog Valves team. The Traxxas summit 1/16th has all the same features except they are all scaled down and less powerful.

8. Software Initial Design

The autonomous vehicle started at its charging home station. When it received a signal from the mesh network to water a location or locations, it loaded a map with all currently known terrain data with coordinates of each sensor node from the mesh network that required watering in a hamiltonian graph. The shortest path to reach each midpoint exactly once was calculated creating a list of coordinates to reach each target node that needed to be watered and factoring in known obstacles in the area, when the list of nodes were empty the vehicle returned to its home station. The vehicle was going to attached to a retractable hose tether from its home station to provide water. But the tether hose was not implemented in the final project. From the mesh network it received sensor data such as which sensors are more dry and where its located. While the autonomous irrigation vehicle was traveling to each target midpoints, it used computer vision obstacle detection system to detect obstacles such as ditches, steep slopes, rocks, trees, or anything that may impede the path or damage the autonomous vehicle. When a new obstacles are detected, it updated its map of obstacles and target nodes and recalculated a new path accordingly. It also constantly monitored for whenever the vehicle entered the range of the next target node on its path. When it did, it turned on its on board water dispensing system and circles once around the node watering the area. Once it completed its circle around the midpoints it turned off its watering system and headed to the next midpoint on its path list. Since this project was interdisciplinary, for Electrical and Computer Engineering(ECE) students purpose instead of using sprinkler head, the ECE group used LED light to indicate when the vehicle travel to the target midpoint and when it watered the midpoint. The Mechanical students will implement the sprinkler in the future semester. The overall software design flowchart shown in Figure 24.

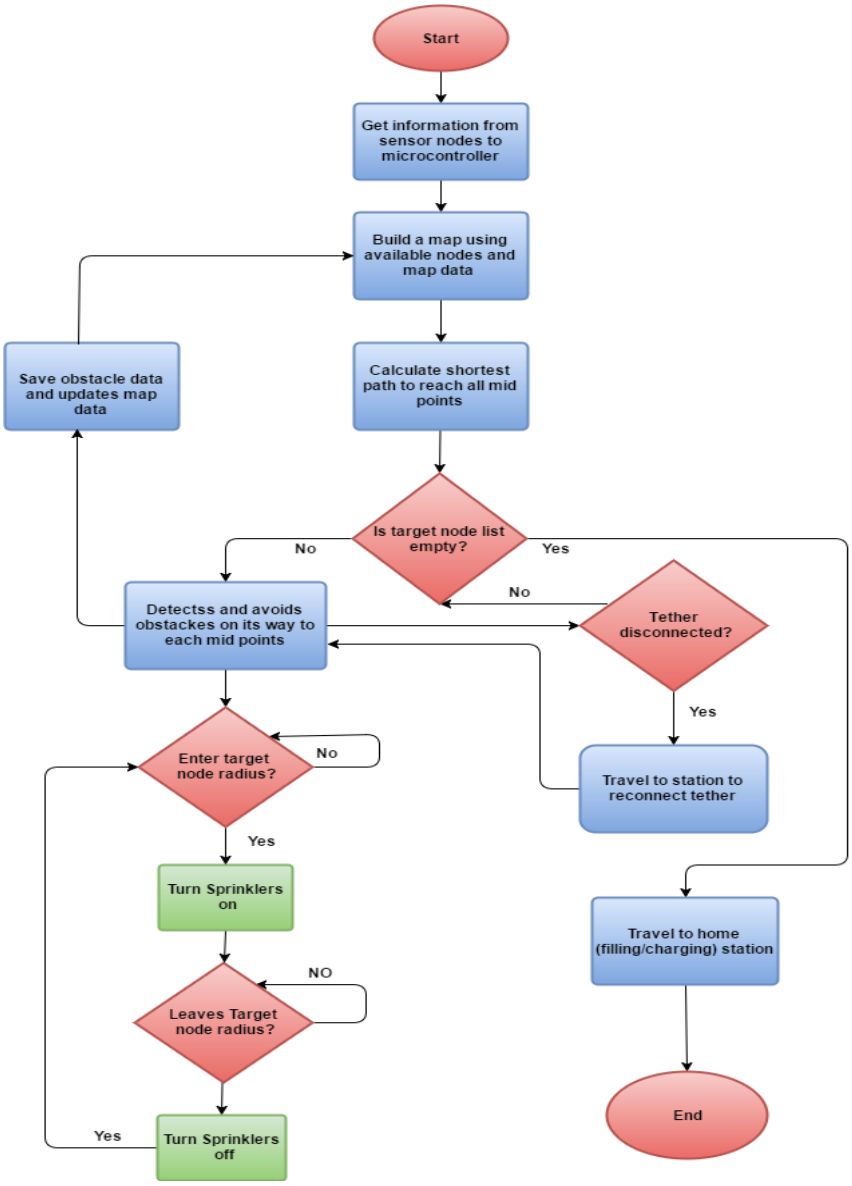


Figure 22: Software Flowchart

This comes across as an example of the classical “Traveling Salesperson Problem” (TSP), which asks to find the shortest possible route to visit each “city” given a list of “cities” and the distance between each pair of “cities. This can be approached using a combination of an uniform cost search (UCS) and a greedy algorithm such as repetitive nearest neighbor algorithm (RNNA) or an A Star search algorithm, a Markov Decision Process (MDP) could also work well for this too. With either A Star search or Markov Decision Process, heuristics and reward functions could be based on a water dispersion rate, an energy consumption rate, and time or distance costs for it in the future the vehicle becomes cordless with a water tank, but for the time being would start out with only time and distance while it uses the tether, with the tether RNNA may be sufficient enough.

Repetitive nearest neighbor algorithm is basically when you run a nearest neighbor algorithm using each node as a starting node. The nearest neighbor algorithm started

from a node and chose the nearest unvisited node from the to move to next and repeats the process until all nodes have been visited. This generated at least as many paths as there are nodes, more if at any point there are multiple neighbors who are equally close to each other and the path needs to split. From all the paths generated, the shortest distance path was chosen.

8.1 Software Goal

The goal of the autonomous irrigation vehicle project was to provide the end user with an irrigation system that does not over water the grass and reduce the human interaction with the system. Having this autonomous irrigation vehicle, the customer did not have to worry about the changing the sprinkler control manually and turned off the sprinkler controls on rainy day. To obtain the autonomous irrigation vehicle's main goal, it effectively and efficiently navigated to the destination location to disperse the accurate amount of water. The autonomous irrigation vehicle received the sensor data from mesh network and when the data was received the software calculated the nearest midpoint using nearest neighbor algorithm to travel to the target midpoints. The software detected the obstacles and avoided collision and the software updated the two dimensional matrix array every time it traveled to the midpoints. The software received the sensor dryness reading form mesh network.

The main software goal:

- Effectively irrigated a 30x20 ft plot of grass
- Effortlessly avoided obstacles
- Prevented collisions
- Dispensed accurate amount of water
- Updated the two dimensional map dynamically
- Reduced water consumption
- Effectively communicated with mesh network of sensors
- Controlled rate of water flow

8.2 Class Diagram

This section explains the class diagram of the autonomous irrigation vehicle. The class diagram shows each of the classes in the autonomous irrigation vehicles. Each class was divided into three components such as class name, attributes and operation. The class ID was unique to each class, and the attributes represented in +public, -private and #protected. The operation component has shown the specific operation done by that class. The class diagram for the autonomous vehicle is shown in the Figure 25

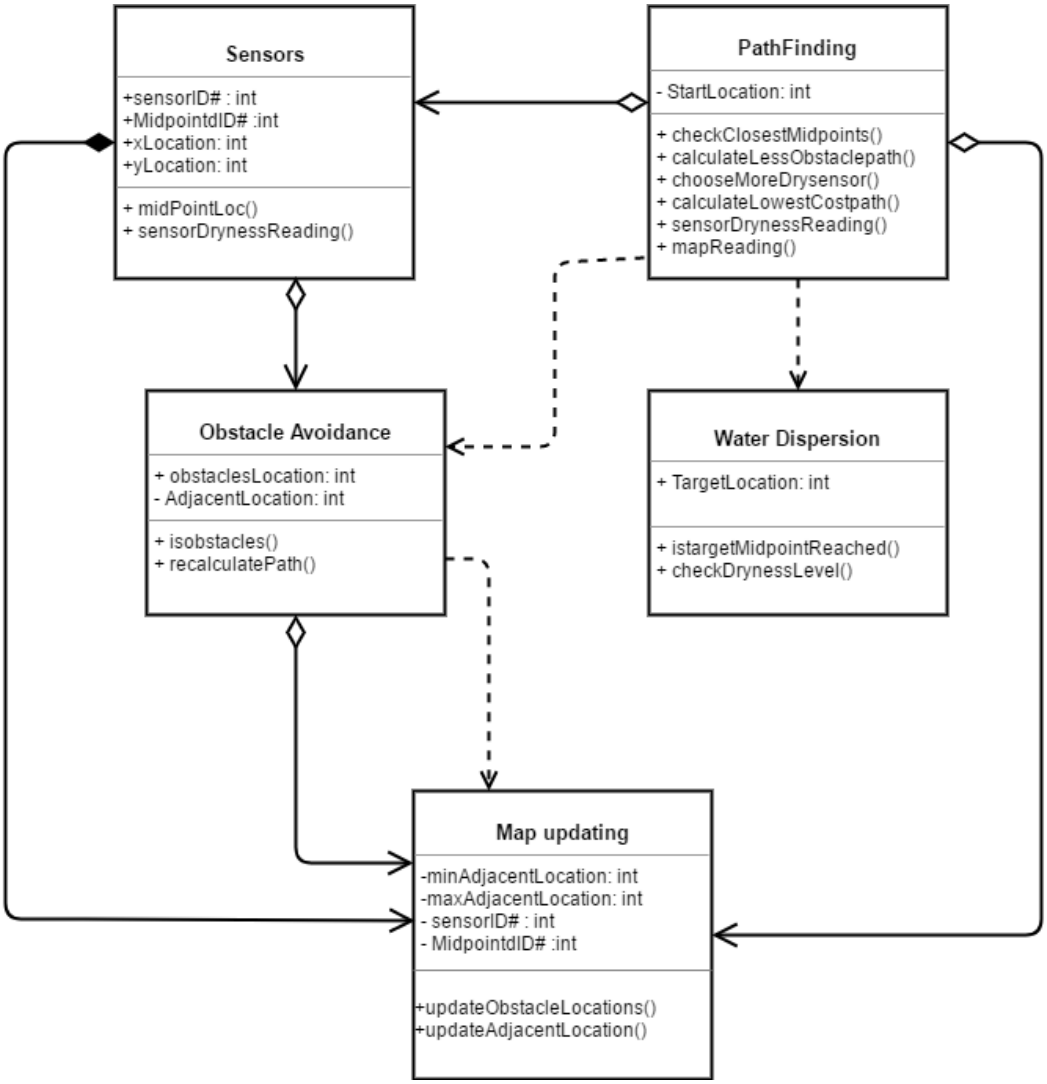


Figure 23: Class Diagram for the Autonomous Irrigation Vehicle

Also, the class diagram shows all classes and the relationship between the classes. The association relationship was denoted by single line that shows what are the classes need to communicate each other and aggregation relationship which denoted by a empty diamond symbol. Besides, composition relationship, which is represented in dark black diamond, shows the strong relationship between each of the classes and the dependency relationship is shown in dotted line.

The first class shown in the class diagram is a sensors class. It has four attributes such as sensor ID number and the midpoint ID number, x and y location of the indices. All four of the attributes are defined as public so it was accessed in other classes as well. After received the sensor data from the mesh network, the x and y locations of each of the sensors in the network was represented as indices for the two-dimensional array map and marked as the value one million counting up by the sensor's ID value to easily identify it. In addition 1999999 was used as a flag value for sensor, so the sensor values we can go up to 1999999 which was the upper limit for the sensors value. This value can be increased in the future development if needed. So these sensors values will be stored

as a integers in the program and if needed we can subtract the sensor constant value to get the each sensor's actual ID number. Having this sensor value marked it easier to calculate the midpoint between the sensors and to determine when to activate and deactivate the sprinkler system in the autonomous irrigation vehicle.

Furthermore, the class had a unique midpoint ID number and it had x and y locations of midpoints which was stored in the two dimensional array map. In the map two million represented the midpoint locations. At the indices on the two dimensional array map for each midpoint location will store the value two million plus the midpoint unique ID number. 2999999 was used as a flag point for the midpoints. If the unique midpoint id was needed, it could have been calculated by subtracting two million from the value stored at the indices in the two dimensional array map. In addition, the class has two operation functions which was midpointLocation operation and DrynessCheck operation. The midpointLocation operation keep tracked of how many midpoints were in between the sensor and its locations. When the autonomous irrigation vehicle received the sensor data from the mesh network, it used the path finding algorithm to calculated the shortest path to reach the target midpoint. The autonomous irrigation vehicle traveled to the midpoints between water sensors rather than to the water sensors itself, because by having the sensors read the soil moisture at the edges of the water dispenser's range we would get a more accurate reading of how dry the soil was to provide a accurate amount of water for it. If the sensor were to be in the middle of the path in which the autonomous irrigation vehicle is traveling, then the sensors would give inaccurate readings as soil would be more damp in the middle and the edges may still be dryer than they should be. The DrynessCheck operation was used to keep track of the sensor dryness reading. Once the sensor received the maximum amount of water it sent signal back to the network to stop water that midpoint. The boolean value will be used in DrynessCheck operation to indicate the sensor reading, 0 means the sensor was dry it needed water whereas 1 means the sensor was no longer dry. Moreover, the sensor class has aggregation relationship with path learning class and obstacle avoidance class. Also, sensor class has composition relationship with map plotting class. Composition relationship indicate the strong relationship between classes. Map plotting class were not function without the sensor class because the sensor class contained all sensor and midpoints ID number to plot the map.

The other specific class was a path finding class which has a FindStart attribute and its marked as private to that class. To achieve the goal of the autonomous irritation vehicle, it should effectively and efficiently navigate to the target midpoints in the shortest time in terms of distance and obstacles. This was crucial for the autonomous irrigation vehicle to determine where to go and how to get there in order to dispense water to the destination midpoints. For the autonomous irrigation vehicle, the two dimensional array map stored the start location of the vehicle. There were several operations were defined in the pathfinding class. When the autonomous irrigation vehicle received the sensor data from a mesh to water a target midpoints, the program loaded a two dimensional Locations object array map with all the known sensor location and midpoints location. Choosing which location to travel to dispense water was a major decision of autonomous irrigation vehicle so using the algorithm the NearestDryDestination function will calculate closest midpoint to reach each target midpoints locations between dry sensors to supply water to

the sensors. While the closest midpoint was selecting the target midpoint path, the calculates obstacle path function also checks which path has less obstacles. Choosing less obstacle path helped the autonomous irrigation vehicle to prevent from collision.

In addition, if two midpoints were in same distance from the start location, chooseMoreDrySensor function was used to calculate which sensor was driest in order to go to first. Also calculateLowestCostpath function was used to calculated the shortest path in a possible less distance from the start location and mapReading function would be used to get the sensor location and midpoints location from the two dimensional array. The DrynessCheck operation was used to keep track of the sensor dryness reading. Once the sensor received the maximum amount of water it sent signal back to the network to stop water that midpoint. The boolean value was used in DrynessCheck operation to indicate the sensor reading, zero means the sensor was dry it needed water whereas one means the sensor was no longer dry. To achieve the autonomous irrigation vehicle goal, the pathfinding class associated with several other classes such as it had a aggregation relationship with sensor class and map updating class. Because the pathfinding class needed the sensors and midpoints location data from the sensor class to be able to calculate the shortest path to reach the destination. And the map updating class needed the data form the pathfinding class to update the sensors and midpoints locations on the two dimensional array map. The obstacle avoidance class and the water dispersion class had a dependency relationship with pathfinding class because the functionality of the obstacle avoidance dependent on the pathfinding class as well as the water dispersion class functionality depended on the pathfinding class. Because the autonomous vehicle had to find the shortest path first to travel to the midpoint to dispense water so without the proper function of the pathfinding operation the water dispersion class could not trigger its function.

Another important class was the obstacle avoidance class. To accomplish the autonomous irrigation vehicle target, it effortlessly detected the obstacle to prevent collisions. Light detection and ranging used in the autonomous irrigation vehicle to detect the objects. It detected objects such as rock, trees, ditches or any other objects that may damage the autonomous vehicle. In class diagram obstacle avoidance class has two attributes such as findObstacleLoc and FindDir. The obstacle location was defined as public because the other classes should be able to access the obstacle Locations from that class. And the FindDir declared as private because it was only used in the obstacle avoidance class. There would be two operations performed in the obstacle avoidance class. When the data was transmitted form mesh network, the list of midpoints that need to water would be stored in the list. The path finding algorithm which was the nearest neighbor algorithm was used to calculate the shortest path to reach the target midpoint to disperse accurate amount of water.

While the autonomous irrigation vehicle was on its way to the target midpoint, the isobstacle function was triggered and the Light detection and ranging detected if there was any objects on its way, when it found any obstacle it sent signal to that function. Then function updated the obstacle location in two dimensional Locations object array to keep track of the obstacle location. In the map number nine was used as flag to represent

the obstacle location. Once it sent signal to the function regarding the obstacle, it called the recalculatepath operation to recalculate next shortest path to reach the target point. If the autonomous vehicle did not find any obstacle on its way it sent signal to the function to indicate there is no obstacle found and continue on the same path to reach the target midpoint. The obstacle avoidance class was associated with many other classes such as it had aggregation relationship with map updating and sensors, dependency relationship with pathfinding and map updating. It had dependency relationship with pathfinding because obstacle avoidance was dependent on the all functions of the pathfinding, first the autonomous vehicle should be able to travel to target midpoint to detect any obstacles.

Water dispersion was another important class. It has the TagetLocation attribute and it declared as public since other classes had to access that variable. The water dispersion class also had two operations such as isGoal and DrynessCheck. When the autonomous irrigation vehicle travelled to the midpoint the isGoal function checked if the vehicle reached its destination, if it reached its destination it sent signal to that function and it turned on the sprinkler to start watering the place. While the sprinkler was watering the specific area, the DrynessCheck operation was keeps checking if the sensor was reached its maximum water level. Once it reached it max level it sent signal to function to turned off the sprinkler and move on to the next target point. This function helped the autonomous irritation vehicle to not to over water the area. The final class was the map updating which was a important part to keep track of the obstacles locations, sensors locations and midpoints locations. Map updating class had four attributes such as sensor ID number, midpoints ID numbers, minimum adjacent location and maximum adjacent locations. All these attributes were declared as private variables and all these sensor, midpoints and adjacent values were marked in the two dimensional array of Locations objects under hazard level.

The value nine was used to mark the obstacle location on the map and every index adjacent to every obstacle, that did not have the value nine in it, is updated with the number of obstacles that they are adjacent, with the exception of values greater or equal to one million for the sensors. The maximum number of obstacles that can be adjacent to a location was eight while the minimum number of obstacles that can be adjacent to a location was zero. This class had two operations such as isObstacle and updatemap. When the vehicle travel to the destination point, it checked for obstacles on its way if it is to find any obstacles it updated location on map using the isObstacle and updatemap operations. Also, the map updating class is associated with several other classes. It had dependency relationship with obstacle avoidance class and aggregation relation with pathfinding and obstacle avoidance class. As well as It had composition relationship with sensor class because without sensor class the map updating would not function well to attain its purpose.

8.3 Use Case

This section explains the use case diagram of the autonomous irritation vehicle. The use case diagram showed that who interacted with the system and what type of communications or actions were performed between the division of the system. As well as it showed the important functionality of the system. In our project the autonomous irrigation vehicle was the actor which interacted with the all component and performed the important functions. The line between the actor and the actions represented that the actor participated in the use case. Figure 26 shows the use case diagram of the autonomous irrigation system.

In the project the autonomous irrigation vehicle performed all functions such as receiving data from sensors, finding the path to reach the destination, disperse water and detects obstacles. In addition, the dashed line with arrow showed that the functionality of the one use case can be described in another use case. In the diagram map plotting extended to sensors because to plot the sensors data in two dimensional array the vehicle needed sensor data from sensor use case. Similarly, the map updating use case extended to map plotting because to update the two dimensional array, the sensor data had to be plotted in the array. Also, it extended to obstacle avoidance use case because first the vehicle had to find the obstacle location in order to update the obstacle locations on the two dimensional array map. In addition, the disperse water use case extended to the pathfinding use case because if the vehicle need to water a specific area, at first the autonomous irrigation vehicle had to reach the destination point to deliver the accurate amount of water in the specific location.

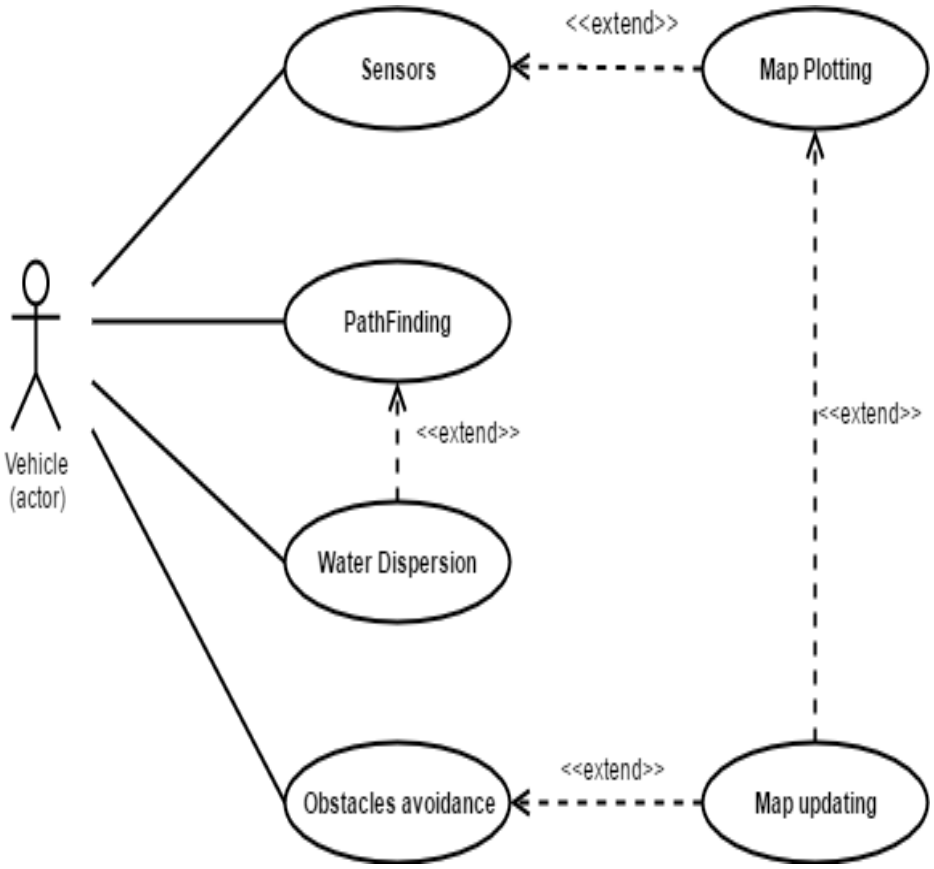


Figure 24: Use Case Diagram for Autonomous Irrigation Vehicle

8.4 Sequence Diagram

This section explains the sequence diagram of the autonomous irrigation vehicle. The sequence diagram shows the interaction between the each components of the autonomous irrigation vehicle. And it showed the time sequence of which component interact first. The each objects are shown in the square box and the vertical dashed lines represents the object's life line and the horizontal lines between the objects life lines represents the messages between each components. Figure 27 shows the sequence diagram of the autonomous irrigation vehicle.

The sequence diagram showed the five different objects, the first one was map plotting when the sensors data received from the mesh network team, all sensors locations plotted on the two dimensional array map. Then it sent the list of dry sensors that need to be watered to the pathfinding algorithm. The main responsibilities of the path finding was that it should find the shortest path reach its destination area. So when the pathfinding algorithm received dry sensors list, using data from the two dimensional array

map, first it checked which midpoint was the closest from the current location then it calculated the shortest path utilizing the nearest neighbor algorithm. In addition, the next component in the sequence diagram was the obstacle avoidance, while the autonomous irrigation vehicle travel to the destination point it checked if there were any obstacle on its way when it found any obstacles then it updated the obstacle location on the two dimensional array map. If the vehicle did not find any obstacle, it continued on the same path. The next sequence after the map updating was the dispersed water, when the autonomous vehicle reached its target location, it checked if the vehicle arrived the correct midpoint to disperse water. If the condition was true then it turned on the sprinkler to deliver the water. This message was shown in the sequence diagram as message between pathfinding and Disperse Water objects. The autonomous irrigation vehicle followed the same sequence whenever it traveled to the destination midpoints.

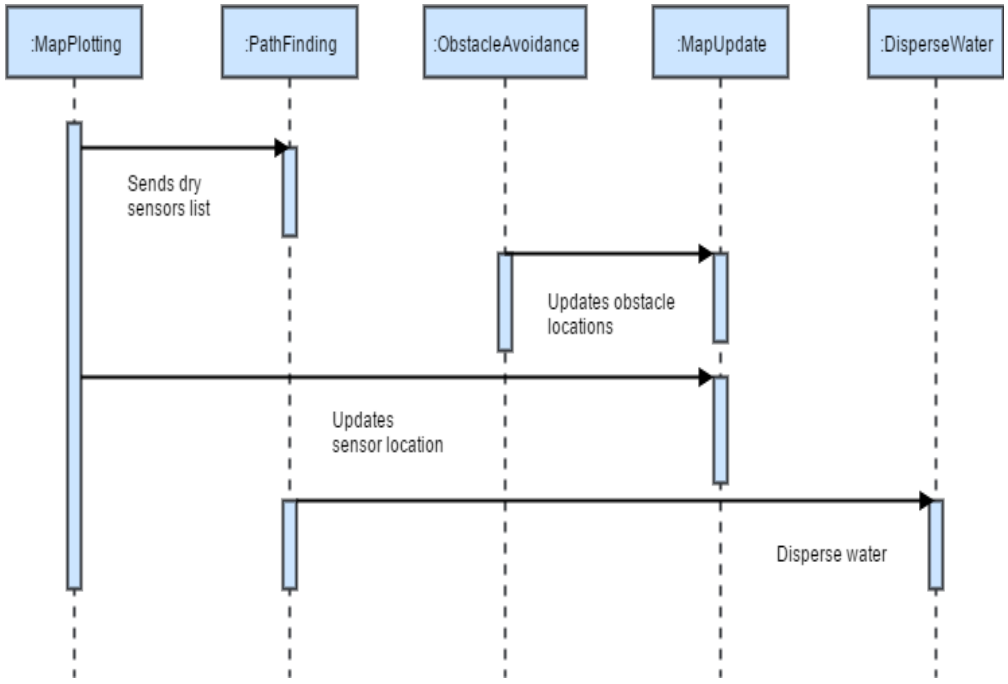


Figure 25: Sequence Diagram for the Autonomous Irrigation Vehicle

8.5 Methodology

A map of the area being watered was represented as a two-dimensional Locations object array with all its initial hazard level values initialized to zero. This was useful for navigating the autonomous irrigation vehicle based on sensor locations, destination midpoints, and obstacles in order to meet the autonomous irrigation vehicle's goal in the most water and energy efficient manner possible. Figure 28 as seen below, was a depiction of the mapping of the area.

This map was updated with sensor locations, midpoint destination locations, obstacle locations, and locations adjacent to obstacles to represent their risk of collision with an obstacle. All of this information was necessary for the autonomous irrigation vehicle to

safely travel around the area. With safe travel accomplished, the autonomous irrigation vehicle can deliver the precise amount of needed water to wherever the sensors in the area determine needed to be watered. By realizing and actualizing the watering of areas where watering was needed, the total distance that was needed to be traveled to reach every one of those locations exactly once has been minimized. With these goals being reached, the success of the autonomous irrigation vehicle becomes one step closer.

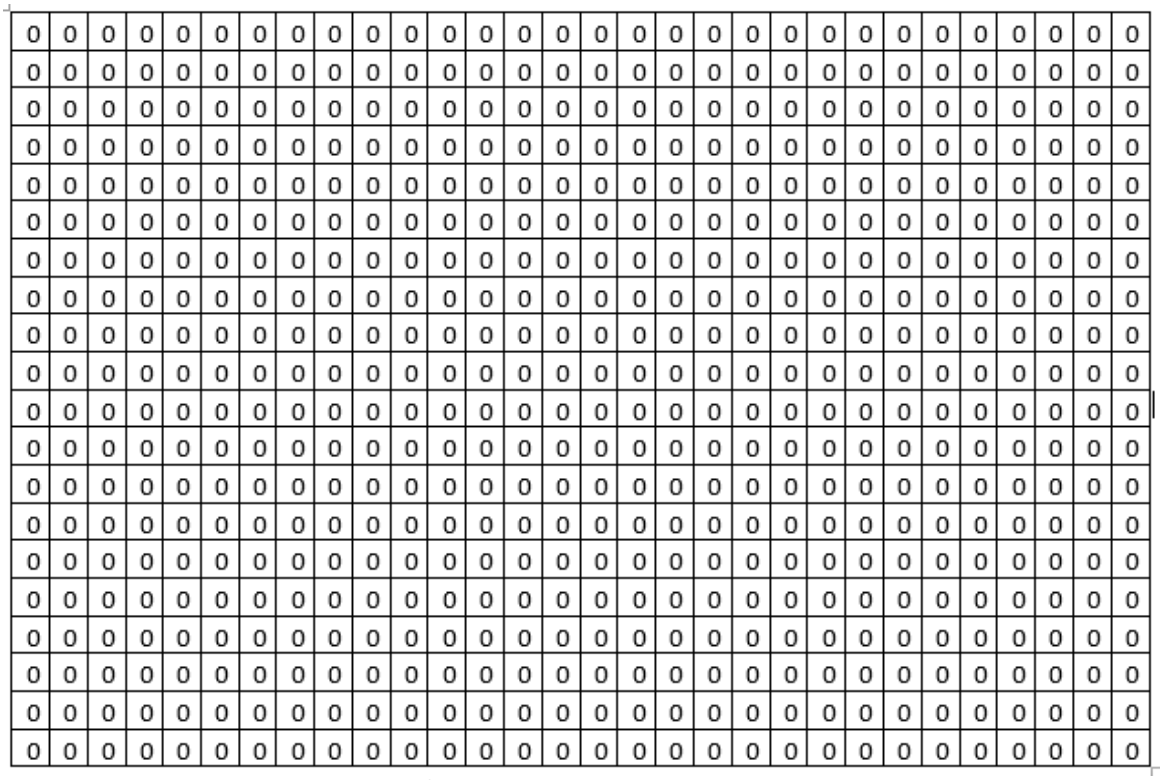


Figure 26: (30 x 20)ft^2 Two Dimensional Locations Object Array Map

8.6 Libraries

This section explains in detail about the libraries which were used in the software algorithm to effectively communicate with the autonomous irrigation vehicle as well as communicate with mesh network data. In programming standards libraries are resources which can be utilities while writing the program. The libraries usually contain preset of standard function and math formulas. For the autonomous irrigation vehicle project algorithm, we used Visual Studio which is an Integrated Development Environment(IDE), to maintain and develop the program. Integrated Development Environment was helpful to maintain different classes, variables and functions as well having it prevent any spelling error. The high level language C++ was used for path learning algorithm s. The integrated development environment had several open source libraries for C++, so those

libraries were utilized for the algorithm. Also, there several open source libraries were used for C++ programming language communication and plotting and graphs. Move over, ATmega328 microcontroller will used for communication part. Since the Arduino is an open platform, for this ATmega328 microcontroller communication the Arduino open source libraries will be used. The Node.js program was used the noble library, the node-beacon-scanner and bleacon libraries to specifically scan the BLE beacon packets. When the program scanned the beacon, it got the iBeacon's Universal Unique Identifier (UUID), Received Signal Strength Indicator (rssi), measured power, plus minor and major values. Having this value helped the vehicle to identify each beacon separately. Also, the JsonCpp library was used in path finding algorutham to parse the JSON file data in C++.

8.7 Communication Over Wireless

The main purpose of the autonomous irrigation vehicle was to efficiently navigate the target midpoint in possible shortest time without colliding into any objects. To reach the autonomous irrigation vehicle target, first it should successfully communicate with mesh network. The mesh network sent the all sensor details such as which sensors were dry, how much water it needed and where it located in the two dimensional array map. For the effective communication between the mesh network and the autonomous irrigation vehicle the IEEE 802.15 a wireless personal model was used. Once it established successful wireless connection between mesh network and the autonomous irrigation vehicle. Then the mesh network sent the sensor data in JSON format and the path finding algorithm parse that data format to C++. Then the pathfinding algorithm calculated the shortest path to travel to the target midpoints.

8.7.1 Map Plotting

After the transmitted data from the mesh network was parsed from JSON, the column x and row y locations of each of the sensors in the network was represented as indices for the two-dimensional array map and marked as the value one million counting up by the sensor's ID value to identify it. So sensor 0 would have the value 1000000 and sensor 1 would and the value 1000001 and so on up to 1999999, which was a distant enough of an upper limit for flexibility in future development and implementation. The count would stop at 1999999 and not go up to 2000000 because the 1 in the millionth digit of 1000000 to 1999999 was the flag that identifies this as a sensor specifically. This in the future could be increased to 1000000000 with 2000000000 as the upper limit if needed, but 1000000 and 2000000 seem sufficient enough for now. These values will be stored as a constant to be used in the software for this very reason. You can subtract the sensor constant value to get each sensor's ID number from the map. Below in Figure 29, was an example of what the two dimensional Locations object array could look like to the autonomous irrigation vehicle's computer processor. In the absence of the wireless mesh network, hard coded values were used for testing.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1m	0	0	0	0	0	0	0	0	0	1m	0	0	0	0	0	0	0	0	0	0	1m	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1m	0	0	0	0	0	0	0	0	0	1m	0	0	0	0	0	0	0	0	0	0	1m	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27: (30 x 20)ft^2 Map with Sensors Marked

As seen in (Figure 29) above, the locations of six sensors have been recorded on the map from the wireless sensor mesh network at the indices of [4][4], [14][4], [25][4], [4][15], [14][15], and [25][15]. They have been marked with 1m for readability in this example. In the software each of those locations would equal the sensor ID number plus one million for as many sensors listed to avoid conflicting with other values on the map that could be used to identify obstacles, locations adjacent to obstacles, and midpoint travel destinations for the autonomous irrigation vehicle to know where to go. In this case illustrated above the locations of the sensors would be marked from 1000000 to 1000005 with the sensor ID count beginning at 0.

Each sensor will also have its own class in the software storing its ID, x and y indices, a boolean dryness reading value, and a list of all the midpoints it borders. All of these sensor classes will be stored in a list indexed by their unique id number. This will make accessing the information about each sensor much easier for determining when to activate or deactivate the water delivery system, calculating midpoints between each sensor and their neighbors for the autonomous irrigation vehicle to deliver water, and for building a path to where needs water be delivered. The boolean value representing the dryness reading would be updated whenever the autonomous irrigation vehicle receives a signal from the wireless sensor mesh network indicating a change in the dryness level at a particular sensor. This helped the autonomous irrigation vehicle know whether or not

water needs to be delivered to any particular region near a sensor and when to shut off the water delivery system in a region its currently providing water.

By including the sensor ID with the one million digit flag, we were able to identify and access from its location on the map by hazard level without interfering with any other sensor value flags each sensor's class and therefore retrieve their data concerning their dryness readings, their neighbors, and the calculated midpoint destinations between them and their neighbors. The sensors could also be checked this way to make sure that they were marked at the correct locations as this data would also include their column x and row y index locations on the two dimensional Locations object array map. This made the communication through the data structures easier to determine to nearest sensor to calculate their midpoint watering destinations as well as getting a current reading from the specific sensors data to know that sensors dryness for building the path to water the area most efficiently. This was also useful in development to be able to know where each individual sensor was on the map as opposed to having a general single integer value flag just to identify that there was a sensor present in that location under the Locations object's hazard level.

All this information was important for knowing how to accurately provide just enough water to a specified region without providing too much nor too little water from the water delivery system onboard the autonomous irrigation vehicle. A major goal of the autonomous irrigation vehicle was better water usage efficiency compared to current inground sprinkler plumbing systems and having precise measurements of what locations in a region need water in crucial in achieving this goal. By plotting out the locations of the sensors that indicate whether or not a region needs water on a map representation for the autonomous irrigation vehicle to understand, it had more accurately calculated potential destinations and decided which of those destinations to provide water to while minimizing the waste of water usage. This information was also helpful to determine the most efficient way to deliver water to all of the destinations that need so that the autonomous irrigation vehicle would also reduce its energy consumption as a longer less accurate path could use up more energy to travel.

8.7.2 Sensor Midpoint Calculation

From the sensor data in the two dimensional Locations object array, we found which sensors have which other sensors as their neighbors in their network. This allowed for calculating the midpoints between two neighboring sensors as destinations for the autonomous irrigation vehicle to travel. Ideally the midpoint between two points were calculated by first getting the column midpoint x position by adding the two column x indices of the two edge points which was divided by two and then getting the midpoint row y position by adding the two row y indices of the two edge points which was also divided by two. This can be seen Formula 1 below:

$$m = (\frac{x_2 + x_1}{2}, \frac{y_2 + y_1}{2})$$

Formula 1: Midpoint Calculation

However the indices of the two dimensional Locations object array map were integers, so in the event that a midpoint has a decimal value it would be rounded down. For instance, if x_2 and x_1 were 5 and 2 respectively, the midpoint would be calculated as 3.5 but we would take it as 3 for the midpoint x-axis index. This slight adjustment was negligible and did not affect the performance in any real way. Most likely if two sensors were neighboring, they will have the same x values if their y values were different or the different x values if their y values were the same, but this may not be the case for every instance, which was why the input from the mesh network telling which sensors which other sensor as their neighbors have was important. Figure 30 below depicted a sample of a two dimensional locations object array as was explained.

The reason we were having the autonomous irrigation vehicle travel to the midpoints between water sensors rather than to the water sensors itself, is because by having the sensors read the soil moisture at the edges of the water dispenser's range we would get a more accurate reading of how dry the soil was to provide a suitable amount of water for it. If the sensor were to be in the middle of the path in which the autonomous irrigation vehicle was traveling, then the sensors would give inaccurate readings as soil would be more damp in the middle and the edges may still be dryer than they should be. Pathing was easier if you have coordinate locations to travel to, hence why the midpoints were calculated to give the autonomous irrigation vehicle travel destinations with the sensors at the edges of its path.

Each midpoint will have a class data structure for each one containing information on its coordinates on the two dimensional Locations object array map, which two sensors were at their edges, how many of those sensors were dry, and an unique ID number beginning at 0. The at the indices on the map for each midpoint location will store the value two million plus the midpoint unique ID number. The same constant upper limit constant used to check that to prevent the sensors from exceeding 1999999 to maintain their 1xxxxxx sensor flag so that the midpoints can have a 2xxxxxx midpoint flag where it would have a two in the millionth position to identify that this location contains a midpoint. The retrieve the unique ID number of a midpoint from the map, subtract two million from the value stored at it's indices in the two dimensional Locations object array map. For this we can set up the value three million as an upper bound for midpoint identification allowing for sensors to be marked on the map ranging 2000000 up to 2999999 in order to maintain the value two in its millionth position as its midpoint identification flag. For this the value three million could be saved as a constant variable to be used in code for simplicity. If in the future for some reason the sensor constant value were to be raised to the value of one billion with its upper bound at two billion, this new two billion upper bound would be used as the new midpoint constant value, however the maximum value of an integer in a 32-bit system was 2,147,483,647 which would have to be used as the new upper board

for the mapped midpoint values instead of three billion, unless if the autonomous irrigation vehicle were to be upgraded to a 64-bit system.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1m	0	0	0	0	2m	0	0	0	0	1m	0	0	0	0	2m	0	0	0	0	0	1m	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2m	0	0	0	0	0	0	0	0	0	2m	0	0	0	0	0	0	0	0	0	0	2m	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1m	0	0	0	0	2m	0	0	0	0	1m	0	0	0	0	2m	0	0	0	0	0	1m	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28: (30 x 20)ft^2 Map with Midpoints

As seen in Figure 30 above, six sensors have been recorded on the map at indices [4][4], [14][4], [25][4], [4][15], [14][15], and [25][15] marked with 1m for readability in this example. Midpoints had been calculated at [9][4], [19][4], [4][9], [14][9], [25][9], [9][15], and [19][15] and marked with 2m for readability. Between the six sensors there were seven midpoints with unique IDs 0 through 6 for seven total. On the two dimensional Locations object array map within the software these would be marked as 2000000 through 2000006 at each of their respective x and y indices. With the map on the autonomous irrigation vehicle always being updated, it should be readily available to make any adjustment as needed in real time. As such, the autonomy of the autonomous irrigation vehicle becomes more and more of a realization.

8.7.3 Bluetooth Sensor

The Estimote Bluetooth sensors were used as a mesh network node on the terrain. There were six sensors used in this project to test the irrigation vehicle. The sensors were permanently placed on the grass to communicate with the vehicle. The Bluetooth Low

Energy (BLE) sensors communicated with the vehicle through the Node.js program. The Node.js is a JavaScript run-time environment and an open source server environment. It runs on multiple platforms, and specifically supports Raspbian. Also, Node.js is ~~a~~ lightweight and memory efficient since it runs on a single thread. The Node.js program was used the noble library and the node-beacon-scanner library to specifically scan the BLE beacon packets. When the program scanned the beacon, it got the iBeacon's Universal Unique Identifier (UUID), Received Signal Strength Indicator (rssi), measured power, plus minor and major values. Having this value helped the vehicle to identify each beacon separately but it doesn't provide the distance between the vehicle and the node. The ratio of the rssi and the calibrated measured power value is used to calculate the distance between the vehicle and the sensor node. Once all the required values are calculated in Node.js the values are pushed back to the path learning algorithm to communicate with the vehicle. For this communication between the two programs, a JSON (JavaScript Object Notation) file is used since JSON is a light weight data interchange format and language independent. All the beacon data is saved in the JSON file and the path learning algorithm (C++) reads the JSON file and saves the beacon ID and the distance in the program. Using the distances of all the beacons, the vehicle's curent position is calculated. The path learning algorithm uses all these values to navigate to the target midpoints. Figure 0 Shows the picture of estimote bluetooth sensor.

Figure 29 Estimote Bluetooth Sensor

8.8 Water Dispersion

Initially the water delivery system will be off while the autonomous irrigation vehicle is driving to its destination since the water is not yet needed until it reaches where is needed to be delivered. When the autonomous irrigation vehicle reached a midpoint destination with a sensor or two in range at the edges, the autonomous irrigation vehicle turned on its blue LED to represent its future water delivery system to provide a sufficient amount

of water to the specified region as necessary. By having the water delivery system off until needed, a significant amount of water should be responsibly retained, meeting one of the major goals of the development of the autonomous irrigation vehicle to save water.

Water conservation is a very important subject that affects many industries and the lives of many people. There are many economic benefits to water conservation, water costs money to consume and by reducing the amount used by who ever needs to irrigate their land, an individual could save a decent amount of money. Many regions also suffer from severe drought and water scarcity, which in turn could increase the cost of water as more scarce resources that are in high demand have a tendency to increase in monetary value and water is something that will always be in high demand as all biological life as we know it requires it for its very existence. Developing more efficient water use techniques could help alleviate the strain felt by many of these regions that are subject to these conditions, by helping many of their vital systems that rely heavily on water to function. The autonomous irrigation vehicle being developed is aimed mostly towards residential lawns, but it could be the building block for something geared more towards the other markets, such as the agricultural industry. This would benefit the environment with water conservation and keeping lush natural areas alive to enjoy, as well as proving the industries interested in this technology a product to help them save money on water consumption costs and increasing their net profits and potentially increase the value of their market shares if the general public sees them as doing something good that they may want to invest into.

After the timed delay ended to represent the sensor sending the autonomous irrigation vehicle a reading indicating that it was no longer dry, it had shut off its blue LED representing its water delivery system and moved towards its next midpoint destination to repeat the process. This was so that it does not provide more water than is required to regions that do not need water. If the midpoint destination had two edge sensors it would wait for both sensors to give back readings that they were no longer dry before moving on. However, in later development it could also be programed to calculate another midpoint between the current midpoint and the dry sensor in this scenario. It would use this new midpoint to travel closer to the sensor that is still dry and away from the one that now has enough water until the sensor it is moving closer to is no longer providing a reading of being dry, but whether or not this addition would provide any significant benefit would need to be tested. One sensor could provide a reading of not being dry anymore shortly after the other one does. Also, if there were any obstacles between the midpoint and the sensor that is still dry, then the distance traveled to get a little bit closer may not be enough to show any real difference.

8.9 Pathfinding

The purpose of pathfinding was to determine the shortest path from one location to another, it was heavily based on graph theory. This was very important for the

autonomous irrigation vehicle to determine where to go and how to get there in order to provide water to destinations where it was needed as efficiently as possible. The more efficient the autonomous irrigation vehicle travels to its destination, the less power was consumed for it to travel. Reducing power usage was a very important factor for the consumers the autonomous irrigation vehicle would be marketed to as something that used less power, as opposed to something that used more, would be more desirable as far as financial costs and potentially environmental impacts that many who would use such a product may have interest in.

In graph theory, graphs were mathematical structures that consists of locations known as vertices and the lines connecting the vertices together known as edges. A graph can be directional where the algorithm can from a particular vertex to another particular vertex along an edge in only one direction, or unidirectional where it can move in either direction between the two vertices. Graphs can also we weighted where each edge has a weight assigned to it to determine the cost of traveling along that particular edge as opposed to another, or unweighted where each edge was treated the same as far as their cost to travel. These costs can represent anything from distance, risk potential, or any other factors that may make an edge more or less favorable to travel. In many directed graphs that were also weighted, traveling in the opposite direction that an edge was directed towards would result in a negative valued weight for that edge. Graph theory can be applied to many purposes such as telecommunications, data organization, computations, physics, chemistry, biology, and finding the best directions to a physical location from another.

For the autonomous irrigation vehicle, the two dimensional Locations object array map will serve as a weighted undirected graph for the pathfinding algorithm it used to reach its destinations, where each index in the array was a vertex and the edges were the movement from any location to any of its adjacent locations. Moving from any location to any other location would incur a travel cost of one plus the hazard risk value stored at any location to represent moving from one vertex to another along a weighted edge. The used graph was unidirectional, so negative valued weights were not a concern as traveling from one location on the two dimensional Locations object array to another location still added to the total distance traveled even if the path the autonomous irrigation vehicle were to backtrack its path as it travels, if that was the most efficient way to reach all of its destinations with the least distance traveled and most obstacles avoided. The autonomous irrigation vehicle factored in obstacles in the area to avoid because these obstacles would slow down its travels, stop it in place if it got stuck, or cause serious damage that would be expensive for the consumer to repair. By building a path that kept the autonomous irrigation vehicle safe as it traveled efficiently as possible to its destination, the total maintenance costs should also be reduced significantly.

8.9.1 Algorithm Concept

The autonomous irrigation vehicle began at its home station. When it received a signal from the mesh network to water a location or locations, it loaded a map with all currently known terrain data with coordinates of each sensor from the mesh network that required watering in a hamiltonian graph. The shortest path to reach each destination location between dry sensors exactly once was calculated creating a list of coordinates to reach each target location that needed to be watered and factoring in known obstacles in the area, when the list of location was empty the vehicle returns to its home station. Choosing which location to travel to deliver the water to was a very important decision to reducing the total distance the autonomous irrigation vehicle travels. This was tested in the absence of a mesh network by hard coded sensor information values. The vehicle will be attached to a retractable tether from its home station to provide water and power by a team of mechanical engineers.

The Dijkstra's Algorithm was a classic path finding algorithm that found the shortest path from a starting point to all other vertices in a graph. It works with directed weighted graphs, but was unable to work with edges that have negative valued weights. It uses a boolean array to keep track of visited vertices and a priority queue that begins with only the starting edge from the first location in it. While the priority queue was not empty, it traveled to the next edge, checked it as visited in the boolean array, and if it made it to its destination from that edge it returned its weight. If it has not made it to its destination, it added all unvisited edges neighboring the current edge to its priority queue and repeated the process. Bellman Ford was an improvement from Dijkstra’s algorithm that was able to handle negative edge weights with no problem.

This came across as an example of the classical “Traveling Salesperson Problem,” which asked to find the shortest possible route to visit each “city” given a list of “cities” and the distance between each pair of “cities. Neither the Dijkstra's algorithm nor Bellman Ford may be suitable for the Traveling Salesperson Problem in the problem we were working with. A more appropriate method of approaching this would be using a combination of an uniform cost search and a greedy algorithm such as repetitive nearest neighbor algorithm or an A Star search algorithm, a Markov Decision Process could also work well for this too. With either A Star search or Markov Decision Process, heuristics and reward functions could be based on a water dispersion rate, an energy consumption rate, and time or distance costs. Out of these repetitive nearest neighbor algorithm appears to be the best candidate, which was when you run a nearest neighbor algorithm using each travel position as a starting position. Repetitive nearest neighbor algorithm typically out performs nearest neighbor algorithm, however the autonomous irrigation vehicle was attached to a cord at a fixed starting location, so nearest neighbor algorithm by itself should be sufficient enough. The nearest neighbor algorithm started from a location and chose the nearest unvisited destination by comparing the cost to travel from its current location to all other available destinations and selected the one with the lowest cost for it to move to and repeated the process from that next position until all destination locations have been visited. This generated at least as many paths as there were destinations, more if at any point there were multiple neighbors who were equally close and the path needs to split. From all the paths generated, the path with the shortest distance path, or the path with the lowest travel cost if dealing with other constraints such as obstacles,

was chosen. Figure 31 shows how the autonomous irrigation vehicle will be able to traverse its surrounding environment. with four sensors

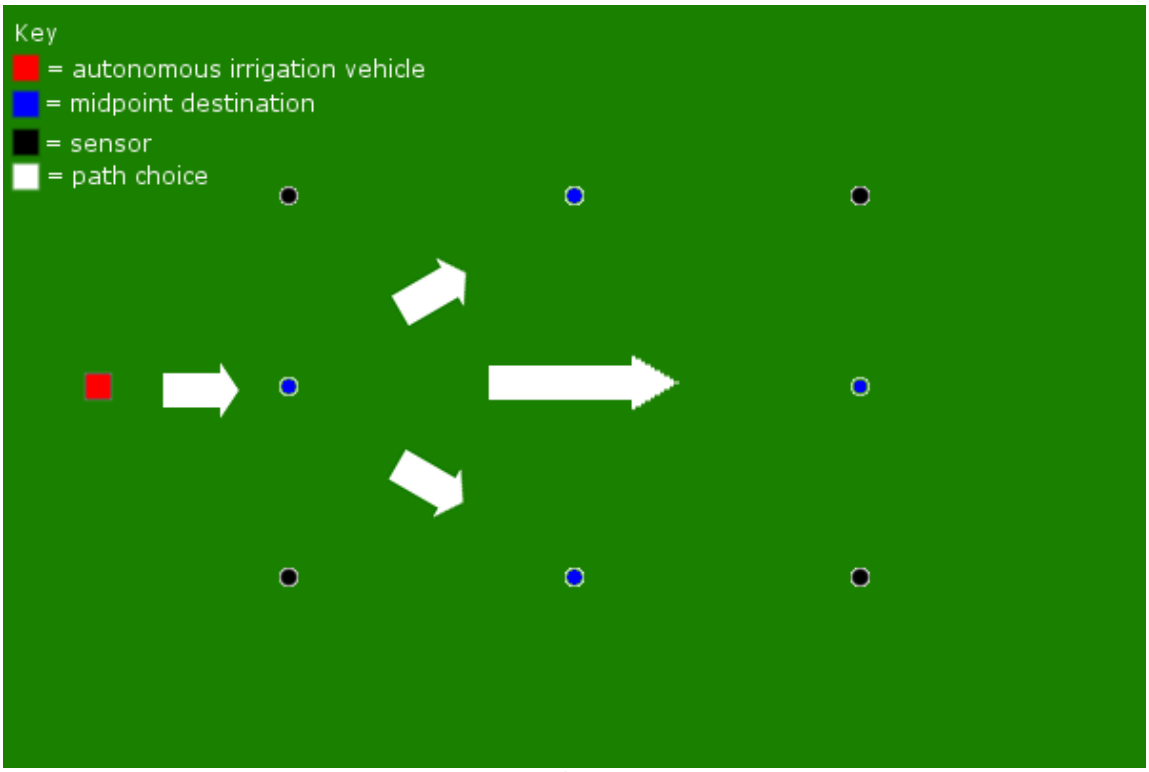


Figure 29: (20 x 30 ft^2) Choices with Four Sensors

In our implementation of this, we will be factoring in the dryness reading of the sensors at the edges of our midpoint travel destinations. It looked to the destination to its left and to the one to its right. If one of them had the more dry sensor readings than the other, it selected it and added it to a list of destinations to travel to. If they both had the same dry sensor readings and they both had sensors that read dry, it chose the next midpoint destination in front of it. If neither of the two side midpoint destinations had sensors that indicate dryness and required watering, the nearest neighbor algorithm was performed to find the nearest sensors with a dryness reading and the autonomous irrigation vehicle travels to the closest midpoint destination in its immediate area to get to them sensor to repeat the process. Once it had its list of destinations, it began to travel to each of those locations in the order of start to finish from its chosen list. Figure 32 shows how the autonomous irrigation vehicle will be able to traverse its surrounding environment with six sensors.

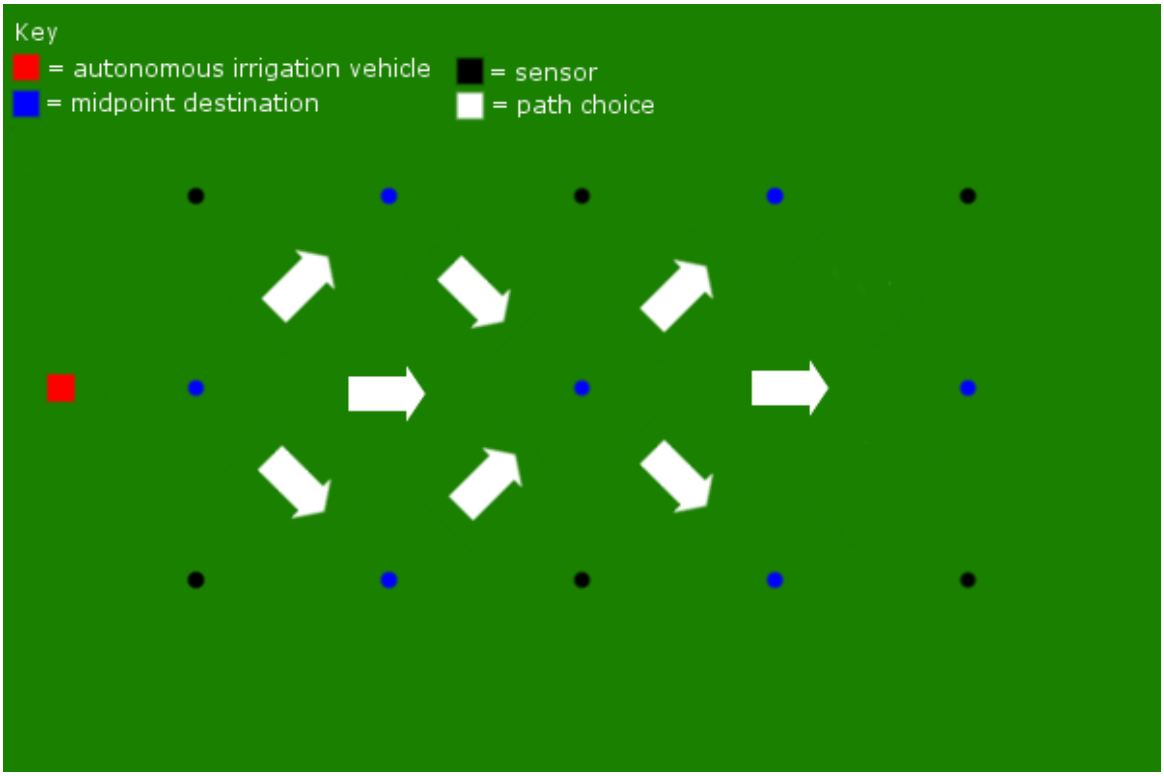


Figure 30: (20 x 30 ft^2) Choices with Six Sensors

Our sponsor offered to provide us with four sensors to test with, for which with only four sensors we would be able to test the single direction decisions of left, right, forward, and finish without needing to search for more with the nearest neighbor algorithm. However, test cases with more sensors could still be tested by supplying the autonomous irrigation vehicle inputs with the same format as it would receive from mesh network to make it think it was in a larger mesh network with more sensors. When it reached a midpoint destination, a timed delay could start in order to simulate a dry sensor at the edge of the midpoint where an LED light would turn blue to indicate watering until the team of mechanical engineers implement the watering system. When calculating the path it would update the information of the sensor object to indicate that it was no longer wet and move on to the next midpoint destination to water the next sensor or sensors. By doing this you could theoretically make the robot think it was in a mesh network with as many sensors as you want to test and refine the algorithm to be the most optimal. This could be done in any physical environment, from an open field to a yard that wraps a house at odd angles as the computer vision obstacle detection system should still be able to detect the obstacles that would be present in any of these environments. A hard coded test case with six sensors were half were dry and half were not was sufficient for the demonstration to show the vehicle's capabilities.

As seen above, an area with six sensors would better illustrate a scenario where the autonomous irrigation vehicle may provide water to the left or right midpoint destination close to where it began on the left side of Diagram 2 and none of the nearest immediate midpoint may have had dry sensor readings, but there were still be dry sensors further away in the area towards the right side of Diagram 2. In this scenario, the autonomous

irrigation vehicle had shut off its water delivery system and travel to the midpoint destination in the center of Diagram 2 in order to travel to whichever midpoint destination need water near the right side of Diagram 2 where it would reactivate its water delivery system until there were no more dry sensor readings. For larger areas than were tested, multiple dry sensors could be detected at equal distance from the autonomous irrigation vehicle, the algorithm would need to branch and overall chooses the one with the list of destinations that have the lowest cost for reaching all of the locations that need to be watered.

8.9.2 Ideal Optimal Path

In order for the autonomous irrigation vehicle's path to each watering location to truly be optimal, the shorted path to the next destination needed to be found to minimize energy consumption cost of the autonomous irrigation vehicle's travels. The path it travels also needs to be a safe one, so it will factor in both distance and hazards of potential paths. By making the autonomous irrigation vehicle more risk averse when plotting its course, it would minimize damages, maintenance or repair, as well as the risk of the autonomous irrigation vehicle getting stuck on an obstacle that could prevent it from traveling around the area performing its functions, to the customer's annoyance. For this, the breadth first search algorithm was originally chosen. Figure 33 shows a depiction of how breadth first search algorithm works.

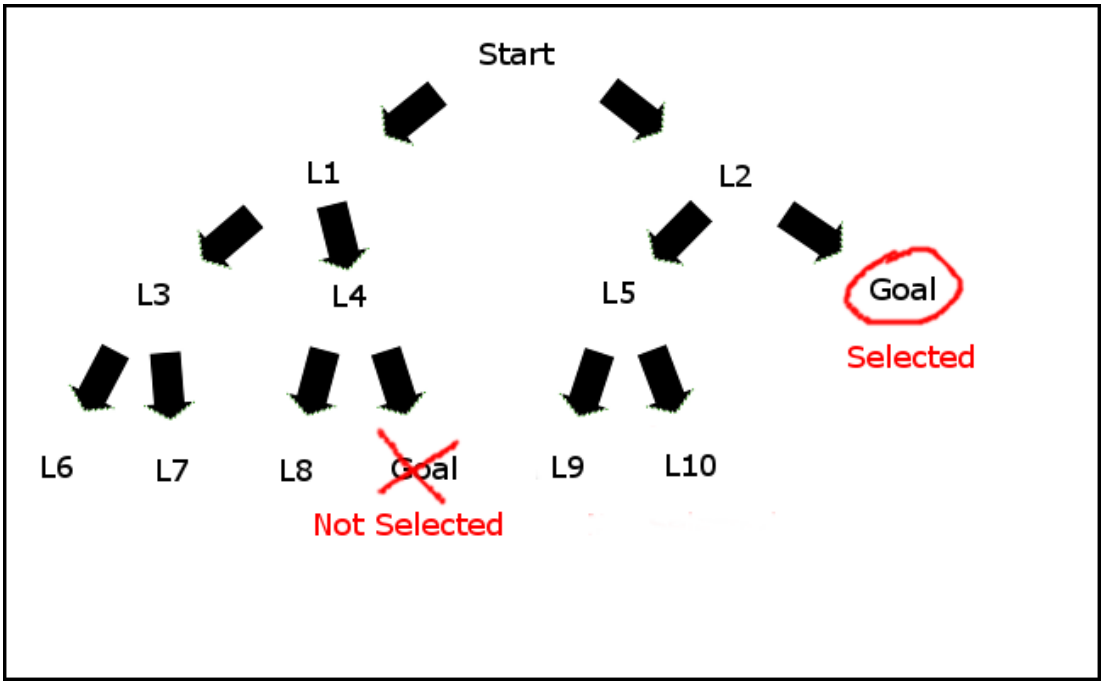


Figure 31: Breadth First Search Illustration

The more optimal A Star search was later chosen over breadth first search to be used to plot a course from one location to the destination's location. The way that A Star search works was similar to breadth first search in that it searched all paths in a uniform depth from its starting location and expanded outward at the same depth for each path storing all adjacent locations stored in a queue ordering by first in first out fashion. It marked the locations it had visited to prevent a path from repeating locations and traveling more than it should. This had been achieved using a second two dimensional boolean array map, with the same dimensions as the Locations object map, to mark locations as visited or not based on their indices from the Locations map. A Star search differed from breadth first search in that it included a heuristic value to give some locations a lower and more favorable cost to travel.

For the autonomous irrigation vehicle, each location a potential path entered, it updated its total path cost by adding one plus the value stored in that location on the two dimensional array map as a backwards cost and the distance of that location to the goal destination the path was built towards as a forward cost. A location with the integer nine value stored in it will be considered invalid to travel to and avoided. A heuristic value was calculated by summing the backwards cost, forward cost, and number of adjacent obstacles and stored in the array. Each location also stored the coordinates of the last path that search it that found the lowest heuristic cost. A location with the integer nine value stored in it will be considered invalid to travel to and avoided. Once a potential path reached the destination, it built a path backwards through the previous stored location coordinates from the goal to start, for this reason we search starting from the goal to start. This returned the lowest cost sequence of locations that formed the path from the start location to the destination location used as the course for the autonomous irrigation vehicle to follow to reach its destination. Figure 34 shows a depiction of how A Star search algorithm works with hazard costs.

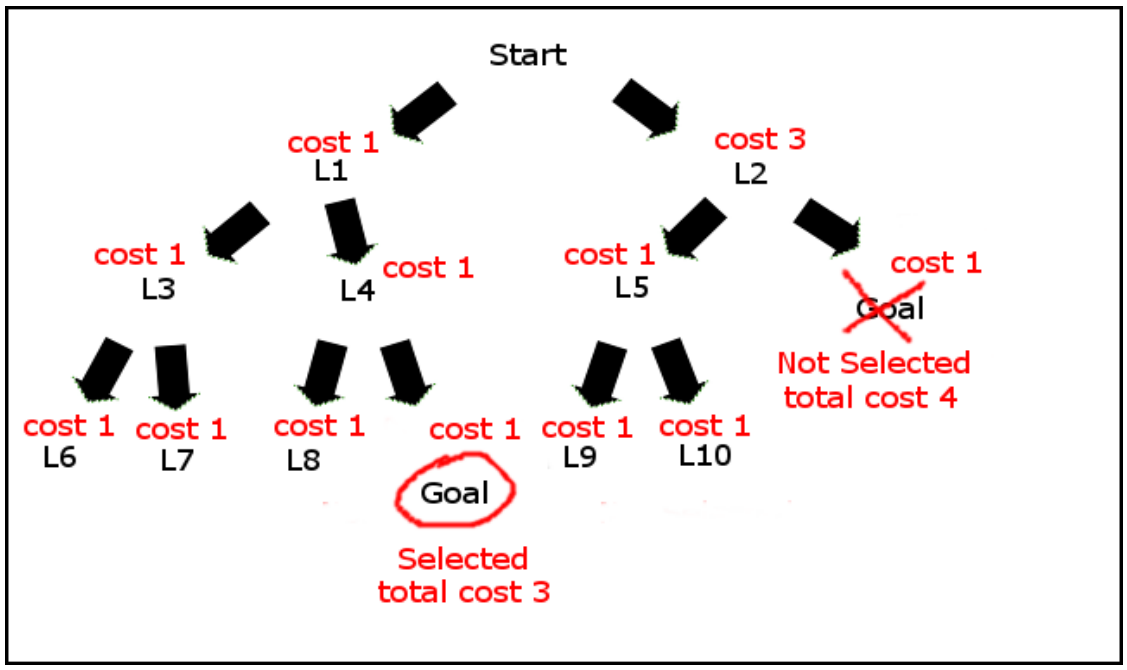


Figure 322: Modified A Star Search with Hazard Costs

To keep track of the sequence of positions that build each path, each position was stored as a Locations object. These objects stored their count of adjacent obstacles as a hazard level, backwards and forwards cost, heuristic value, row and column coordinates, and the coordinates of the location that last searched it with the lowest heuristic value. These objects were stored in a vector in the order of previous location and to their next location, the destination.

8.9.3 Path Localization

The autonomous irrigation vehicle read in data from the proximity sensors on each moisture sensor in JSON format with their uuid and distance from the vehicle based on rssi and stored this information inside the objects of their respective moister sensors. As the vehicle traveled, it updated the distance it was from each of the proximity sensors by reading in more JSON files and used the three closest sensor distances and row and column coordinates to calculate the actual coordinates of the vehicle on the map to compare with where it thought it was on its path. If the vehicle was off course and somewhere it was not supposed to be, it recalculated its path from the calculated location from the proximity sensors to get to where it needed to go.

8.10 Computer Vision

Computer vision is an interdisciplinary field which ties in concepts from biology, computer science, and engineering to create models of human like vision processing in computational systems that can perform tasks based on visual input the way a human can, if not better. Its goal is to make sense out of visual input in the form of an image or video for computer systems to be able to understand and work with for whatever functions to be performed with it. It is able to process both three dimensional and two dimensional visual data input for performing various tasks such as recognition and motion analysis. Much inspiration for computer vision has come from neurobiology where researchers have developed methods to mimic how the human brains works to an extent in the field of machine learning through the use of perceptrons and convolutional neural networks.

Computer vision has been around for decades and has evolved much throughout the years. It was first envisioned in the 1960's but it was in the 1970's when many of the fundamental algorithms that built the field, such as edge detection, were developed. Many more recent computer vision researchers have pushed to boundaries of the field even further for more practical applications such as autonomous vehicles, medical research, and even the pioneering of the field of computer touch. At MIT, Edward H. Adelson works on a project called GelSight which aims to give computers and robots the ability to understand tactile sensation. Audio recognition software is already available in the market in popular products, such as Dragon speech recognition software, with further advances in computer vision and the beginnings of computer touch many mechanical devices are gaining the ability to process information more like the ways a human can which could lead to many more promising prospects.

The autonomous irrigation vehicle used Lidar, a more simple form of computer vision, to detect obstacles for it to avoid colliding into. The reason for this is that many terrains, such as residential lawns, may contain many obstacles, whether that be rocks, trees, ditches, or anything else that may hinder, if not completely stop or damage, the autonomous irrigation vehicle. It needed to be able to sense its surrounding area in order to safely traverse to where it needed to go to deliver water. Computer vision has been used many times in the research and development of autonomous vehicles to navigate through an environment and to determine whether or not an object is coming towards or moving away from said vehicle, as well as being able to identify objects. Object identification is extremely important in the field of driverless cars for instance, a driverless car would needs to be able to identify traffic signs and signals as well as avoiding other vehicles and pedestrians in order for it to be viable for the consumer market. There may not be any need for something as complex as a convolutional neural network made of perceptrons or classifiers trained to recognize any sort of object since the autonomous irrigation vehicle only needs to be able to locate where hazardous obstacles are to avoid them, so something a little bit more simple similar to a radar system could be very effective to meet this goal.

8.10.1 Obstacle Avoidance and Collision detection

The ultimate goal of the autonomous irrigation vehicle was to effectively and efficiently navigate the target midpoint in possible shortest time. To reach the autonomous irrigation vehicle target, it effortlessly detected the obstacles and prevent the collisions. This section explains details about how the computer vision obstacle detection system was used in autonomous irrigation vehicle to detect obstacles. While the autonomous irrigation vehicle traverses the terrain several times, it adapted and learned its surrounding and permanent obstacles. And, Figure 35 shows the autonomous irrigation vehicle's obstacle detection and avoidance flowchart.

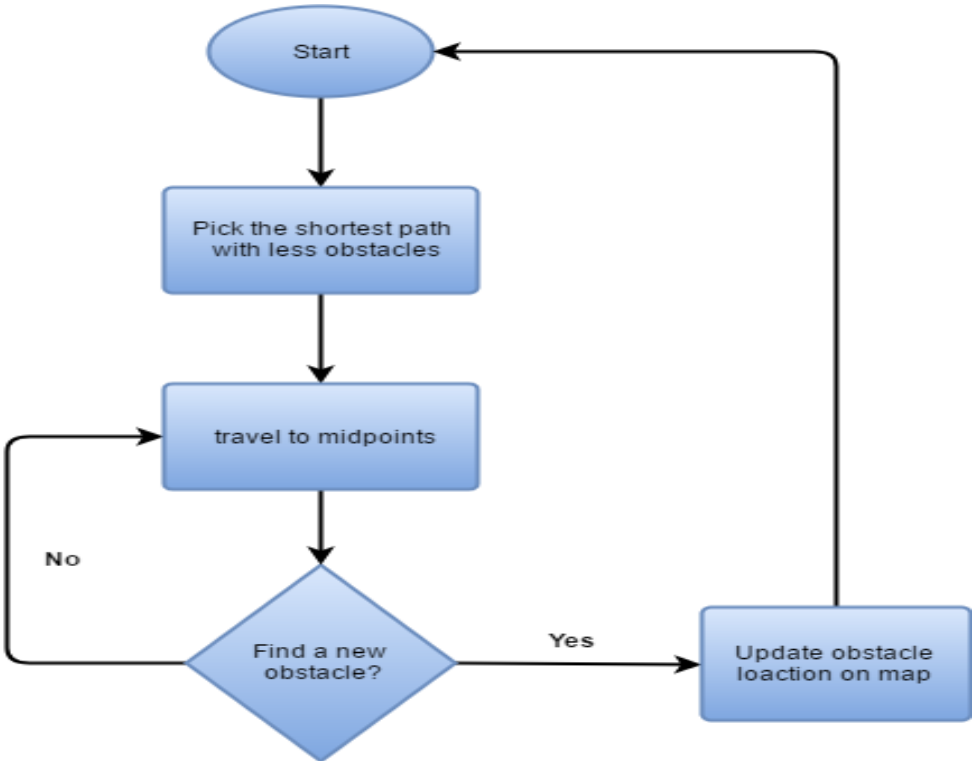


Figure 33: Obstacle Detection and Avoidance Flowchart

During the learning phase the autonomous irrigation system takes some time to learn the path while it travel to the destination midpoints as well as it used the computer vision obstacle detection system which was LIDAR, to detect any obstacles such as small trees, ditches, big stone, any hard objects, lawn status and flipping over pond. whenever the computer vision obstacle detection system detected any obstacle, it updated the obstacle location and locations adjacent to obstacles on two-dimensional array map.

whenever the autonomous irrigation vehicle travel to the midpoints it persistently updated the obstacles coordination on two-dimensional array map. Having this updated map data helped the autonomous vehicle to reach the target points without being stuck by any obstacles. Besides, all of this obstacles data were necessary for the computer vision obstacle detection system to safely travel around the terrain surroundings to deliver the correct amount of water. The path finding algorithm not only generated the lowest cost

path but it also created the lowest cost path with less obstacle levels path. The autonomous vehicle had a option to choose the shortest path with less obstacle, by choosing a less risk path keeps the autonomous irrigation vehicle safe as it travelled efficiently as possible to reach its destination midpoints. Also, it helps the consumer to maintain the autonomous vehicle effortlessly. The computer vision obstacle detection system need to be able to identify its surrounding in order to travel around the lawn to water the target midpoints.

Avoiding the collision was a crucial part of the autonomous irrigation vehicle. Once the map was updated with the locations of the obstacles and locations adjacent to the obstacles, the path finding algorithms recalculate its path to reach all the midpoint destinations that need to be watered based on the lowest cost factoring in the shortest distance while avoiding obstacles as well as passing through the fewest locations adjacent to those obstacles if possible. The Path finding algorithms creates list to keep track of which are the sensors need water, this will enable the autonomous irrigation vehicle to reach its destination more effectively. Once the autonomous vehicle reached all the midpoints in the list, it will return to the home station to charge or wait for another set of data from mesh network.

The computer vision detection system used LIDAR to detect the obstacles. The LIDAR played a significant role in the autonomous irrigation vehicle's obstacle detection process. The light detection and ranging will be placed on the autonomous irrigation vehicle to detect objects and avoid collision. The light detection and ranging detected the objects in up to 6 meter distance and the specific LIDAR model A1m8 can go as high as 6 meter and as low as 0.2 meter. Even though, the autonomous irrigation vehicle going to use only 180 degree, this A1m8 model has angular field of view range from 0 degree to 360 degree and it provides laser wavelength as low as 775 nanometer and as high as 795 nanometer. As well as it has maximum sample frequency of 2010 Hertz. This distance range would be helpful for the autonomous irrigation vehicle to detect the obstacle before reaching too close to the object, this prevented the vehicle from hitting the object. Moreover, when the LIDAR detected the object using lights, it also recogni what types of objects it was such as tree, water, rocket or some type of statue. As we know some of the objects such as tree and statues were most likely permanent objects. Based on this data the computer vision detection system updated on the two-dimensional array map as permanent objects or non-permanent objects.

8.10.2 Elevation Detection

As the autonomous irrigation vehicle traverses its course providing needed water to dry regions, one serious type of hazard that could entrap the autonomous irrigation vehicle could be sudden elevation changes. This includes the possibility of the autonomous irrigation vehicle calling into a hole, a ditch, or even a swimming pool where either it could get stuck and unable to fulfil its irrigation duties or severely damaged and needing costly and inconvenient repairs or even replacement depending on the extent of damages.

Another possibility could be going up a slope too steep where it could tip over which could result in similar predicaments as the ditch, hole, or pool as far as becoming immobile or damaged enough to no longer serve its purpose.

One solution that had not been implemented but could later be implemented in future projects would to have the on board LIDAR elevated and tilted at an angle where its constantly scanning one hundred eighty degrees in front of it. This would be measuring the value for the hypotenuse of the right triangle formed from the ground to the LIDAR and the distance that the LIDAR reaches on the ground horizontally from the autonomous irrigation vehicle. This value can also be calculated by the distance of the LIDAR from the ground divided by the cosine of whichever angle it is tilted at to check if it matches the range the LIDAR is measuring. If this value is constant or falls in a range of values depending on the autonomous irrigation vehicle's tolerance to unevenly elevated terrains, this signifies that it's safe to traverse across level or tolerable the area of the region being irrigated.

One possible angle the LIDAR could be tilted could be at forty-five degrees, this would give a one-to-one ratio for the height of the LIDAR from the ground the distance from the autonomous to where the LIDAR would reach the ground. Sudden changes in the measured distance measured by the LIDAR at is less than the calculated hypotenuse based on the angle and height of the LIDAR from the ground would indicate an obstacle calculated to be the measured distance multiplied by the sin of the angle in degrees as feet away from the autonomous irrigation vehicle. If the measured distance is greater than the calculated hypotenuse, this would indicate an obstacle in the form of a hole, ditch, swimming pool, or any hazardous drop in elevation and the obstacle would be marked on the two-dimensional Locations object array map at the last known location before where this change in detected range occurred.

For instance, if the LIDAR was three feet off the ground it would reach three feet in front of the autonomous irrigation vehicle with the range measured by the LIDAR of four and-a-quarter feet on a flat and level surface, as three divided by cosine of forty-five degrees is approximately four and-a-quarter. If the LIDAR where to measure two and eight tenths feet, this would indicate there is an obstacle close by as two and eight tenths is less than four and-a-quarter. The obstacle's location would be determined to be two and eight tenths multiplied by the sin of forty-five degrees, which comes out to be two feet and thus an obstacle would be marked as two feet away from the autonomous irrigation vehicle in the direction the LIDAR was pointing and marked on the two-dimensional Locations object array map accordingly. If the measured value were to dramatically jump from four and-a-quarter feet to twelve and-three-quarters feet, this would indicate a sudden drop of six feet, like in the case of a swimming pool, where the last location on the two-dimensional Locations object array map that measured four and-a-quarter feet would be marked as an obstacle to avoid. Figure 36 shows a depiction of the LIDAR working at a 45-degree angle.

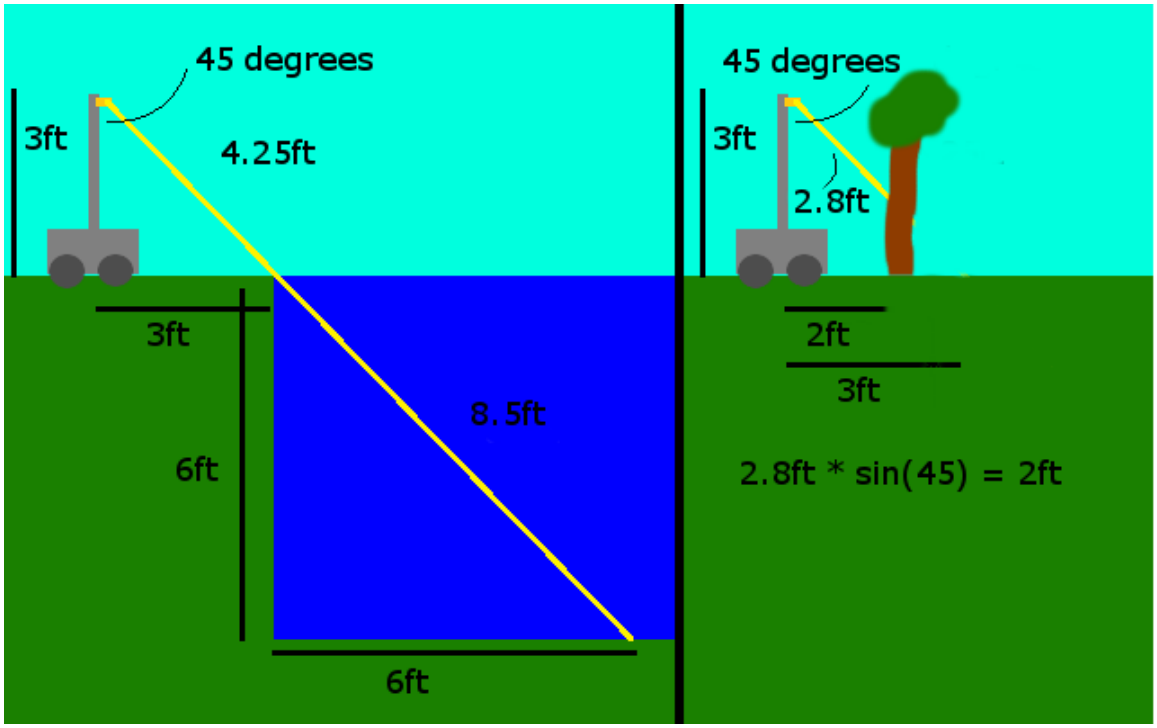


Figure 34: Elevation Detecting LIDAR at 45º Angle

The range at which the autonomous irrigation vehicle can detect obstacles and elevation changes away from itself could be increased without increasing the height at which the on board LIDAR is from the ground would be to increase the angle at which it is tilted. This would lose the one-to-one ratio, but the increased distance away from the autonomous irrigation vehicle could still be calculated by multiplying the height at which the LIDAR is from the ground by the tangent of the new angle and may be more advantageous to do so. The hypotenuse value for a level surface could still be calculated by the height of the LIDAR from the ground divided by the cosine of the angle and the distance from the autonomous irrigation vehicle to any obstacles detected by a measured value less than the hypotenuse value could still be calculated by the measured distance multiplied by the sin of the angle.

For instance, if the angle is increased to seventy-one and six tenths degrees and the LIDAR remains three feet off the ground, the LIDAR would reach three multiplied by the tangent of seventy-one and six tenths degrees which is nine feet. This is triple the range away from the autonomous irrigation vehicle than it would be if the LIDAR was tiled at a forty-five-degree angle. The hypotenuse of the right triangle formed when on a level terrain would be three feet divided by the cosine of the seventy-one and six tenths-degree angle of the LIDAR's tilt which is approximately nine and a half feet. In this case if the LIDAR were to measure eight and four tenths feet as opposed to nine and a half feet, the location of the obstacle that caused this change could be calculated by multiplying eight and four tenths by the sin of seventy-one and six tenths degrees at a distance of eight feet away from the autonomous irrigation vehicle and marked on the two-dimensional

Locations object array map accordingly. Likewise, a measured distance greater than nine and a half feet would indicate an elevation drop where the last location before this change would be marked on the two-dimensional Locations object array map would be marked as an obstacle for the autonomous irrigation vehicle to avoid. Figure 37 shows a depiction of the LIDAR working at a 71.6-degree angle.

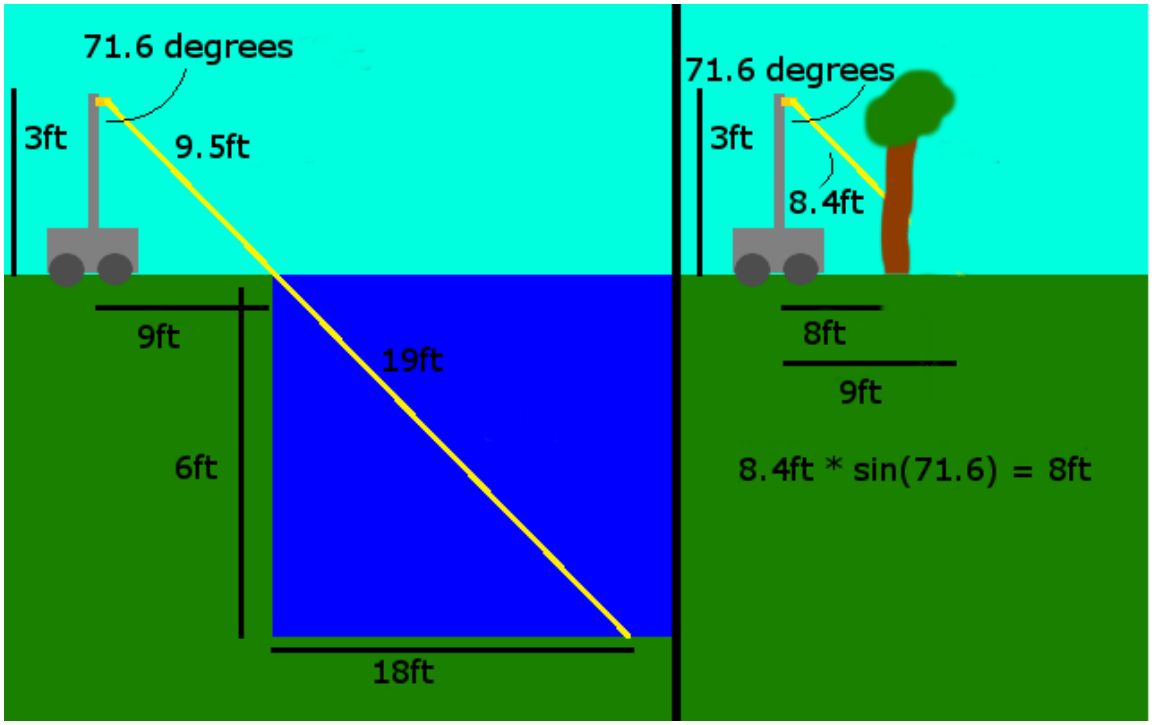


Figure 35: Elevation Detecting LIDAR at 71.6º Angle

The angle could be increased further to increase the range more if need be, but with increased ranges on LIDAR comes with greater financial costs. Realistically the LIDAR does no need to reach the bottom of a region in a deep drop in elevation like a swimming pool. If the LIDAR does not pick anything up in range, its safe to assume its measuring something greater than the calculated hypotenuse and thus a hazardous drop in elevation to avoid. Increasing the detection range also brings up the issues on decreased sensitivity to changes in the measured distance regarding the calculated stable hypotenuse for whichever angle. If there happened to be a very narrow but steep drop in elevation, the LIDAR could hit the other side of the trench where it ends and comes back up with not much of a change in the hypotenuse depending on the angle used and range of the LIDAR range detection system and the width of the pit.

8.10.3 Map Updating

After a potential collision was detected, the location of the obstacle that would cause the collision was represented as indices for the two-dimensional Locations object array map and the location at those indices on the map was marked with the value nine. Every index adjacent to every obstacle, that does not have the value nine in it, was updated with the number of obstacles that they were adjacent under hazard level, with the exception of values greater or equal to one million for the sensors. The maximum number of obstacles that can be adjacent to a location was eight while the minimum number of obstacles that can be adjacent to a location was zero, which was why the value nine was used to mark the obstacles like a flag. General obstacles were to be avoided so there was no need for them to be labeled individually like the sensors. The values within the locations adjacent to the obstacles represented the risk of collision with their neighboring obstacles as the autonomous irrigation vehicle would be more likely to collide into an obstacle if it were in a location adjacent to more obstacles.

One method that could be used in coding this would be by brute force with a worst case runtime of O(m * n) where m was the length and n was the width of the two dimensional Locations object array map. Where once everything was placed it scanned the map in a double loop for each dimension and checked at every location that was not marked with a nine or one million up to two million to see if any of the locations around them were equal to nine. Locations with a value greater than or equal to two million will be moved to the next location adjacent to its current location with a value of zero to keep the autonomous irrigation vehicle safe from getting stuck or damaged from obstacles, if there was no adjacent location with a zero value then the location with the lowest value if selected and its adjacent locations were checked for a zero to transfer the midpoint location to. If all adjacent locations have the value of nine, then the one closest to the next midpoint was selected to check for the lowest of its adjacent locations until a zero was found to place the midpoint destination into. For each adjacent location for nine, a counter that's been initialized at zero increments its value and once all adjacent locations have been checked the current location was marked with the value of the count. The counter then reset to zero and the next location was checked.

Alternatively, you could also check all eight of the adjacent locations of each obstacle while marking the location of an obstacle. All it would need to do was check indices that were minus one or plus one to the x or y index values to see if they were within bounds of the two dimensional Locations object array map and to increment their stored hazard level values by one if they were. This would significantly reduce the runtime because rather than scanning the whole map to find where the locations were all over again, which in itself would have the O(m * n) runtime as mentioned above. The new runtime would only be O(8) which can be simplified to O(1), also known as a constant runtime. This runtime was ideal because the autonomous irrigation vehicle would spend much less time and resources updating its map that could be used towards other critical computations and processes. The treatment of sensor locations and the movement of midpoint locations would remain the same as mentioned before. In Figure 38, we see how the map would behave in a hypothetical situation.

0	0	0	0	0	0	0	1	2	2	1	0	0	0	0	0	0	1	2	3	3	3	2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	2	9	9	2	0	0	0	0	0	0	1	9	9	9	9	9	1	0	0	0	0	0	0
0	0	0	0	0	0	0	2	9	9	2	0	0	0	0	0	0	1	2	3	3	3	2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1m	0	0	0	0	2m	0	0	0	0	1m	0	0	0	0	2m	0	0	0	0	0	1m	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2m	0	0	0	1	1	1	0	0	0	2m	0	0	0	0	0	0	0	0	0	0	2m	0	0	0	0
0	0	0	0	0	0	0	0	1	9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	3	9	2	0	0	0	0	0	0	0
0	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0	0	1	9	9	9	9	2	0	0	0	0	0	0	0
0	0	0	0	0	0	2	9	9	9	2	2m	0	0	0	0	0	1	2	3	3	2	1	0	0	0	0	0	0	0
0	0	1	1	1m	0	3	9	8	9	3	0	0	0	1m	0	0	0	0	2m	0	0	0	0	0	1m	1	2	2	1
0	1	3	9	2	0	3	9	9	9	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	9	9	2
0	2	9	9	2	0	3	9	6	3	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	3	9	9	3
0	2	9	3	1	0	2	9	9	1	0	0	0	1	9	1	0	0	0	0	0	0	0	0	0	0	2	9	9	2
0	1	1	1	0	0	1	2	2	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	2	2	1

Figure 36: (30 x 20)ft^2 Collision Obstacles & Adjacent Locations

The updated map would be saved for reuse for any future runs, granted there were no changes in the sensor number, sensor locations, or any new obstacles introduced to the area. If there were changes involving the number of sensors or their positions, the map will be reinitialized with the information of those changes received from the mesh network as if it was a new area to be watered. The reason for this was the sensors were stationary and any drastic changes to them would most likely be that the autonomous irrigation vehicle has been brought to a new location to do its job there. The map will only update its current map when there was a change in the area concerning the number of obstacles or where the obstacles were located as detected from the computer vision obstacle detection system, in such a case the current map will update the position of that obstacle and its adjacent positions all at once with the exception of the values one million up to two million. For values locations with two million and above, the midpoint values were shifted like before to locations with the value of zero that were adjacent to their previous location in a cascading fashion.

The map stores the value nine at the locations of all the obstacles with all of their adjacent locations marked with the count of obstacles they were adjacent to under hazard level with the exception of sensors. As seen were obstacles of various sizes to demonstrate the values in the adjacent location.

8.10.4 Scan Routines

The presence of stationary obstacles may be dynamic in that someone may leave an object such as a bicycle on a lawn one day and gone the next. This could cause a hazardous collision that the obstacle avoidance system on the autonomous irrigation vehicle would detect and mark on the two-dimensional Locations object array map to steer clear of on its traversal of the area proving water to where it's needed. When the temporary object is no longer present this area would once again be safe travel through and maybe even favorable at times to travel though to provide water in the most efficient and optimal fashion possible. On the next traversal through the area the autonomous irrigation vehicle should detect this change and remark its location on the map to be all clear, but for the user's benefit it may be safe to include in possible future development route scanning routines to map out an area for hazards on a separate run where it is not supplying water to any location but only scanning for obstacles. This could be useful for mapping out areas the autonomous irrigation vehicle has not visited to map out before or to reset its mapping of the area by the user due to changes in the hazards and land scape of the area or for any other reason that they may so choose.

The sensors and their neighbors should still be provided by the wireless sensor mesh network where midpoint destinations could still be calculated and plotted onto the two-dimensional Locations object array map. The autonomous irrigation vehicle could be provided a list of all the midpoint destinations in the area and build a path to reach every single one despite dryness readings as no water would be supplied during this traversal. It could follow the list and sweep the area in a grid like fashion, which would be thorough but not necessarily optimal are far as distance traveled and energy consumption. The autonomous irrigation vehicle could use more advanced pathfinding algorithms such as the nearest neighbor algorithm to reduce travel distance and energy consumption, but there was still also the concern of the tether connecting the autonomous irrigation vehicle to its water supply source getting caught obstacles or tangle with its self as well as the issue of it having to backtrack back to its charging station in the reverse of its course for the sake of preventing either of those stated issues from occurring when its finished with its scan routine. However, because it's not proving water to the area during scan routines, the autonomous irrigation vehicle does not necessarily need to be attached to its water supply tether when all it is currently doing is scanning for obstacles. By freeing the autonomous irrigation vehicle from the constraints of the water supply tether, algorithms more advanced than the nearest neighbor algorithm could be used, such as the repetitive nearest neighbor algorithm of a markov decision process.

The protocol for the scan routines could be initiated by a reset signal that future development could include, that is sent to the autonomous irrigation vehicle directly from the user. Another option could be a timer that the user could set up to send the signal from at designated times that they may decide for whatever reason on a weekly, monthly, or any other period. Either the direct input or the timer could be directly accessible on the autonomous irrigation vehicle, the wireless sensor mesh network, or from a mobile application that could be developed to give the user more input in a convenient manner.

8.11 Restricted Zones

There may be regions in which it would be unfavorable for the autonomous irrigation vehicle to traverse or provide water to as it could prove to be unnecessary, hazardous, or wasteful. While the obstacle avoidance system should protect the autonomous irrigation vehicle from colliding into stationary obstacles, there may still be some additional factors to consider. This may include things too small for it to recognize such as flowerbeds that could be trampled or more dynamic regions such as a busy high way where cars are coming and going. Driveways and sidewalks may also need to be included as providing water to these regions would be wasteful, which goes against the goal of the autonomous irrigation vehicle's purpose. There could be even more regions the consumer may want to consider further down in development based on currently unforeseen factors the market may demand.

This could be achieved by possible future development to provide additional geographical input to label various areas as restricted zones on the two-dimensional Locations object array map. The values from ten up to nine hundred ninety-nine thousand nine hundred ninety-nine were left alone during the initial potting of sensors, midpoints, obstacles, and locations adjacent to obstacles for later use if additional flag types are needed. Restricted zones could be labeled with a single integer value telling the autonomous irrigation vehicle not to enter those regions that could be the same as the obstacles with adjacent locations marked with hazard risk values in the same fashion. Alternatively, restricted zones could be categorized into various classes to be handled in specific ways using the vast pool of available integer flags. One type of restricted zone could have full absolute restrictions like an obstacle to be avoided like a busy road, another could have water restrictions where it could still travel but must not dispense any water to prevent waste such as a sidewalk, there could be a zone designation for travel restrictions where water would be provided but the autonomous irrigation vehicle cannot enter like in the case with flower beds that could be trampled over, or anything else the consumer may ask. Different classes of restricted zones could have different unique integer flags to distinguish them from one another. A digit-based flag method like the ones used for sensors and midpoints if specific restricted zones may have an id value.

The restricted zones could be labeled on the two-dimensional Locations object array map through a variety of methods. One such method could be through manual input from the user via a mobile application where they could select precisely which regions to mark as restricted zones or type of restricted zone if that is an option. Images from an online source, such as google earth or something similar, could also be used to supply info for labeling or categorization of various restricted zones and any necessary data related to them. Another option could be to supplement or replace the LIDAR system with an even more advanced computer vision system which could utilize machine learning with convolutional neural networks to classify and identify objects via specific visual input from

a camera to determine restricted regions to be marked on the two-dimensional Locations object array map. These three example options mentioned will be further explored in fuller detail in later sections.

8.11.1 Camera

A future modification could be attach a camera to the autonomous irrigation vehicle to provide live visual input for it to make decision for improved obstacle detection and restricted zone categorization that could supplement if not replace the current LIDAR system. This would greatly improve the precision and allow additional functionality to the autonomous irrigation vehicle. This could be achieved using more advanced computer vision methods such as object recognition. Another method that could be used is to somehow incorporate using perceptrons, more specifically sigmoid perceptrons as seen in Figure 39.

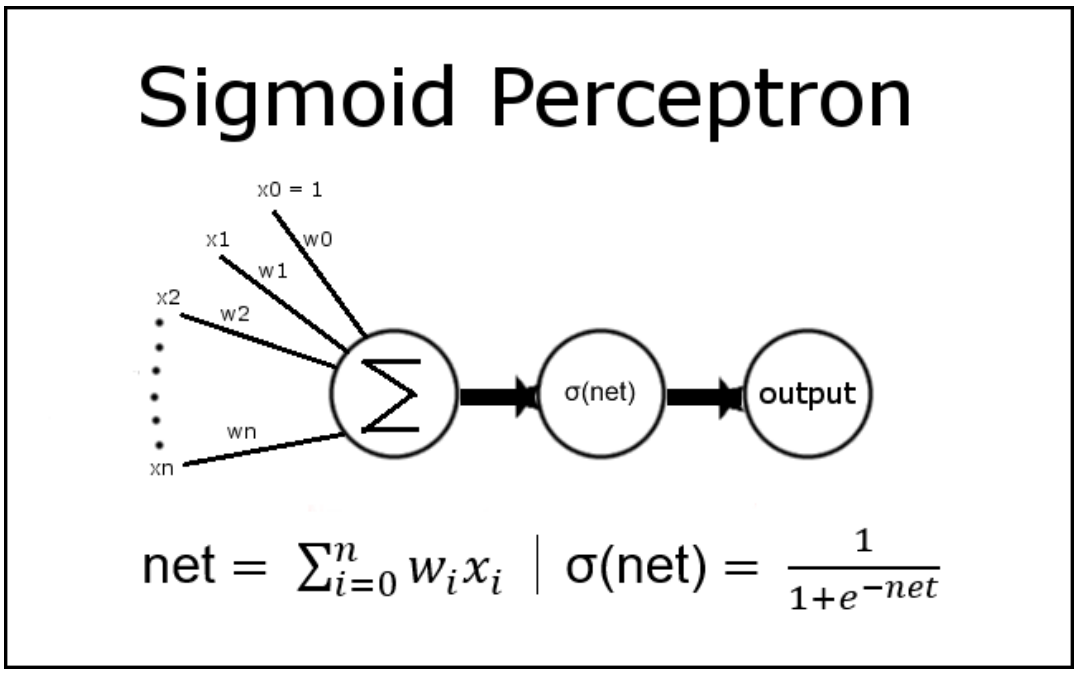


Figure 37: Sigmoid Perceptron

Object detection can be achieved using machine learning with convolutional neural network of a few layers. Machine learning uses training data that could either be proved or collected while performing tasks to improve the performance of the algorithms for certain tasks. Convolutional neural networks are a form of artificial neural networks made up of a collection of artificial neurons such as perceptrons that use a linear classifier to fire a signal when it receives a specific input its trained to recognize based on a weighted value, data points, and its classifier function. The training data used for visual input is usually a set of positive images and a set of negative images to give the system conditions to look for in similar features of the object its trained. Logistic regression is able to adjust

their weights and, in many cases, preferable over a threshold function used for classification for better accuracy, perceptrons that use a logistic to classify are called sigmoid perceptrons.

The goal of artificial neural networks is to build a machine learning system inspired by arguably the most effective and advanced learning system known to the world, the human brain. Convolutional neural networks are built from layers of sigmoid perceptrons stacked together where the outputs of previous layers are used as inputs for each following layer until they reach a final output and use gradient decent to train their networks as pictured in Figure 40. Each sigmoid perceptron can classify things that are linearly separable, but for things that are not linearly separable, networks of perceptrons are needed to perform nonlinear regression. Convolutional neural networks have other applications aside from classification to identify objects, they are also useful for information retrieval, image captioning, and detection.

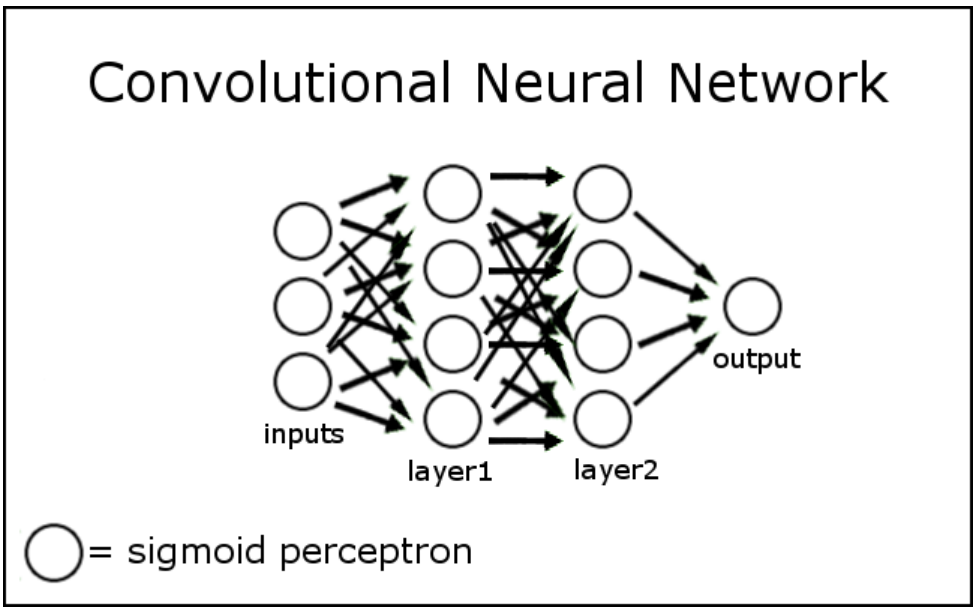


Figure 38: Convolutional Neural Network with Sigmoid Perceptrons

Giving the autonomous irrigation vehicle the ability to receive and process visual input and the ability to identify objects would allow for it to make better decisions and provide more functionality than with just the range detecting LIDAR. With vision capabilities, the autonomous irrigation vehicle could detect gardens or flower beds to provide specialized services for their optimal watering needs, which could also hold significant promise for the agriculture industry as a market. The autonomous irrigation vehicle would also be able to see and identify roads and sidewalks and know to keep water out of those areas as well as avoid moving hazards in real time. The method offers much greater precision and would complement well with a robotic arm with a sprinkler built in with multiple specialized nozzles for various tasks attached to the autonomous irrigation vehicle to vastly improve water delivery capabilities with significant efficiency increase for water conservation.

8.12 Backtracking

When the autonomous irrigation vehicle completes its course through an area, it needs to be able to return to its home charging station to recharge its battery for the next time it is used to provide water to areas that need it once the water supply tether is implemented by the team of mechanical engineers. It also would need to reel it tether back up so that its not just laying across the region in a mess where its could get caught, tangled, or damaged which would impeded further use of the autonomous irrigation vehicle with the need for maintenance, repairs, or the replacement of parts at the inconvenient expense of the consumer. The primary plan to handle this is for the autonomous irrigation vehicle to save a copy of its list of destinations it previously visited on its water delivery course and traverse them again in reverse order from finish back to start while driving in reverse. If the tether is made to be retractable, it should wind up safely out of harm and without any tangles as the autonomous irrigation vehicle backs up along its reverse backtracking course back to its home charging station to begin again next time it needs to without any issues. With the retractable water supply tether, it would be faster and possibly more energy efficient to have the autonomous irrigation vehicle placed in neutral where the charging station could reel in the entire thing in a straight line directly back to itself. This would risk damaging the autonomous irrigation vehicle as it could run into hazardous obstacles while being reeled back in, thus the backtracking method of it retracing its previous path in reverse with the tether being retracted at the same pace as it travels, with minimal slack and tension, would be the safest method for it to return to its charging station.

A power supply chord could be included with the water supply chord bundled together as a cable within the tether to solve the issue of it having to return to recharge as it would have a constant and reliable source of power. However, bundling the water supply with the power supply introduces new risks, if any damage where to happen to the tether with both the water and power supply there could be the safety hazards to humans and animals of electrocution, this could a e fire hazard to the user's property if some areas in the region are still dry and damages to the power supply chord causes a spark that could ignite those areas, as well as a hazard to the autonomous irrigation vehicle itself if power supply chord damages fry any of the on board electronic systems that it relies upon to carry out its functions. This also does nothing to solve the issue of the supply tether being left sprawled out across the region exposed to the elements in harm's way when the autonomous irrigation vehicle finishes its job. Below in Figure 41, a flowchart depicts the order of processes carried out during this operation.

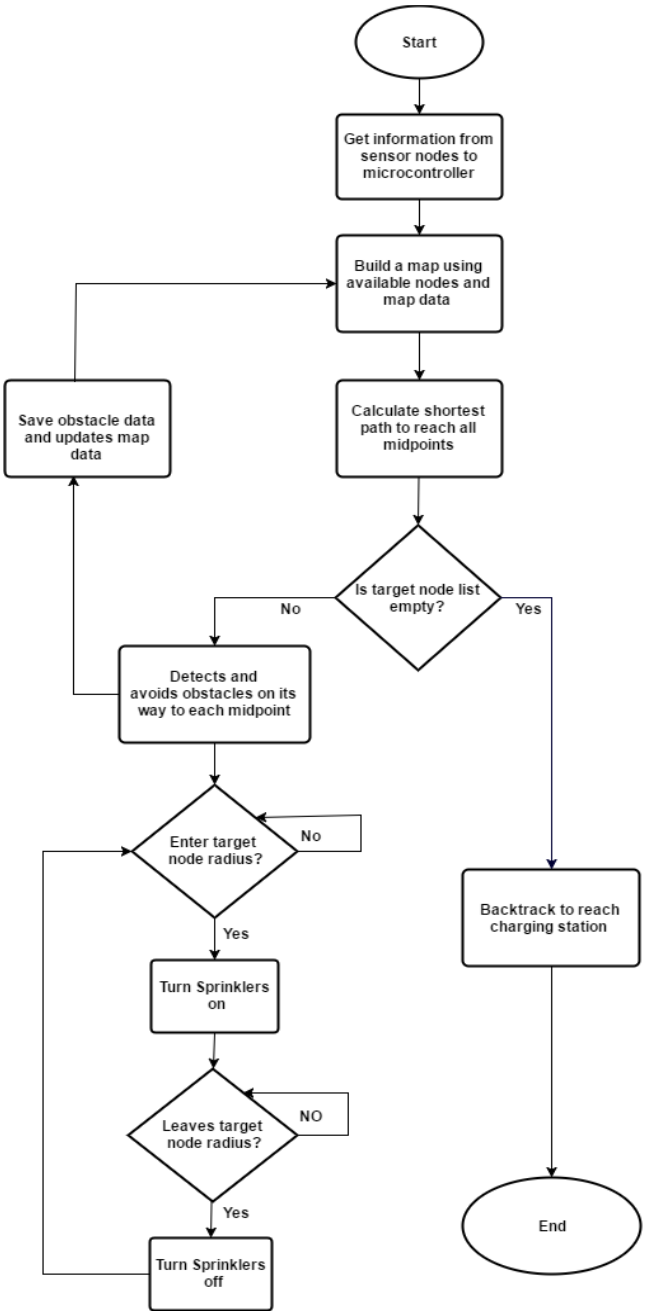


Figure 39: Software Flowchart with Backtracking

Alternatively, another possible future development could be for the autonomous irrigation vehicle could disconnect from its retractable water supply tether to return to its charging station. The tether could be reeled in back to the charging station and the autonomous irrigation vehicle could calculate a much simpler and more streamlined path back to its charging station from its current location rather than having to backtrack through the full extent of its course in reverse order without the restrictions of being bound to its water supply tether that could get caught or tangled by itself or obstacles in the area. Once the disconnected free roaming autonomous irrigation vehicle returns to its charging station it

could reattach to its water supply tether for the next time it needs to deliver water to every single midpoint destination location with at least one dry sensor reading when it is needed.

8.13 Unrestricted Movement

The autonomous irrigation vehicle would have more efficient routes to reach all the midpoint destinations with dry sensor readings within the landscape its responsible for in a potentially shorter distance traveled if it had more freedom for it to move across the area in a less restrictive linear path. A purely linear path may not be the optimal path to travel, but due to current constraints of the design with the water supply tether attached to the autonomous irrigation vehicle, some kind of linear path is inevitable. This linear path constraint restricts the types of pathfinding algorithms that can be used to find paths for the autonomous irrigation vehicle to build a list of which midpoint destinations with sensors that give dry readings at their borders to visit in whichever order is the most optimal.

In the sections ahead, potential future modifications will be discussed to help solve or improve this issue that restricts the autonomous irrigation vehicle from finding the best path that it can to reach all the dry regions of the area that need water delivered to them with improved movement freedom in which better pathfinding algorithms can be used. These modifications include the options of a detachable and re-attachable water supply tether that is also retractable to reduce risks of entanglement and backtracking the path from where it ends back to the charging station in favor of a shorter path. The option of an onboard water reservoir tank will also be discussed to elaborate on its potential for breaking free from the linear path constraints as well as determine its feasibility as a design to be implemented. A modified form of the detachable and re-attachable water supply tether that is also retractable with the addition of multiple charging stations at different locations for the autonomous irrigation vehicle to attach from will also be discussed for its improved pathfinding capabilities, its improved maneuverability around more complex obstacles in oddly shaped yards, and its scalability to where a single autonomous irrigation vehicle could provide watering services across many lawns within a neighborhood subdivision.

The implications of how these modifications to the design will allow for stronger algorithms that outperform the current algorithms being used for finding the best path the bring water to dry regions of the lawn determined by the sensors within the wireless mesh network in the fastest and safest manner possible will also be discussed. From options that allow minor algorithm modifications for marginal improvement, to others that would allow for complete freedom of movement across the entire area without physically being bound to the location of the charging station where it would no longer be restricted to linear paths and thus able to use more optimal algorithms to find the shortest and safest route to reach all the water delivery locations, to a more feasible option that grants more freedom of movement within the restrictive bounds of the linear path constraints. An improved algorithm that takes advantage of the additional factors that come to play with less

restricted movement to find the order of which midpoint destinations with at least one dry sensor reading to travel to in the least distance traveled would reduce the power cost of the autonomous irrigation vehicle to travel, especially when scaled up to larger areas that would require for it to travel across to perform its duties.

8.13.1 Tether Disconnect

With the water supply tether, the autonomous irrigation vehicle is restricted to follow a more linear path which may not be optimal to reach every single one of its locations in the fastest and safest manner possible. There are also the risks of the autonomous irrigation vehicle's water supply tether getting caught on an obstacle or tangled which would impede the movement of the autonomous irrigation vehicle. Additionally, if the water supply tether were to be damaged by something while the autonomous irrigation vehicle is providing water to dryer areas of the landscape that need it, it would be out of commission and unable to continue its water distribution duties until repair as made or the water supply tether is replaced with a new one.

To lessen the risk of the water supply tether getting caught on obstacles or tangled during the autonomous irrigation vehicle's travels to perform its duties or when it is backtracking back to its charging station to recharge its battery when it is finished providing water to everywhere that needs it, the water supply tether will be retractable, similar to a dog leash as seen in Figure 42. Future developers could have it release from the charging station at the same rate at which the autonomous irrigation vehicle travels and reels back up at the same rate too so that there is not too slack but also not too much tension. This is something that is being implemented in this project to help, but it helps with some of the issues, it's still far from perfect as it still restricts the autonomous irrigation vehicle's pathfinding to a possibly less than optimal linear path for it to take and if the water supply tether still receives damage while it is in the middle of a path, the autonomous irrigation vehicles would still be out of commission.

An improvement to this solution later developers could work on would to also make the water supply tether detachable from the autonomous irrigation vehicle and re-attachable at the charging station. This would allow for the autonomous irrigation vehicle to detach from the water supply tether if it were to get caught or tangled on any obstacles or itself where the detached water supply tether could be reeled in back to the charging station and the autonomous irrigation vehicle could plot the shortest and safest course possible from its current location to the charging station where it would re-attach to the retractable water supply tether once it is back in place. Right before the retractable water supply tether detaches, the source of water that goes to it would shuts a valve closed to cut off the flow of water through it to prevent excessive amounts of water waste like when a hose is left on unattended flooding an area.

Figure 40: Retractable Dog Leash like the Retractable Tether

Having additional attachable and detachable water supply tethers that are also retractable at the charging station as backups. This would be extremely useful in cases where the previous detached retractable water supply tether is unable to reattach to the autonomous irrigation vehicle. There is a wide variety of reasons as to why this could occur while in the middle of proving water delivery services. This would allow for the autonomous irrigation vehicle to go back to what it was doing before it had to detach the retractable water supply tether until its issues are resolved, whether that be manually freeing it from being stuck, repairing it, or replacing it.

One possible reason could be that the previous retractable water supply tether was damaged while the autonomous irrigation vehicle was providing was to dry regions of the landscape that needed it and had to detach from it to get a new one to complete its job. The damage to the retractable water supply tether could be detected by a water pressure sensor in the charging station that could send a signal to let the charging station and autonomous irrigation vehicle know that the retractable water supply tether had been damaged from a change in water pressure cause by a leak, obstruction, or any other kind of damage it could have received. This would let the autonomous irrigation vehicle know to detach the retractable water supply tether and both charging station and the autonomous irrigation vehicle know not to re-attach to the damaged retractable water supply tether and instead to use one of the backup retractable water supply tether to attach to the autonomous irrigation vehicle to use to complete its course and deliver water to all the remaining dry areas that need it.

Another reason why having multiple retractable water supply tether backups could be useful would be a case where a detached retractable water supply tether could not be reeled back in to the charging stations because either it had gotten stuck on something or it got tangled with itself. There could be a sensor in the charging station by where the

retractable water supply tether is connected to that could give a signal that the other end of the retractable water supply tether that connects to the autonomous irrigation vehicle has returned by having something inside it for it to detect when it is near and in place to be reattached. If the autonomous irrigation vehicle does not receive the signal that the retractable water supply tether is in place to be reattached, it will attach to the next available backup retractable water supply tether for it to use for the remainder of its water delivery services.

8.13.2 Water Reservoir

The retractable water supply tether will always restrict the autonomous irrigation vehicle's traversal to linear paths if the retractable water supply is attached to the autonomous irrigation vehicle while it delivers water to all the places of the landscape where it is needed. While the retractable water supply tether may be a very reliable water delivery system to provide the autonomous irrigation vehicle with a supply of water needed to fulfil its duties, the restricted linear movement it enforces could prevent a more optimal path from being chosen by more advanced pathfinding algorithms. The only way for the autonomous irrigation vehicle to truly have unrestricted movement while it is driving around the terrain providing water to the dry areas, it will need to replace the retractable water supply tether with something less restricting that can still supply it with water untethered.

The obvious choice here would appear to be an onboard reservoir capable of storing enough water to provide to all the dry areas that need water. The autonomous irrigation vehicle would fill this on-board water reservoir tank at its charging station to use as a water source to deliver water to all the dry areas of the region that the sensors in the wireless mesh network indicates needs water. When the on-board water reservoir tank runs out of water, the autonomous irrigation vehicle would return to the charging station to refill its water supply before resuming its course providing water to wherever it is needed throughout the landscape guided by the reading of the sensors in the wireless mesh network to recalculate a new optimal path that factors out all the regions previously watered before its water supply ran out.

In a worst-case scenario such as a hot sunny day, a yard would require one and a half inches of water per square foot. There are twelve inches in a foot, so this amount could be converted to cubic inches by squaring twelve inches and multiplying it by one and a half inches which results in two hundred sixteen cubic inches. There are two hundred thirty-one cubic inches in a gallon so two hundred sixteen cubic inches divided by two hundred thirty-one cubic inches per gallon would result in nine hundred thirty-five thousandths of a gallon per square foot, which could be rounded up to one gallon of water per square foot.

For a testing area of 20X30 squared feet, which is six hundred square feet, so to provide all this area on a hot and dry day with enough water a six-hundred-gallon tank would be

needed. One gallon of water weighs eight and thirty-three hundredths pounds so with the six-hundred-gallon tank the autonomous irrigation vehicle would need to carry a load of four thousand nine hundred ninety-eight pounds. This is obviously not being a feasible option for an autonomous irrigation vehicle that measures two feet by two feet and three feet off the ground. Dividing the load to carry five or ten gallons of water could still be strenuous on the autonomous irrigation vehicle and require a lot of power as well as one hundred twenty or sixty trips back to the charging station, depending on water reservoir tank capacity, to refill its water reservoir tank throughout its travels to water all the dry parts of the landscape, which is still not feasible.

8.13.3 Multiple Stations

Since there were concerns of the movement of the autonomous irrigation vehicle, a review for methods to increase unrestricted movement of the autonomous irrigation vehicle was under taken. With the autonomous irrigation vehicle having a water reservoir supply tank equipped to the autonomous irrigation vehicle would be too impractical in practice to implement. The other option is having a retractable water supply tether that is both attachable and detachable from the autonomous irrigation vehicle. The tethered water supply also has an option where the autonomous irrigation vehicle can dock at the charging station to reattach its retractable water supply tether.

This consideration has shown a lot of promise, but this still has limitations. This has led to the idea, other developers could later implement, of having multiple charging stations at different locations in the region for the autonomous irrigation vehicle to charge its battery and to attach to a retractable water supply tether. As such, the autonomous irrigation vehicle will be given even greater unrestricted movement throughout the environment in which it is surrounded in. With the autonomous irrigation vehicle moving through its surrounding environment almost unhindered, it can reach any of its predesignated locations with ease. Once it reaches it designated destination of where the sensors have told the autonomous irrigation vehicle is dry, the watering operation commences. Figure 43 shows an example of what is being explained with a clear picture.

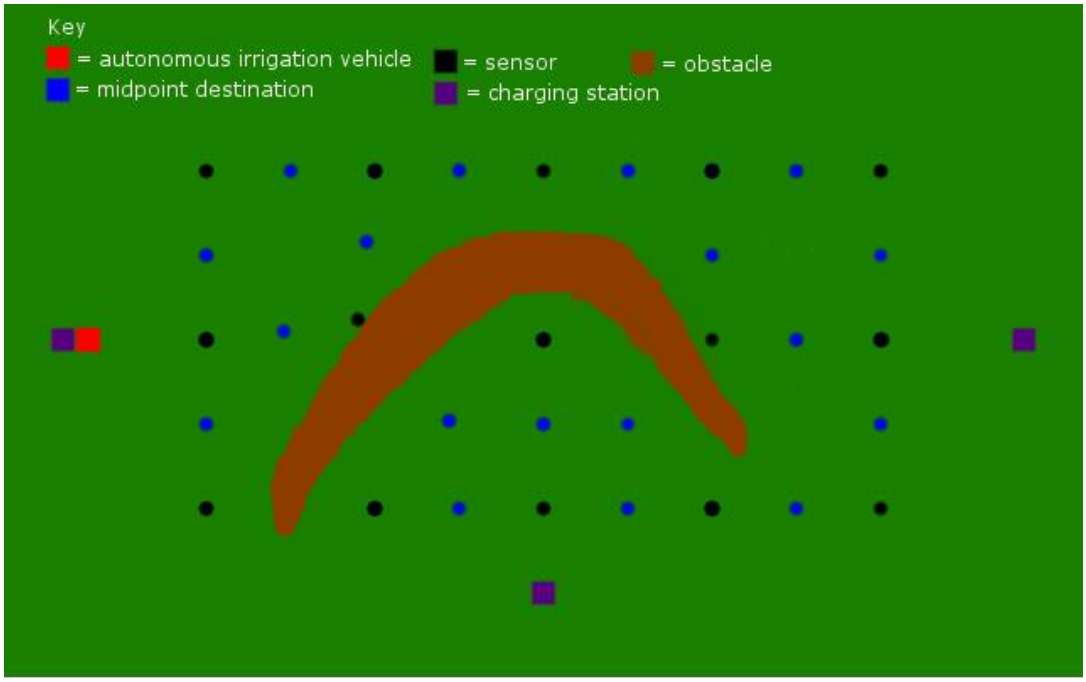


Figure 41: Multiple Stations in Larger Area with Obstacles

From different locations, the autonomous irrigation vehicle may be able to reach various midpoint locations that need water with better ease than it could if restricted to only one starting location due to the limitations of linear paths. Many landscapes may be broken apart by large obstacles such as island bed gardens that can take up large areas in odd shapes. For these cases, if the autonomous irrigation vehicle were to follow a truly linear from one starting location to reach all the dry regions to provide all of them with water and travel back to the one charging station would be very long and the risks of the retractable water supply tether getting caught on something or tangled are more likely, by having multiple charging stations to grab a retractable water supply tether to reach around larger obstacles would significantly reduce this risk.

Having multiple charging and tether stations for one autonomous irrigation vehicle could be expanded to cover larger areas than go beyond where a single retractable water supply tether could reach. An example being multiple lawns within a subdivision where the autonomous irrigation vehicle could dock and attach to retractable water supply tethers from charging stations in different yards. How the autonomous irrigation vehicle deals with sidewalks, driveways, and roads while providing water to dry areas across multiple different yards within the same neighborhood would depend on how it is set to handle restricted zones. All the sensors used across all the lawns within the same subdivision could all be with in the same wireless mesh network which could allow for the autonomous irrigation vehicle to plot a course to visit all their midpoint destinations that need water in a single planned path. The different charging stations could also be marked on the two-dimensional Locations object array map with their own unique integer flag value to identify them so that the autonomous irrigation vehicle can include them in their path if it is optimal to do so to reach every destination midpoint with dry sensor readings

than previously. The depiction of the autonomous irrigation vehicle covering multiple homes can be seen in Figure 44.

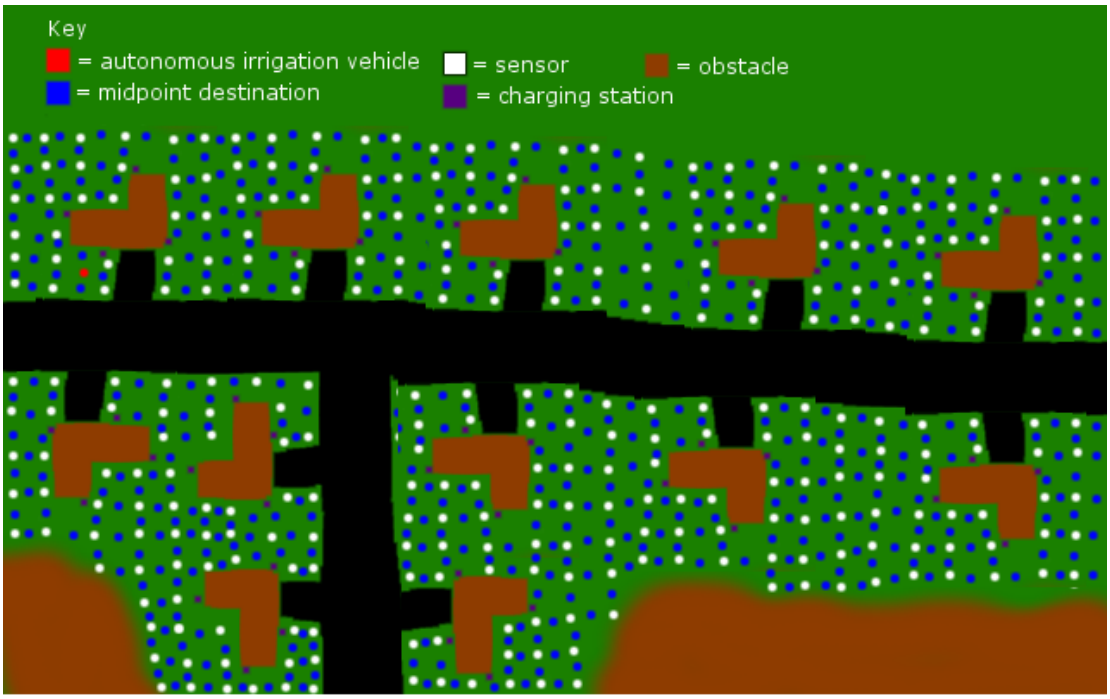


Figure 42: Multiple stations & 1 Autonomous Irrigation Vehicle.

This method seems more feasible, more efficient, and more scalable than the current limited linear implementation or other alternatives to the mentioned. It is still somewhat restricted to linear paths, but the larger linear path can be broken down into smaller linear paths to allow for some freedom of movement for better paths. It is not perfectly unrestricted movement, but it is close and removes many of the risks and inefficiencies of having a single retractable water supply tether as well as allows for more advance pathfinding algorithms.

8.13.4 Alternative Algorithms

With more freedom of movement, more advanced algorithms could be used by other developers later on to find better optimized paths for the autonomous irrigation vehicle to provide precious life sustaining water to all the dehydrated regions of a lawn. The retractable water supply tether carries the risks of getting caught on something or tangled and constrains the autonomous irrigation vehicle to a linear path that may not be as efficient as it would if it had a little bit more freedom with its movements. The autonomous irrigation vehicle will be using a nearest neighbor algorithm to build its path to reach every

single midpoint destination with at least one dry reading sensor at its edges to deliver water until there are no more dry sensor readings in the landscape.

With the retractable water supply tethers that can detach from the autonomous irrigation vehicle and reattach to another one when docked at a charging station, the autonomous irrigation vehicle would still be restricted to a linear path. However, the detachable and re-attachable water supply tether that is retractable would eliminate the risks of entrapment on obstacles and entanglement with itself by allowing the autonomous irrigation vehicle to release itself to grab a new retractable water supply tether to continue its water delivery route. With this you could set up the algorithm to tell the autonomous irrigation vehicle to detach from the retractable water supply tether if there is a far away midpoint destination with a dry sensor region that would take longer to reach from the current location and backtrack back to the charging station than it would to return to the charging station in the shortest path and head from there to the distant location after re-attaching to a new retractable water supply tether.

A water supply reservoir tank would allow for completely unrestricted movement of the autonomous irrigation vehicle with the absence of the retractable water supply tether is not physically bound to the charging station. This would allow for algorithms that can out-perform our nearest neighbor algorithm, such as the repetitive nearest neighbor algorithm which performs the nearest neighbor algorithm with every location the needs to be traveled to as a potential starting location then compares all the generated paths where it selects the most travel cost efficient path. The main issue here however is that using an onboard water supply reservoir tank is completely impractical due to weight constraints of using a larger tank and the travel costs of using smaller tanks that would require frequent refill trips.

The option of using detachable and re-attachable water supply tethers that are retractable is the most promising option for something closer to unrestricted movement. This method potentially breaks apart the long linear path with a single source that the autonomous irrigation vehicle travels into multiple smaller linear paths with multiple different sources if it is more optimal to do so. With this setup a weaker repetitive nearest neighbor algorithm that could still out-perform the nearest neighbor algorithm by using any of the different charging stations, where the autonomous irrigation vehicle could attach to a retractable water supply tether, as potential starting locations to reach all the midpoint destinations with dry sensor readings.

9. Final Coding Plan

The path finding algorithm and computer vision obstacle detection system was used to achieve the autonomous irrigation vehicle project goal. The nearest neighbor algorithm was used for the path finding concepts and RPLidar A1M8 model was used to detect the

obstacle. Once it detected the obstacles it constantly updated the obstacle location into two dimensional array map, the detail explanation of obstacle detection was given in Obstacle Avoidance and Collision detection section. Also, the detail process of nearest neighbor algorithm was given in the algorithm concept section. The process of coding plan is shown Figure 45 to show the coding plan

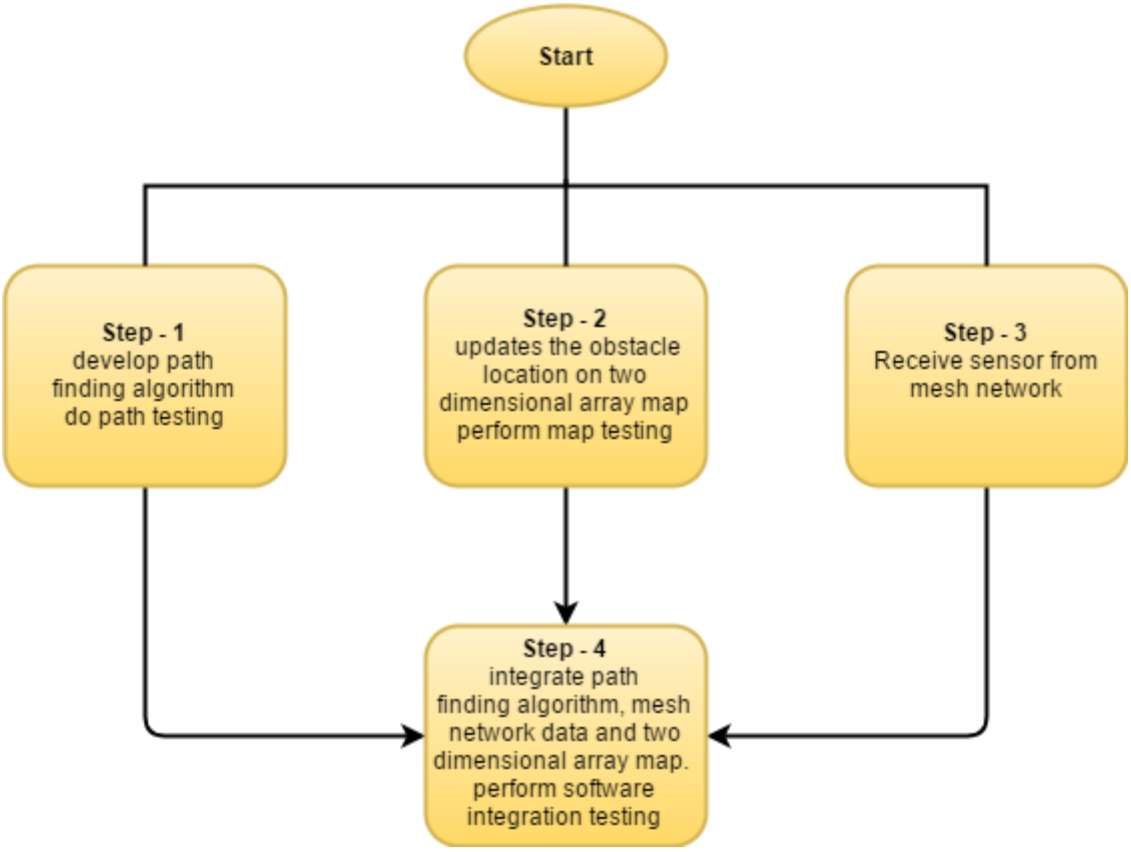


Figure 43: The Process of Coding Plan

10. Software Test Environment

Testing was a significant part of software development. All the testing phases were done effectively to ensure the quality of the autonomous irrigation vehicle. The test cases were written for each module to test the functionality of the path finding algorithm and obstacle detection two-dimensional array map. The visual studio integrated development environment were used as the software testing environment. The visual studio integrated development environment had an available unit test integrated framework. This IDE allowed for the testing of public class method using test driver stubs. When the autonomous irrigation vehicle was build, it was tested in outside environment as well.

The test plan provided a guide for the verification of the requirement of the autonomous irrigation vehicle. The test plan helped to ensure that all the functionality gets properly tested.

- The autonomous irrigation vehicle receive the sensor data from mesh network
- The software shall be able to calculate the nearest midpoint using NNA algorithm.
- The software shall be able to detect the obstacles and avoid collision
- The software shall be able to update the two dimensional matrix array every time it travel to the midpoints.
- The software shall be able to get a feedback from the sensor about is it still dry or is it wet

10.1 Unit Testing

This section explains the unit testing part of the autonomous irrigation vehicle. The main goal of the unit testing was to find the faults in each module. Unit testing was done for each module separately, when one modules was done coding it was tested it by checking the syntax error, inaccuracy of the algorithm or inconsistencies. By performing unit testing for each module, increased the efficiency of the program as well as it prevented future faults. Besides, the two types code review was conducted to find faults. Followed by the unit testing there were several testings was done to verify the autonomous irrigation vehicle quality and functions. Figure 47 shows the testing flow chart which shows the sequence of the testing. In the next sections each testing process was explained in detail.

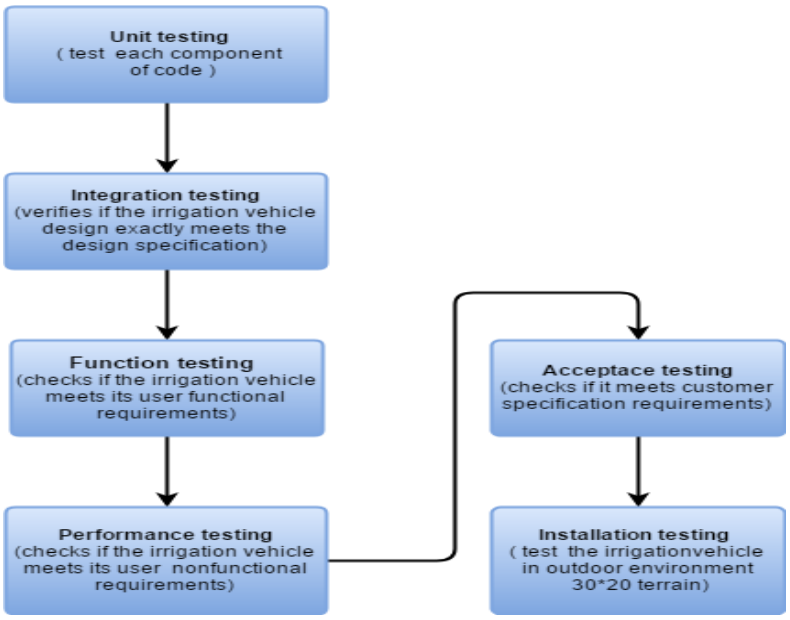


Figure 44: Testing Flowchart

10.2 Performance Testing

Testing was a significant part of software development. All the testing phases was done effectively to ensure the quality of the autonomous irrigation vehicle. The test cases will written for each modules to test the functionality of the path finding algorithm and obstacle detection two-dimensional array map. The visual studio integrated development environment was used as the software testing environment. The visual studio integrated development environment had an available unit test integrated framework. This IDE allows for the testing of public class method using test driver stubs. When the autonomous irrigation vehicle was build, it was tested in outside environment as well. The test plan will provide a guide for the verification of the requirement of the autonomous irrigation vehicle. The test plan helped to ensure that all the functionality gets properly tested.

- The autonomous irrigation vehicle should receive the sensor data from mesh network
- The software shall be able to calculate the nearest midpoint using NNA algorithm.
- The software shall be able to detect the obstacles and avoid collision
- The software shall be able to update the two dimensional matrix

array every time it travel to the midpoints.

- The software shall be able to get a feedback from the sensor about is it still dry or is it wet

This sections explains the performance testing of the autonomous irrigation vehicle. The performance testing checked the nonfunctional requirements of the autonomous irrigation vehicle. Usually the system's non functional requirements were given by the customer, for irrigation vehicle project our sponsor was the customer so the vehicle's nonfunctional requirements are set by our sponsor. The performance testing was inspected how efficiently the algorithm worked and its quick response to navigate to the destination. There were several performance testing was done to validate the nonfunctional requirement of the system. The following performance test was done to validate the system.

- Stress Test
- Volume Test
- Configuration test
- Compatibility tests
- Regression test
- Timing test
- security test
- Quality test
- Recovery test

The tests listed above were different types of performance testing but we did not use all these test to check the autonomous vehicle requirements but we tested only the suitable tests to validate the our system requirements. The stress test analysis how the system would perform if a short period of time it reaches it limits. The autonomous irrigation vehicle project used maximum of four sensor to water the terrain. So, when the system received sensor data from mesh network the algorithm should work efficiently to reach all target sensor in a possible time. so the stress test will be evaluating the vehicle performance when it operated with maximum number of sensors. Furthermore, the volume test evaluated the performance of the autonomous irrigation system when it handled a large amount of data. The configuration test was performed to evaluate if the autonomous vehicles hardware and software configured correctly. Configuring the vehicle's hardware and software requirement helped the vehicle to achive its goal. Since the autonomous irrigation vehicle had to interact with mesh network to retrieve the sensors data, the compatibility test was performed to check if one system autonomous vehicle could interacted with another component of the system as per the requirements. The timing test was performed to analyze if the vehicle performed its functions as per the requirements. The environmental test was done to evaluate how well the system perform in its actually work environment place. The main requirements of the autonomous irrigation vehicle was that when it reaches the target destination midpoint, it dispersed the accurate amount of water to the grass. So we done with all the functions of the irrigation vehicle, we performed the environmental test to make sure that it deliver the right amount

of water and not over watering the grass as well as the LIDAR staied away from the water. The quality test was performed to examine the reliability, maintainability and availability of the autonomous irrigation vehicle. The following sections explains the performance testing of the of specific components of the autonomous irrigation vehicle.

10.2.1 Pathfinding Testing

The section explain the pathfinding testing of the autonomous irrigation vehicle. The vehicle performed the following functions to validate the pathfinding testing.

- The autonomous irrigation vehicle should be able to receive the sensors data from the mesh network
- The autonomous irrigation vehicle shall be able to access the sensor locations from the two dimensional Locations object array map.
- The vehicle shall be able to identify the midpoint location between the dry sensors on two dimensional Locations object array map
- The autonomous irrigation vehicle shall be able to spot the obstacle location on the map
- The vehicle shall be able to calculate the shortest path to reach the destination midpoints
- Once it visited all the destination point, the vehicle should be able to backtrack to reach the home station (charging station).
- If the autonomous irrigation vehicle run out of battery on its way to destination it should able to back track to reach the charging station

The autonomous irrigation vehicle should be able to satisfy all these above requirements to attain its goal. The performance testing will be performance to validate the autonomous irrigation vehicle met all the required functionality mentioned above. Especially this testing will check the speed of the vehicle and the accuracy of the calculation. The following test cases executed to ensure the pathfinding performance.

- Send the maximum number of dry sensors to the autonomous irrigation vehicle to check nearest neighbor algorithm calculation accuracy.
- Check the travel speed of the autonomous irrigation vehicle

These test plans were executed to check the pathfinding algorithm requirements. When the vehicle did not satisfy any of the requirements we followed the Testing failure process

flowchart, which is in software failure process section, to fix the problem and make sure the system performs its function as per the customer requirements.

10.2.2 Obstacle Avoidance Testing

The section explain the obstacle avoidance testing of the autonomous irrigation vehicle.

- The autonomous irrigation vehicle shall be able to detects the obstacles on its way to the destination point
- The autonomous irrigation vehicle shall be able to detects the obstacle in 180 degrees of range.
- he autonomous irrigation vehicle shall be able to detects any objects from 3 feet distance
- The autonomous irrigation vehicle shall be able to recalculate its path if it detects any obstacles
- The autonomous irrigation vehicle should be able to updates the obstacle location on the two dimensional Locations object array map if it finds any obstacle

When the autonomous irrigation vehicle travel to the midpoints, it effortlessly detected obstacles to prevent the vehicle from collision. The performance testing was examined all these mentioned requirement to check the performance of the irrigation vehicle. Also we executed the following step to check the test case for obstacle avoidance.

- First check that the light detection and ranging is mounted at correct angle on the vehicle
- Send some sensor data to the pathfinding algorithm to calculate the shortest path to reach the midpoint
- While the autonomous irrigation vehicle travel to the midpoint, verify that it checks any obstacle from 3 feet distance
- Will monitor the two dimensional Locations object array in the software to checks, if the vehicle updated the obstacle location correctly
- There will an object place on the vehicle path to checks, once the vehicle find the obstacle it recalculate the next shortest path using pathfinding algorithm

These test plans were executed to check the obstacle avoidance requirements. When the vehicle did not satisfy any of the requirements, the testing failure process flowchart was followed to fix the problem and make sure the system performed its function as per the customer requirements.

10.2.3 Mapping Testing

This section explains the two dimensional Locations object array map testing. Here is the requirement of the two dimensional Locations object array map

- The two dimensional Locations object array map is used in the autonomous irrigation vehicle to keep track of the sensors location
- Keep track of all Midpoints locations
- Keep track of all Obstacle locations
- Minimum number of adjacent indices should not be less than zero
- Minimum number of adjacent indices should not be greater than nine

When the autonomous irrigation vehicle travel to the midpoints, it should effortlessly detects obstacles to prevent the vehicle from collision. Once it detects any objects it updates the location in the map. The performance testing will examine all these mentioned requirement to check the mapping functionality of the irrigation vehicle. Also we be executing the following step to check the test case for obstacle avoidance.

- Check if the sensors x and y coordination location added correctly on the two dimensional map
- Run the pathfinding algorithm to calculate the shortest path
- Will monitor the two dimensional Locations object array in the software to check, if the vehicle updated the obstacle location correctly
- Also, double check if the obstacle location's adjacent indices numbers are updated correctly

These test plans will be executed to check the mapping requirements. If the vehicle does not satisfy any of the requirements we will follow the testing failure process flowchart to fix the problem and make sure the system performs its function as per the customer requirements.

10.2.4 Water Allocation Testing

This section explains the water allocation testing. Here is the requirement of the water allocation testing.

- The autonomous irrigation vehicle should be connected to the retractable tether hose before it leaves the home station
- The autonomous irrigation vehicle shall be able to turn on the sprinkler head to disperse water, when it reaches the destination location.
- The autonomous irrigation vehicle shall be able to deliver exact amount of water in the specific area
- The autonomous irrigation vehicle shall be able to turn off the sprinkler, when it receives signal from sensor
- After turned off the sprinkler the autonomous irrigation vehicle should travel to the next destination point.

The crucial function of the autonomous irrigation vehicle is to disperse the correct amount of water in the allocated area. When the autonomous irrigation vehicle travel to the midpoints, it should efficiently disperse the water in the area. The performance testing was examined all these mentioned requirement to validate that the vehicle delivered correct amount of water. Also, the following test cases were executed to verify the obstacle avoidance functionality.

- Check the autonomous irrigation vehicle properly connected the retractable tether hose
- Monitor the program to check if the vehicle turn on the sprinkler at the correct place.
- Check if the sensor send the signal at the right time.
- Monitor the program to check, after it turned off the sprinkler the autonomous irrigation vehicle is traveling to the correct next destination midpoint.

These test plans were executed to check the water allocation requirements. When the vehicle did not satisfy any of the requirements, the testing failure process flowchart was followed to fix the problem and make sure the system performed its function as per the customer requirements.

10.2.5 Backtracking Testing

This section explains the backtracking testing of the autonomous irrigation vehicle.
Here are some of the requirements that the autonomous irrigation vehicle was performed.

- The autonomous irrigation vehicle should be able to backtrack to reach its home station (charging station)
- If the vehicle run out of battery on its way, it shall be able to backtrack to reach its charging station to recharge it.

There will be two ultrasonic sensors attached in the back of the autonomous irrigation vehicle to detects any objects when it back tracking. Since the vehicle is backtracking in the same way it travel to reach its destinations, we assume there should not be any problem to backtrack even if there is any objects on its way, the ultrasonic sensor will detects that and updates the map. The performance testing will examine all these requirements to check the backtracking functionality of the irrigation vehicle. Also, the following test cases will be executed to verify the backtracking functionality.

- Check if the ultrasonic sensor mounted at correct angle on the back of the autonomous irrigation vehicle
- Run the backtracking algorithm to check if its works as per the requirement

These test plans were executed to check the backtracking requirements. When the vehicle did not satisfy any of the requirements, the testing failure process flowchart was followed to fix the problem and make sure the system performed its function as per the customer requirements.

10.2.6 Communication Testing

This section explains the communication testing of the autonomous irrigation vehicle. Here are the some requirements that the autonomous irrigation vehicle was performed.

- The autonomous irrigation vehicle shall be able to communicate with mesh network to receive sensor data
- the autonomous irrigation vehicle shall be able to communicate with the microcontroller raspberry pi

- The Lidar and ultrasonic sensors shall to able to communicate effectively to share data

- The raspberry pi shall be able to communicate with other components over the wireless

These test plans were executed to check the communication testing requirements. When the vehicle did not satisfy any of the requirements, the testing failure process flowchart was followed to fix the problem and make sure the system performed its function as per the customer requirements.

10.3 Software Failure Process

Testing was significant part of the software development. All those different types testing were performed to check the all functional and nonfunctional requirements of the autonomous irrigation vehicle. Although we perfomed all the test, its not guarantee that the test plan will pass all the requirements, few test cases might fail. When the test plan failed we followed the testing failure process flowchart to make sure that the function have been fixed. Testing failure process flowchart shown in Figure 47.

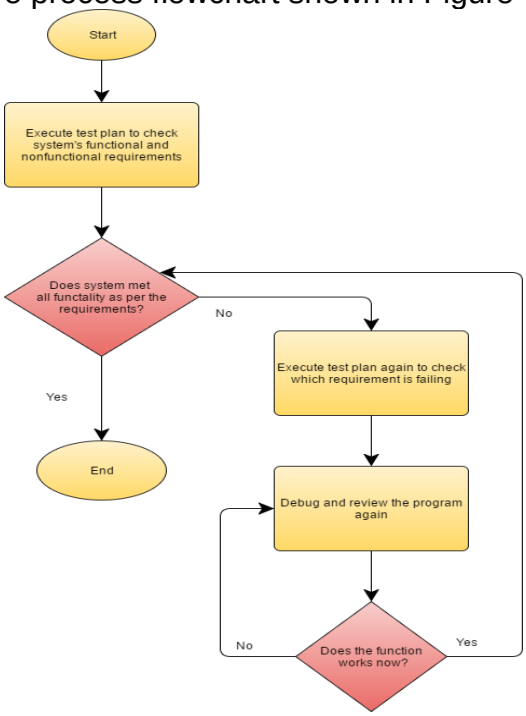


Figure 47: Testing failure process flowchart

In the flowchart at first the test plans were executed to check if the system's functional and nonfunctional requirements are met and is met by as per the customer requirements. When It met all requirements then the system was ready to perform acceptance testing. When it did not meet the requirement then the test plan was executed again to check exactly which requirement was failing. Moreover, the specific component was debugged and reviewed and tested again. When it produced any other error the debug process continued until it got fixed. IN the case when it did not produce any other error then it continued to the acceptance test.

11. Administrative Content

The group has come to a clear and genuine understanding of how each other operates, conduct themselves, and coordinate with one another. This collaboration is needed for the proper and successful execution of the autonomous irrigation vehicle. As such, there have been milestones put in place in order for the group to remain on schedule. Along

with an appropriate scheduling of tasks to complete, there was also been a plan of financing that has been discussed and agreed upon. With all of the aforementioned steps, the group has a clear and steady path to research, document, develop, and finally prototype the autonomous irrigation vehicle.

11.1 Milestone Discussion

Listed below is Table 4 which lays out an approximate estimation of deadlines the team has set out for itself. The purpose of aiming for an approximate deadline prior the actual deadline gives the team time to make final overlooked errors, incorrect data, and deal with extraordinary circumstance that may occur. By taking into account of the final revision and random events that could arise, the group has spare time to make the proper adjustments that are needed. As such, achieving each milestone within a reasonable timing window is a critical step in the successful completion of the autonomous irrigation vehicle.

The Senior Design 1 Milestones outlines a set of estimated completion dates as previously aforementioned. The estimated completion dates for the Documentation section allow the team to correct any errors or faults that may occur. Research completion dates are listed as such as to allow the team to collect the proper amount of data in order to obtain the precise information while also staying within a reasonable timeline to complete an item from the Documentation section. The biggest hurdle the group faces whilst writing the final documentation is proper formatting and documentation which could hold back the initial stages of development for the autonomous irrigation vehicle.

#	Task	Estimated Completion	Completion Date	Assignee
Senior Design 1				
1	Project Selection	January 18, 2018	January 16, 2918	All Members
2	Assignment of Roles	January 31, 2018	January 24, 2018	All Members
	Documentation			
3	Initial Project Document - Divide & Conquer	February 9, 2018	January 28, 2	All Members
4	Updated Divide & Conquer Document	March 6, 2018	March 8, 2018	All Members
5	60pg Draft Senior Design 1 Documentation	April 9, 2018	4April 9, 2018	All Members
6	Final Documentation	April 27, 2018	April 27, 2018	All Members
	Research			
7	List of peripherals & sensors to be used	February 7, 2018	February 7, 2018	All Members
8	Ideal Microcontroller Unit (MCU)	February 12, 2018	February 2, 2018	CpE Members

9	Machine Learning Algorithms	February 16, 2018	February 20, 2018	CpE Members
10	Circuit Schematics	March 9, 2018	March 12, 2018	EE Members
11	Methodology to Interface MCU with PCB & peripherals	March 16, 2018	March 16, 2018	CpE Members
12	Robot Vision Algorithms	March 16, 2018	March 16, 2018	All Members
13	Finalize Research & Acquisition of Items	April 20, 2018	April 22, 2018	All Members

Table 4: Senior Design 1 Milestones

With the given Milestones set from Table 4, the team has used the given dates to address and also estimate the completion dates of assignments, requirements, and the presentations that are due in Senior Design 2. As the estimations are to be completed a few days prior to the actual given deadlines within Table 4, Table 5 will also be constructed in an identical fashion as Senior Design Milestones 1. In Table 5, Senior Design 2 Milestones, there has been a numerous amount of new elements that have been included such as proper testing of components, ensuring efficient coding, and building of proposed circuit schematic as well as to begin prototyping. The group's aim is to complete each of the given tasks on time so as to make any last minute changes and corrections for items that have been overlooked during the entire process.

The initial printed circuit board design and autonomous irrigation vehicle design will need to be completed by the 11th of May, 2018, preferably by the end of April, 2018. Once the designs have been polished and carefully analyzed, acquisition of items will begin. With the items and components in hand, then begins building and putting together the circuit on a breadboard for further analysis and tests. As the circuit is being built, the programming of the selected microcontroller unit will have also begun along it. At a later date, proper interfacing of the printed circuit board and microcontroller unit will be performed for efficient data acquisition. Final testing of both the printed circuit board and microcontroller unit builds will then allow the group to construct a working prototype.

#	Task	Estimated Completion	Completion Date	Assignee
Senior Design 2				
	Development			
15	Initial PCB Design	May 25, 2018	May 25. 2018	EE Members
16	Initial Vehicle Design	May 25, 2018	May 25, 2018	All Members
17	Testing of Acquired Items	May 30, 2018	May 26, 2018	EE Members
18	Build proposed Circuit Schematic on Breadboard	June 8, 2018	June 11, 2018	EE Members
19	Build & Test proposed design of PCB	June 8, 2018	June 11,2018	EE Members
20	Program MCU w/ appropriate functions	June 8, 2018	June 11, 2018	CpE Members
21	Design PCB Revision 1	June 22, 2018	June 11, 2018	All Members

22	Interface MCU & PCB	June 29, 2018	June 13, 2018	All Members
23	Interfacing & debugging of MCU	June 29, 2018	June 13, 2018	All Members
24	Build Prototype Unit	July 6, 2018	June 26, 2018	All Members
25	Finalization of Prototype	July 19, 2018	July 24, 2018	All Members
	Presentation			
26	Peer Review Presentation	July 30, 2018	July 30, 2018	All Members
27	Final Report	July 30, 2018	July 30, 2018	All Members
28	Final Presentation	July 25, 2018	July 25,2018	All Members

Table 5: Senior Design 2 Milestones

Within a course of two weeks, a finalized prototype of the autonomous irrigation vehicle will have been constructed. Once the prototype of the autonomous irrigation vehicle has been constructed, final testing will commence to ensure quality has been met. Once all of the testing of the autonomous irrigation vehicle prototype has been completed, work on the final presentation will commence. The final presentation will encompass all of the group's research, documentation, and work of the autonomous irrigation vehicle over the past two semesters.

11.2 Budget and Finance Discussion

After careful consideration, our team has come to the conclusion of obtaining a couple of items in order for the autonomous irrigation vehicle to operate autonomously and safely. Throughout the research process, the autonomous irrigation vehicle has undergone multiple revisions. As such, the autonomous irrigation vehicle has discontinued the usage and implementation of certain components. Due to the revisions and the desired successful operation of the autonomous irrigation vehicle, the following items are required:

List of Items

- LIDAR
  - Allows autonomous irrigation vehicle to detect obstacles within its path
- Battery
  - Drives motors, sensors, and other applications
- Printed Circuit Board
  - Used to allow autonomous irrigation vehicle to communicate with LIDAR and other electronic components
- Microcontroller
  - Extracts data from LIDAR and mesh network of sensors

    - Analyzes and interprets extracted data
    - Operates motors
- Circuit Elements
    - Used in printed circuit board

The LIDAR will be taking in data from the vehicle's surrounding area which will be used in conjunction with the vehicle vision and machine learning algorithms that will be incorporated to allow the vehicle to move autonomously. Solar panels will be installed at the home base which will allow the vehicle to recharge its battery as well as refill its water tank from the home base's reservoir tank. The 12V battery will provide the current based on the vehicle's power demand which is dependent on the amount of water the vehicle is carrying. The microcontroller will be used to control all operations of the vehicle. PCB provides communication between the LIDAR and microcontroller. Below is Table 6 which lays out all of the financial issues that need to be addressed before the autonomous irrigation vehicle can begin its construction. Some items are subjected to change at the will of the sponsor.

Component	Estimated Cost	Actual Cost	Vendor
LIDAR	$100.00	$149.99	RobotShop
Ultrasonic Range Finder	$5.00	$16.00	RobotShop
Battery Pack	$15.00	$59.99	Amazon
Raspberry Pi	$35.00	$36.98	Amazon
Waterproof Servo	$50.00	$42.99	RobotShop
Atmega328P	$4.50	$4.50	Amazon
Brackets	$20.00	$17.90	RobotShop
Traxxas RC Car	$499.99	$249.95	Traxxas
Button & RBG LED	$5.00	$14.95	Monk Makes
Total	**$734.49**	**$593.25**	

Table 6: Project Budget

Given the high costs of the items needed to obtain the desired results of the autonomous vehicle, even when going with the economical purchases, we as a team seek out sponsorship that can provide financial aid with item procurement. Without sponsorship, we as a team propose self financing with each member paying a distributed portion in order to obtain all items on the budget list. By splitting the costs between the team members, acquirement of the needed components becomes a feasible possibility.

Summary

Current irrigation systems are seriously flawed in regard to being invasive to install, prone to frequent user error, require frequent maintenance, and expensive to install and maintain. Traditional irrigations systems also produce enormous amounts of water waste, and for all these reasons there was obvious room for improvement. Overall the goal of the autonomous irrigation vehicle was to provide water to dry regions of an area only when those regions need it to reduce water and energy consumption and waste through monitoring the soil. The autonomous irrigation vehicle will be designed to be user friendly, durable, and autonomous with obstacle avoidance capabilities to reduce repairs and maintenance. The water and energy savings coupled together with the reduced maintenance and repair costs will save the customer to which the autonomous irrigation vehicle will be used by a substantial amount of money. This project was built upon soil moisture sensors that were developed during a previous senior design project by a team of mechanical engineers who were sponsored by Guard Dog Valves, who decided to move forward with future projects to develop future water conservation technologies due to its success. The autonomous irrigation vehicle was being developed by three teams for senior design who are all sponsored by Guard Dog Valves.

The first team is made up of three mechanical engineers who a developing the physical structure of the autonomous irrigation vehicle. They plan to build it with four wheel drive to handle difficult terrain, which it could come across on all the various areas it may be deployed. The autonomous irrigation vehicle will also be designed with a water proof shell so that its water delivery system down not cause any water damage to any of the electronic systems that it depends upon for its software to find the most efficient path for it to travel to provide water to all the dry regions that need it. They will also design the autonomous irrigation vehicle with a rechargeable battery and a retractable water supply tether to supply it with water and power. In addition, they are also responsible for the development of a charging station for the autonomous irrigation vehicle to dock at in order for it to recharge its battery. The charging station will also provide the water source for the retractable water supply tether controlled by a shut off valve and will also include a reeling system to collect portions of the water supply tether as the autonomous irrigation vehicle travels back to the charging station to dock to remove any excess slack that could result in tangles that would impede upon the autonomous irrigation vehicle's ability to perform its intended water delivery duties. They are looking at the Traxxas Summit 1/10th Scale as a potential design to work with.

The second team is made up of a computer engineer, an electrical engineer, and another electrical engineer who has a computer science minor. Their contribution is the development of the wireless mesh network made up of moisture sensors to act as nodes within their network. Their moisture sensor nodes are an improvement upon the previous moisture sensor developed. They aim to include features for moisture detection, as well as humidity and temperature sensor within their sensor nodes as well as a durable protective enclosure to keep them safe from the elements and reduce maintenance, repair, and replacement labor and costs. Each sensor node within the wireless mesh network will be solar powered with excess energy stored in batteries to remove the need of invasive wires and improve the self-sustainability of the system with renewable energy, meeting one of the goals to be as eco-friendly as possible. The mesh network itself will include all of its senor nodes, the autonomous irrigation vehicle traveling around the area of the wireless mesh network, and mesh gateway to manage the network. The wireless mesh network will be scalable enough to cover very large areas and reliable to where if a node fails, the communications path between the nodes in the network can reroute to maintain the integrity of the network. They also plan to include a mobile web app to allow the user to have access to moisture readings and determine if manual watering is needed through an easy user friendly interface they can access though a smart phone or any device that can connect to the internet.

The third team was made up of two electrical engineers and two computer engineers who were developing the electronic systems and software for the autonomous irrigation vehicle to plot a course to traverse a landscape to provide water to all the regions that need it in the shortest and safest path possible. The electronic systems onboard the autonomous irrigation vehicle will be made up of a Raspberry Pi as a computer to analyze and processed received data to adjust its path and direction and an ATMEGA32P-PU microcontroller to provide communication to sensors and other onboard electronics as well as providing control for its servos and sensors. The autonomous irrigation vehicle will communicate to the wireless mesh network using the E01-ML01PX chip that uses the 802.15 Standard protocol due to its low cost, low power consumption, and wireless personal area network capabilities. While the autonomous irrigation vehicle was traveling around the area watering the areas that the wireless mesh network's sensors determines needs it, it will be detecting obstacles along its course for it to avoid and prevent structural damage and course impedance that would result in an unhappy user with maintenance costs or a dehydrated lawn. The obstacle will be detected using ultrasonic sensors and a LIDAR system that has been weather proofed and attached to a servo for one-hundred-and-eighty-degree movement where it scans the area in front of it and sends bac the data of the locations of the obstacles detected to the Raspberry Pi to label on its map with the number nine and all locations adjacent to them with the number of their adjacent obstacles. The autonomous irrigation vehicle will also include a battery management system that utilize a voltage divider integrated onto a printed PCB circuit to ensure the most efficient energy use within all the electronic systems as possible. The autonomous irrigation vehicle will map out the locations of the moisture sensors from the wireless mesh network and calculate then plot the midpoints between them onto a two-dimensional Locations object array map. From its current location, the autonomous irrigation vehicle

will look to the midpoints between the sensors to its left and right and travel to whichever has the most dry sensor readings, if the both have the same number of dry sensors that was not zero, it will travel straight ahead to the center midpoint, if none of the midpoints around it have dry senor readings but the mesh network still has dry sensors in it, the autonomous irrigation vehicle will employ the nearest neighbor algorithm to find the closest midpoint with dry sensor readings for it to travel to. If it detects multiple midpoints equal distance away with the same number of dry sensor readings, the algorithm will branch to both and look for the next one from there and overall choose the one with the shortest and safest path. It will build its path from one location to another location it utilized an A Star search algorithm.

APPENDICES

Appendix A – Reference

[1] Ackerman, Evan. Guizzo, Erico. "*iRobot Brings Visual Mapping and Navigation to the Roomba 980."* IEEE SPECTRUM, September 2015, www.spectrum. ieee. org /automaton/robotics/home-robots/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980. Accessed 14 March 2018.

[2] Alex. "*Power Distribution Boards - How to choose the right one."* DroneTest, September 2015, https://www.dronetrest.com/t/power-distribution-boards-how-to-choose-the-right-one/1259. Accessed 9 April 2018.

[3] Anderson, Mickie "*Soil-moisture Sensors May Produce Big Water Savings for Homeowners, UF Study Shows" Institute of Food and Agricultural Sciences and University Of Folrida, www.blogs.ifas.ufl.edu/news/2007/11/13/soil-moisture-sensors-may-produce-big-water-savings-for-homeowners-uf-study-shows/.* Accessed 2 April 2018.

[4] *"AVR-ATmega32A_Datasheet-Summary.*" Atmel Corporation, August 2016, ww1 .micro chip.com/downloads/en/DeviceDoc /Atmel-8155-8-bit-Microcontroller-AVR-ATmega32A_Datasheet-Summary.pdf. Accessed 14 March 2018.

[5] "*Beginner's guide to ultrasonic level transmitter."* Coulton, www.coulton .com/beginners_guide_to_ultrasonic_level_transmitters.html. Accessed 9 April 2018.

[6]Cochrum, Andrew., Corteo, Joseph., Oppel, Jason. Seth, Matthew." *The Manscaper Autonomous Lawn Mower*." University of Central Florida, 2013 PDF file.

[7]Darbyshire, Paul. Granda, Alice. Halterman, Brandon. Torres, Yahsiel. Ravenswaay van, Zavanio. Sookdeo, Jonathan. "*Integrated Water Monitoring System*" University of Central Florida, 2017 PDF file

[8] "*Droplet is a reinvention of the way plants get water."* Droplet Inc, www. smartdroplet.com*.* Accessed 14 March 2018.

[9] "*Edward H. Adelson*." Edited by Elmer, People/Adelson.txt, 22 Sept. 2017, persci.mit.edu/people/adelson.

[10] "*The Floridan Aquifer"*. The Florida Springs Institute, Floridasprings institute.org/ floridan-aquifer. Accessed 14 March 2018.

[11] Grumney, Dave. "*Ultrasonic Transmitters vs. Guided-Wave Radar for Level Measurement*." Questex LLC, March 2011. www.sensorsmag .com/components / ultrasonic-transmitters-vs-guided-wave-radar-for-level-measurement. Accessed 9 April 2018.

[12] Huang, T. Computer Vision: Evolution and Promise(PDF). 19th CERN School of Computing. Geneva: CERN. pp. 21-25. http://cds.cern.ch/record/400313/files/p21.pdf

[13] IEEE Standards Association Family of Sites. IEEE, 2018, standards.ieee.org/ Accessed 2 April 2018
[14] "*I2C*". SparkFun Electronics, www.learn.sparkfun.com/tutorials/i2c. Accessed 9 April 2018.

[15] Jackson, Jonathan., Velasco, Alejandro., Oppel, Jason. Wilson, Christopher.*" F.A.R.M. (Fundamental Agriculture Resource Monitor)*." University of Central Florida, 2015 PDF file.

[16] King, Austin., Plaza, Kevin., Baumgartner, Adam., Hromada, Ryan. "*Military Surveillance Robotic Vehicle*." University of Central Florida, 2016. PDF file.

[17] *"LIDAR-Lite v3."* GARMIN, www.buy.garmin.com/en-US/US/p/557294#overview. Accessed 9 April 2018.

[18] "*Meet the Roomba® family of vacuuming robo*ts." iRobot Corporation, www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx. Accessed 14 March 2018.

[19] Minesweeper. Curt Johnson. Microsoft, 1990. Microsoft Windows.

[20] Olson, Tim. "*How Level Measurement With Pressure Sensors Works*" Automation Products Group, August 2015, www.apgsensors.com/about-us/blog/how-level-measurement-with-pressure-sensors-works. Accessed 9 April 2018.

[21] Parent-Charette, Sébastien. "*LIDAR-Lite Laser Rangefinder – Simple Arduino Sketch of a 180 Degree "Radar".*" RobotShop Inc., April 2015, www.robotshop .com/blog/en/LIDAR-lite-laser-rangefinder-simple-arduino-sketch-of-a-180-degree-radar-15284. Accessed 9 April 2018.

[22] Parent-Charette, Sébastien. "*LIDAR-Lite v3 – Available for pre-order."* RobotShop Inc., September 2016, www.robotshop.com/blog/en/LIDAR-lite-v3-available-for-pre-order-19202. Accessed 9 April 2018.

[23] Pfleeger, Shari Lawrence., Atlee.M, Jonne. *Software Engineering Theory and practice*. Pearson education, 2010.

[24] Prathik, A., et al. "*An Overview of Application of Graph Theory*." International Journal of ChemTech Research, vol. 9, no. 2, 2016, pp. 242–248., www.sphinxsai.com/2016/ch_vol9_no2/1/(242-248)V9N2CT.pdf.

[25] "*RPLIDAR A1M8*". Slamtec. https://www.robotshop.com/media/files/pdf/rpLIDAR-a1m8-360-degree-laser-sca1nner-development-kit-datasheet-1.pdf Accessed 7 Apr. 2018.

[26] Russell, Stuart J., and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice-Hall, 2010.

[27] Sam Travarca. *"Reason for an Development integrated dvelopment environment"*. Vantageone Software, http://vantageonesoftware.com/reasons-integrated-development-environment/. Accessed 8 Apr. 2018.

[28] *"SCHEMATICS - BATTERY MONITOR CIRCUIT"*, Society of Robots, www. society ofrobots.com/schematics_batterymonitor.shtml. Accessed 14 March 2018.

[29] "*Summit.*", Traxxas. www.traxxas.com/products/models/electric/summit. Accessed 14 March 2018.
[30] "*Water Level Sensors*". RS Hydro, http://www.rshydro.co.uk/water-level-sensors/. Accessed 9 April 2018.

[31] "*When it's Hot."* U.S. Environmental Protection Agency, www.epa .gov/watersense/when-its-hot. Accessed 2 April 2018.

[32] "Why learn C++". Codementor. http://www.bestprogramminglanguagefor.me/why-learn-c-plus-plus Accessed 7 Apr. 2018.

[33] Zhang, Qian. Zhang, Young-Fei, Qin, Shi-Yin, " *Modeling and Analysis for Obstacle Avoidance of a Behaviour-Based Robot with Objected oriented methods.*" Journel of computers, Vol .4, No. 4 (2009).

Appendix B - Copyrights Permissions

Alexandre (RobotShop)
Apr 10, 09:16 EDT

Hi Parameswari,

Thank you for contacting Robotshop. You can use that image and table data without any problem.

Please let us know if you have any other questions.

Regards,

Alexandre
RobotShop inc.

Parameswari Chandrasekar
Apr 9, 19:31 EDT

I'm a Computer Engineering student at University of Central Florida. I would like to request permission to use the image of "PRLIDAR A1m8" model that you have listed at https://www.robotshop.com/en/rplidar-a1m8-360-degree-laser-scanner-development-kit.html and the measurement performance table data that is associated with the PRLIDAR A1m8 model
If permission is granted, the image and the table will be used in a senior design paper.

Thank you,
Parameswari

Figure 48: Permission for PRLidar A1M8 Model

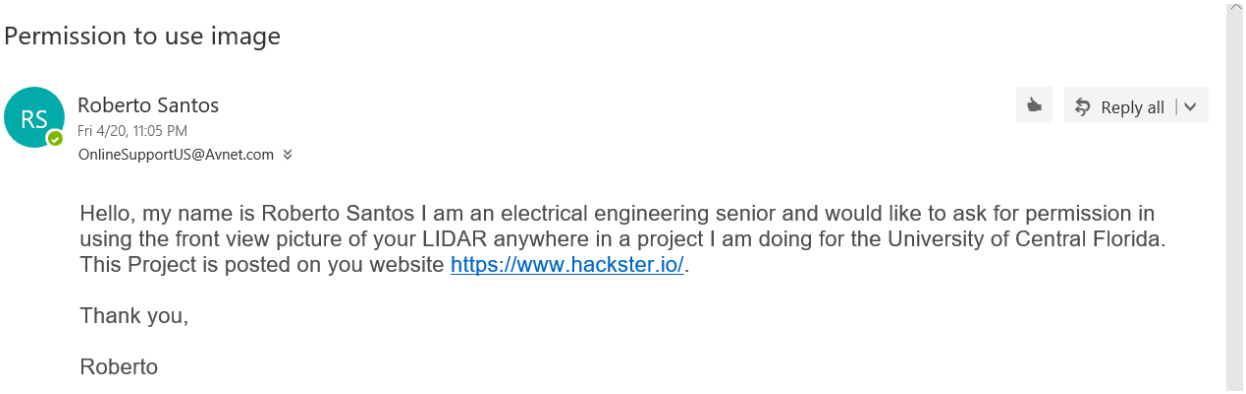


Figure 49: Waterproofing Permission

Request Permission

RS Roberto Santos
Tue 4/3, 8:07 PM

Terri,

Thank you. The project is a autonomous lawn irrigation vehicle.

Sincerely,

Roberto

T Terri.Thill@ocfl.net
Tue 4/3, 2:05 PM

Sure

What is your project about? April is Water Conservation Month

https://www.sjrwmd.com/water-conservation/

Terri Thill
Orange County Utilities Water Division
9150 Curry Ford Road
Orlando, FL 32825
407-254-9602

RS Roberto Santos
Mon 4/2, 10:30 AM
Water.Division@ocfl.net

Reply all

Hello,

I would like to request permission to use an image of the table in the Watering Restrictions part of your web page. The image will be used in a project paper for the University of Central Florida.

Thank you,

Roberto Santos

Figure 50: OUC Permission

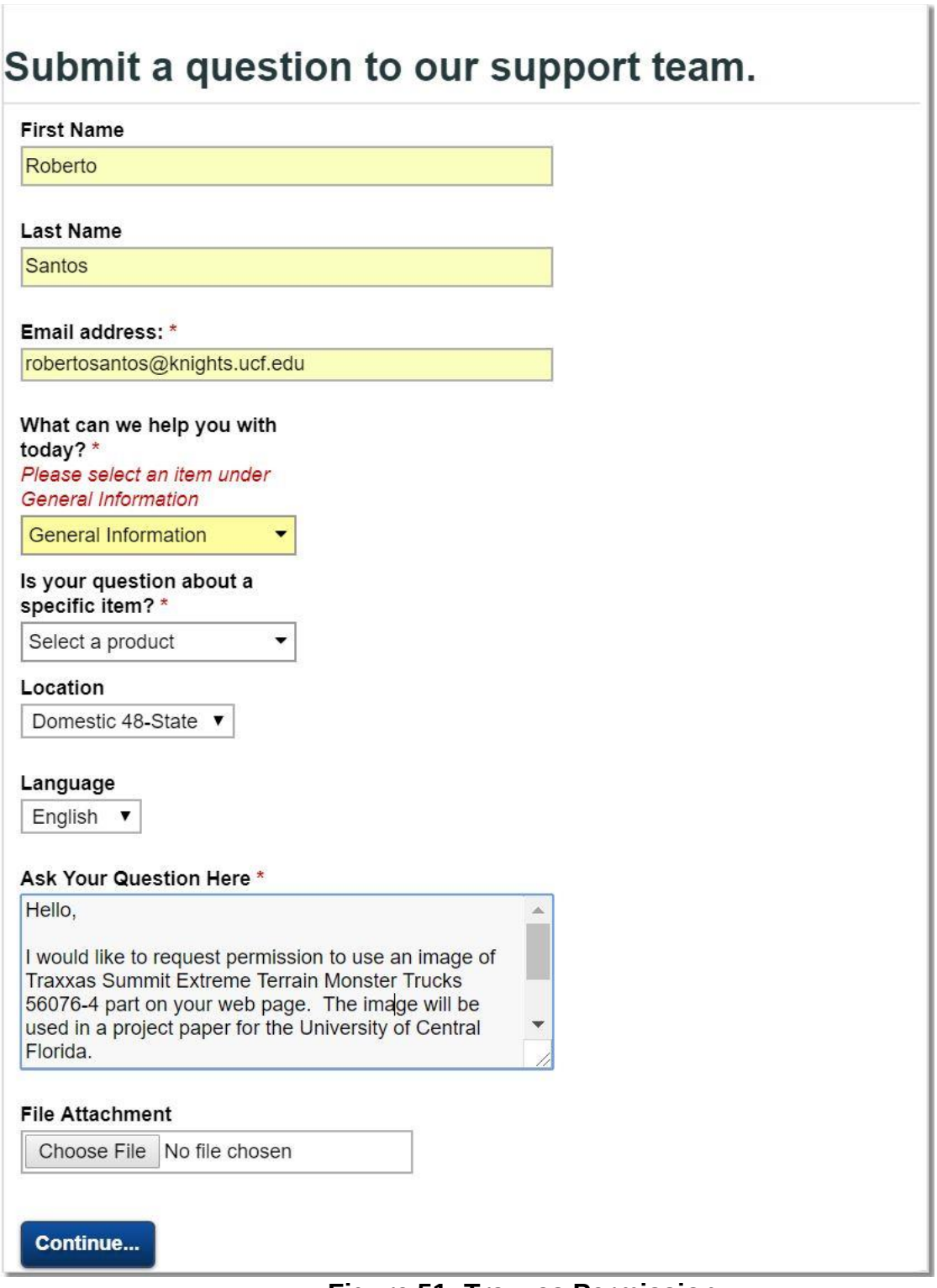


Figure 51: Traxxas Permission

Appendix C - Datasheets

Test	Test description	Test condition	Expected results	Results (pass/fail)
PathFinding test	listed in Pathfinding test	The test will be performed in the environment	Should pass pathfinding test	
Obstacle Avoidance test	Listed in Obstacle avoidance test	The test will be performed in the environment	Should pass obstacle avoidance test	
Mapping test	Listed in map testing	The test will be performed in the environment	Should pass mapping test	
Water allocation test	listed in water allocation testing section	The test will be performed in the environment	Should pass water allocation test	
Backtracking testing	listed in backtracking testing section	The test will be performed in the environment	Should pass backtracking test	

Table 7: Test Plan for Performance Testing

Appendix D - Source Code

```
#ifdef RaspberryPi   // just that the Arduino IDE doesnt compile these files.

#include <bits/stdc++.h>
#include <iostream>
#include <stdio.h>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <queue>
#include <cmath>
#include <unistd.h>
#include <string.h> //for errno
#include <errno.h> //error output
#include <stdint.h> //uint8_t definitions
#include <fcntl.h>
#include <termios.h>
#include <fstream>
```

```
#include <jsoncpp/json/json.h>
#include "IMU.c"
#include <wiringPi.h> //wiring Pi
#include <wiringSerial.h>

using namespace std;

#define PI 3.14159265
#define cmToFeet 30.48
#define metersToFeet 3.28084
#define SprinklerRadius 3.0

#define magXmax 2378
#define magYmax 775
#define magZmax -279
#define magXmin -1282
#define magYmin -322
#define magZmin -1268

//class definitions
class Locations;
class Sensor;
class Midpoint;

//struct definitions
struct obstacleData
{
    int array[3];
    int index;

};

//global variables and structures
struct obstacleData obstacleDetect;
vector<Sensor> SList;
vector<Midpoint> MList;
vector<Midpoint> WaterList;
Locations** mappy;
int dir;
int rows;
int cols;
bool pathExists;

// Find Serial device on Raspberry with ~ls /dev/tty*
// ARDUINO_UNO "/dev/ttyACM0"
// FTDI_PROGRAMMER "/dev/ttyUSB0"
// HARDWARE_UART "/dev/ttyAMA0"
```

```
char device[]= "/dev/ttyACM0";  //drive
// filedescriptor
int fd;    // drive

char Lidar[]= "/dev/ttyS0";  // lidar
// filedescriptor
int fl;      // lidar

unsigned long baud = 9600;
unsigned long highBaud = 115200;
//unsigned long time=0;

//classes
class Locations
{

public:

    int r, c, hazardLevel, prev_r, prev_c;
    double f, g, h;

    Locations();
    Locations(int, int);
    void setCoords(int, int);
    ~Locations();
};

class Sensor
{

public:

    int id, r, c;
    vector<Midpoint> adjMids;
    bool wet;
    double dist, distOffset;
    string proxID;

    Sensor();
    Sensor(int, int, int, bool);
    void setSID(int);
    void setSCoords(int, int);
    void addMids(Midpoint);
    void setSWetness(bool);
    ~Sensor();
};
```

```
class Midpoint
{

public:

    int id, r, c;
    Sensor S1, S2;
    int dryness;

    Midpoint();
    Midpoint(int, int, int, Sensor, Sensor);
    void setMID(int);
    void setMCoords(int, int);
    void setSensors(Sensor, Sensor);
    void setDryness(Sensor, Sensor);
    ~Midpoint();
};

//constructors
Locations::Locations()
{
    r = 0;
    c = 0;
}

Locations::Locations(int rLoc, int cLoc)
{
    setCoords(rLoc, cLoc);
    prev_r = r;
    prev_c = c;
    f = 0.0;
    g = 0.0;
    h = 0.0;
}

void Locations::setCoords(int rLoc, int cLoc)
{
    r = rLoc;
    c = cLoc;
}

Locations::~Locations()
{
    ;
}

Sensor::Sensor()
```

```
{
    id = 1000000;
    r = 0;
    c = 0;
    wet = false;
    dist = FLT_MAX;
}

Sensor::Sensor(int identity, int rLoc, int cLoc, bool w)
{
    setSID(identity);
    setSCoords(rLoc, cLoc);
    setSWetness(w);
}

void Sensor::setSID(int identity)
{
    id = identity;
}

void Sensor::setSCoords(int rLoc, int cLoc)
{
    r = rLoc;
    c = cLoc;
}

void Sensor::addMids(Midpoint Mid)
{
    adjMids.push_back(Mid);
}

void Sensor::setSWetness(bool w)
{
    wet = w;
}

Sensor::~Sensor()
{
    ;
}

Midpoint::Midpoint()
{
    id = 2000000;
    r = 0;
    c = 0;
    dryness = 0;
```

```
}

Midpoint::Midpoint(int identity, int rLoc, int cLoc, Sensor Sensor1, Sensor Sensor2)
{
   setMID(identity);
   setMCoords(rLoc, cLoc);
   setSensors(Sensor1, Sensor2);
   setDryness(Sensor1, Sensor2);
}

void Midpoint::setMID(int identity)
{
   id = identity;
}

void Midpoint::setMCoords(int rLoc, int cLoc)
{
   r = rLoc;
   c = cLoc;
}

void Midpoint::setSensors(Sensor Sensor1, Sensor Sensor2)
{
   S1 = Sensor1;
   S2 = Sensor2;
}

void Midpoint::setDryness(Sensor Sensor1, Sensor Sensor2)
{
   dryness = 0;

   if (Sensor1.wet == false)
      dryness++;

   if (Sensor2.wet == false)
      dryness++;
}

Midpoint::~Midpoint()
{
   ;
}

//helper function prototypes
char * serialGets(char *buf, const int n, const int fg);

int FindStart(Locations cur);
```

```
int findDir(int nextID, int leftID, int rightID);
int calcMid(int n1, int n2);
int InitialDir(int startingMidpointID, int r);
int FindDir(int next, int left, int right);
int countDry(int* MPDry);
int NearestDryDestination(int* neighbors, int* MPDry);
int getPsuedoAngle(Locations cur, Locations next, int curAng);
int getDriveAngle(int rDiff, int cDiff, int ang);
int getCompassDirection();
void getObsDataArr();
int* setNeighbors(int curID, int r, int* neighbors);
int* DrynessCheck(bool* SReading, int* MPDry, int cur);
Locations** updatemap(int tRow, int tCol);
double findDistance(int curR, int curC, int destR, int destC);
bool findObstacleLoc(int VRow, int VCol, int VAng, int obDist, int obAng);
bool atEdge(int curID, int r);
bool inBounds(int row, int col);
bool isSame(int row, int col, int i, int j);
bool sameLocation(Locations curLoc, Locations goal);
bool sameCoordinate(int crow, int ccol, int grow, int gcol);
bool isGoal(Locations current, Locations Goal);
bool isObstacle(Locations** LocMap, int row, int col);
bool* SensorCheck(bool* SReading, int cur);
void commSetup();
```

```
void LidarSetup();

void printPrevMap(Locations** LocMap);

void printSList();

void printLPList(vector<Locations> LPList);

void printMap();

void freeIntArray(int** arr);

void freeBoolArray(bool** arr);

void freeLocationsArray(Locations** arr);

void readMAG(int * m);

Locations getCurrentLoc(int r, int c);

Locations getProxSensorLoc();

Locations Trilocalization(Sensor* sensArray);

vector<Midpoint> findMidpoints(int neighborRows, int neighborCols);

vector<Locations>buildPath(Locations** LocMap, Locations end);

vector<Locations> AStarSearch(Locations start, Locations end);

vector<Locations> FindLongPath(Locations initial, int neighborRows);

//helper functions
char * serialGets(char *buf, const int n, const int fg)
{
    int m;
    m = read(fg, buf, n);
    *(buf + m) = '\0';
    return (buf);
}

int FindStart(Locations cur)
{
    Locations close = cur;
    int i, dist = 0, BestDist = INT16_MAX, MSize = MList.size(), MID = 0, nextID = 0;
    for (i = 0; i < MSize; i++)
    {
        MID = MList[i].id;
```

```
        Locations next(MList[i].r, MList[i].c);
        vector<Locations> path = AStarSearch(next, cur);
        dist = path.size();
        if (dist < BestDist)
        {
            nextID = MID;
            BestDist = dist;
            //printf("dist = %d, nextID = %d", BestDist, nextID);
        }
    }
    return nextID;
}

int findDir(int nextID, int leftID, int rightID)
{
    int newDir = dir;

    if (nextID == leftID)
        newDir = (dir - 1) % 4;

    if (nextID == rightID)
        newDir = (dir + 1) % 4;

    return newDir;
}

int calcMid(int n1, int n2)
{
    int m = 0;
    m = (n1 + n2) / 2;
    return m;
}

int InitialDir(int startingMidpointID, int r)
{
    int LeftID, MSize = MList.size();
    Midpoint LeftMid, cur = MList[startingMidpointID];

    LeftID = cur.id + (r - 1); // for east map direction
    if (LeftID > 0 && LeftID < MSize)
    {
        LeftMid = MList[LeftID];
        if (LeftMid.S1.id == cur.S1.id)
            return 0;
    }

    LeftID = cur.id + r;  // for south map direction
```

```
    if (LeftID > 0 && LeftID < MSize)
    {
        LeftMid = MList[LeftID];
        if (LeftMid.S1.id == cur.S2.id)
            return 1;
    }

    LeftID = (cur.id + 1) - r;  // for west map direction
    if (LeftID > 0 && LeftID < MSize)
    {
        LeftMid = MList[LeftID];
        if (LeftMid.S2.id == cur.S2.id)
            return 2;
    }

    LeftID = cur.id - r;  // for north map direction
    if (LeftID > 0 && LeftID < MSize)
    {
        LeftMid = MList[LeftID];
        if (LeftMid.S2.id == cur.S1.id)
            return 3;
    }

    return 4;
}

int FindDir(int next, int left, int right)
{
    int newDir = dir;
    if (next == left)
        newDir = (dir - 1) % 4;

    if (next == right)
        newDir = (dir + 1) % 4;

    return newDir;
}

int countDry(int* MPDry)
{
    int i, DryCount = 0, MPDSize = MList.size();
    for (i = 0; i < MPDSize; i++)
        if (MPDry[i] > 0)
            DryCount++;

    return DryCount;
}
```

```
int NearestDryDestination(int* neighbors, int* MPDry)
{
   int i, next = 0, closest = 0, MSize = MList.size();
   double Dist, BestDist = FLT_MAX;
   for (i = 0; i < MSize; i++)
   {
      if (MPDry[i] > 0) //to make sure its dry
      {
         //AStarSearch used to find distance instead of distance formula
         Locations startNext(MList[neighbors[0]].r, MList[neighbors[0]].c),
endNext(MList[i].r, MList[i].c);
         vector<Locations> nextPath = AStarSearch(endNext, startNext);
         Dist = (double)nextPath.size();
         //printf("%d is more dry than 0 at distance %lf\n", i, Dist);

         if (Dist < BestDist)
         {
            next = i;
            BestDist = Dist;
            //printf("next is %d at distance %lf\n", i, Dist);
         }
      }
   }
   //printf("next is %d\n", next);
   BestDist = FLT_MAX;
   for (i = 0; i < 4; i++)
   {
      //AStarSearch used to find distance
      Locations startClose(MList[neighbors[i]].r, MList[neighbors[i]].c),
endClose(MList[next].r, MList[next].c);
      vector<Locations> closePath = AStarSearch(endClose, startClose);
      Dist = (double)closePath.size();

      if (Dist < BestDist)
      {
         closest = i;
         BestDist = Dist;
      }
   }
   if (closest == 0)
      dir = (dir + 2) % 4;

   return neighbors[closest];
}

int getPsuedoAngle(Locations cur, Locations next, int curAng)
```

```
{
    if (sameLocation(cur, next))
        return curAng;

    int rDiff = cur.r - next.r;
    int cDiff = next.c - cur.c;
    int calcAng = (int)((atan2(rDiff, cDiff) * 180) / PI);
    //printf(" rDiff = %d, cDiff = %d, calcAng = %lf ", rDiff, cDiff, calcAng);
    if (calcAng < 0)
        calcAng += 360;

    if (calcAng != curAng)
        return calcAng;

    return curAng;
}

int getDriveAngle(int rDiff, int cDiff, int ang)
{
    int turnAng;
    int posAng = (int)((180 * atan2(rDiff, cDiff)) / PI);

    if (posAng < 0)
        posAng += 360;
    if (ang < 0)
        ang += 360;

    //printf("current angle at %d pos angle at %d\n", ang, posAng);

    turnAng = ang - posAng;
    turnAng = (turnAng / 2) + 90;
    if (turnAng < 0)
        turnAng += 180;

    if (turnAng > 135)
        turnAng -= 180;

    return turnAng;
}

int getCompassDirection()
{
    int magRaw[3];
    float scaledMag[3];

    readMAG(magRaw);
```

```
    //Apply hard iron calibration
    magRaw[0] -= (magXmin + magXmax) /2 ;
    magRaw[1] -= (magYmin + magYmax) /2 ;
    magRaw[2] -= (magZmin + magZmax) /2 ;

    //Apply soft iron calibration
    scaledMag[0]  = (float)(magRaw[0] - magXmin) / (magXmax - magXmin) * 2 - 1;
    scaledMag[1]  = (float)(magRaw[1] - magYmin) / (magYmax - magYmin) * 2 - 1;
    scaledMag[2]  = (float)(magRaw[2] - magZmin) / (magZmax - magZmin) * 2 - 1;

    //Compute heading
    float heading = 180 * atan2(scaledMag[1],scaledMag[0])/M_PI;


    //Convert heading to 0 - 360
    if(heading < 0)
          heading += 360;


    //Local declination in mrads into radians
    float declination = 217.9 / 1000.0;

    //Add the declination correction to our current heading
    heading += declination * 180/M_PI;


    //Correct the heading if declination forces it over 360
    if ( heading > 360)
       heading -= 360;

    return heading;
}

void getObsDataArr()
{
    fflush(stdout);
    int q, numTest;
    char highBits, lowBits;
    for(q = 0; q < 3; q++)
    {
       if(serialDataAvail (fl))
       {
          highBits = serialGetchar (fl);
          lowBits = serialGetchar (fl);

          numTest = lowBits | highBits << 8;

          //setvalue depending on position in array
```

```
            obstacleDetect.array[(obstacleDetect.index) % 3] = numTest;
            //printf("lidar obstacleDirection numTest = %d\n", numTest);
            //increment the index
            (obstacleDetect.index)++;

            //printf("%d\n", numTest);
            //printf("%c", highBits);
            //fflush(stdout);
        }
        //printf("raw data obstacleDetect[%d] = %d\n", q, obstacleDetect.array[q]);
    }
    fflush(stdout);
}

int* setNeighbors(int curID, int r, int* neighbors)
{
    neighbors[0] = curID;

    if (dir == 0)
    {
        neighbors[1] = curID + (r - 1); //left
        neighbors[2] = curID + ((r * 2) - 1);  //center
        neighbors[3] = curID + r;  //right
        return neighbors;
    }

    if (dir == 1)
    {
        neighbors[1] = curID + r; //left
        neighbors[2] = curID + 1;  //center
        neighbors[3] = (curID + 1) - r;  //right
        return neighbors;
    }

    if (dir == 2)
    {
        neighbors[1] = (curID + 1) - r; //left
        neighbors[2] = (curID + 1) - (r * 2);  //center
        neighbors[3] = curID - r;  //right
        return neighbors;
    }

    if (dir == 3)
    {
        neighbors[1] = curID - r; //left
        neighbors[2] = curID - 1;  //center
        neighbors[3] = curID + (r - 1);  //right
```

```
        return neighbors;
    }

    return neighbors;
}

int* DrynessCheck(bool* SReading, int* MPDry, int cur)
{
    int i, SensID = 0, SenAdj = 0;
    if (MPDry[cur] > 0)// only if current midpoint is dry
    {
        MPDry[cur] = 0; // set current midpoint to dry

        SensID = MList[cur].S1.id;
        SenAdj = SList[SensID].adjMids.size();
        for (i = 0; i < SenAdj; i++)
            if ((MPDry[SList[SensID].adjMids[i].id] > 0) && !SReading[SensID])
                MPDry[SList[SensID].adjMids[i].id]--;

        SensID = MList[cur].S2.id;
        SenAdj = SList[SensID].adjMids.size();
        for (i = 0; i < SenAdj; i++)
            if ((MPDry[SList[SensID].adjMids[i].id] > 0) && !SReading[SensID])
                MPDry[SList[SensID].adjMids[i].id]--;
    }
    /*int MSize = MList.size();
    for (i = 0; i < MSize; i++)
    printf("dryness at midpoint %d is %d\n", i, MPDry[i]);*/

    return MPDry;
}

Locations** updatemap(int tRow, int tCol)
{
    int i, j, p, q, count;
    mappy[tRow][tCol].hazardLevel = 9;
    for (i = -1; i<2; i++)
    {
        for (j = -1; j<2; j++)
        {
            if ((inBounds(tRow + i, tCol + j) && !(isSame(tRow, tCol, i, j))))
            {
                if ((mappy[tRow + i][tCol + j].hazardLevel != 9) && (mappy[tRow + i][tCol +
j].hazardLevel < 1000000))
                {
                    count = 0;
                    for (p = -1; p <2; p++)
```

```
                {
                    for (q = -1; q <2; q++)
                    {
                        if ((inBounds(tRow + i + p, tCol + j + q) && !(isSame(tRow + i, tCol + j,
p, q))))
                        {
                            if (mappy[tRow + i + p][tCol + j + q].hazardLevel == 9)
                            {
                                count++;
                            }
                        }
                    }
                }
                mappy[tRow + i][tCol + j].hazardLevel = count;

            }
        }

    }

}

    return mappy;
}

double findDistance(int curR, int curC, int destR, int destC)
{
    //distance formula
    double rDiff = (double)curR - (double)destR;
    double cDiff = (double)curC - (double)destC;
    double rSquare = rDiff * rDiff;
    double cSquare = cDiff * cDiff;
    double squareSum = rSquare + cSquare;
    double dist = ((double)sqrt(squareSum));
    return dist;
}

bool findObstacleLoc(int VRow, int VCol, int VAng, int obDist, int obAng)
{
    int ORow, OCol, tRow = 0, tCol = 0;
    bool updated = false;
    double RelativeAngle = (double)((VAng + obAng) % 360);
    ORow = (int)round((double)(obDist * sin(RelativeAngle * (PI / 180))));
    OCol = (int)round((double)(obDist * cos(RelativeAngle * (PI / 180))));
    //printf("obstace distance coords: ORow = %d OCol = %d\n", tRow, tCol);

    tRow = VRow - ORow;
```

```
    //printf("obstace location coords: tRow = %d ", tRow);
    tCol = VCol + OCol;
    //printf("tCol = %d\n", tCol);

    if (inBounds(tRow, tCol))
    {
        if ((mappy[tRow][tCol].hazardLevel) != 9 && (mappy[tRow][tCol].hazardLevel <
1000000))
        {
            mappy = updatemap(tRow, tCol);
            updated = true;
        }
    }
    return updated;
}

bool atEdge(int curID, int r)
{
    int Left, MSize = MList.size();
    Midpoint LeftMid, cur = MList[curID];

    if (dir == 0)
    {
        Left = curID + (r - 1);

        if (Left < MSize && Left > 0)
            LeftMid = MList[Left];
        else
            return true;

        if (cur.S1.id == LeftMid.S1.id)
            return false;
        else
            return true;
    }
    else if (dir == 1)
    {
        Left = curID + r;

        if (Left < MSize && Left > 0)
            LeftMid = MList[Left];
        else
            return true;

        if (cur.S2.id == LeftMid.S1.id)
            return false;
        else
```

```
            return true;
        }
        else if (dir == 2)
        {
            Left = (curID + 1) - r;

            if (Left < MSize && Left > 0)
                LeftMid = MList[Left];
            else
                return true;

            if (cur.S2.id == LeftMid.S2.id)
                return false;
            else
                return true;
        }
        else if (dir == 3)
        {
            Left = curID - r;

            if (Left < MSize && Left > 0)
                LeftMid = MList[Left];
            else
                return true;

            if (cur.S1.id == LeftMid.S2.id)
                return false;
            else
                return true;
        }

        return true;
    }

    bool isSingleDigit(int num)
    {
        if (num > -1 && num < 10)
            return true;
        else
            return false;
    }

    bool inBounds(int row, int col)
    {
        if ((row > -1) && (row < rows) && (col > -1) && (col < cols))
            return true;
        else
```

```
        return false;
}

bool isSame(int row, int col, int i, int j) //determine of locations are the same based on
itteration
{
    if (((row + i) == row) && ((col + j) == col))
        return true;
    else
        return false;
}

bool sameLocation(Locations curLoc, Locations goal) //determining if two defined
locations are the same
{
    if ((curLoc.r == goal.r) && (curLoc.c == goal.c))
        return true;

    return false;
}

bool sameCoordinate(int crow, int ccol, int grow, int gcol) //determining if two locations
are the same by coordinates
{
    if ((crow == grow) && (ccol == gcol))
        return true;

    return false;
}

bool isGoal(Locations current, Locations Goal)
{
    if ((current.r == Goal.r) && (current.c == Goal.c))
        return true;
    else
        return false;
}

bool* SensorCheck(bool* SReading, int cur)
{
    SReading[MList[cur].S1.id] = true;
    SReading[MList[cur].S2.id] = true;
    return SReading;
}

bool isObstacle(Locations** LocMap, int row, int col)
{
```

```
    //printf("checking obstacles at r = %d c = %d and hlevel at %d\n", row, col,
LocMap[row][col].hazardLevel);
    if (LocMap[row][col].hazardLevel == 9)
        return true;
    else
        return false;
}

void commSetup()
{

  printf("%s \n", "Raspberry Startup!");
  fflush(stdout);

  //get filedescriptor
  if ((fd = serialOpen (device, baud)) < 0){
    fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
    exit(1); //error
  }

  //setup GPIO in wiringPi mode
  if (wiringPiSetup () == -1){
    fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
    exit(1); //error
  }

}

void LidarSetup()
{

    //initialize array and index to 0
    int i;
    for(i = 0; i < 3; i++)
    {
        obstacleDetect.array[i] = 0;
    }
    obstacleDetect.index = 0;

    printf("%s \n", "Raspberry Startup!");
    fflush(stdout);

    //get filedescriptor
    // ((fl = serialOpen (Lidar, baud)) < 0)
    if ((fl = serialOpen (Lidar, highBaud)) < 0)
    {
        fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
```

```
        exit(1); //error
    }

    //setup GPIO in wiringPi mode
    if (wiringPiSetup () == -1)
    {
        fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
        exit(1); //error
    }

}

void printPrevMap(Locations** LocMap)
{
    int i, j;
    printf("\n");
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            printf(" [at (%d,%d) from (%d,%d)] ", LocMap[i][j].r, LocMap[i][j].c,
LocMap[i][j].prev_r, LocMap[i][j].prev_c);
        }
        printf("\n");
    }
    printf("\n");
}

void printSList()
{
    int i, j, sSize = SList.size(), mSize = 0;
    printf("SList size = %d\n", sSize);
    for (i = 0; i < sSize; i++)
    {
        printf("Sensor id %d, row %d, col %d wetness ", SList[i].id, SList[i].r, SList[i].c);
        printf(SList[i].wet ? "true" : "false");
        mSize = SList[i].adjMids.size();
        printf(" mSize is %d\n", mSize);
        for (j = 0; j < mSize; j++)
        {
            printf(" Sensor midpoint %d, row %d, col %d ", SList[i].adjMids[j].id,
SList[i].adjMids[j].r, SList[i].adjMids[j].c);
        }
        printf("\n");
    }
    printf("\n done \n");
}
```

```
void printLPList(vector<Locations> LPList)
{
    int i, LPSize = LPList.size();
    //printf("LPList size = %d\n", LPSize);
    for (i = 0; i < LPSize; i++)
        printf("Move %d, row %d, col %d\n", i, LPList[i].r, LPList[i].c);
}

void printMap()
{
    int i, j;
    printf("\n printing map \n");
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            int val = mappy[i][j].hazardLevel;
            if(val < 1000000)
                printf("%d ", val);
            else
            {
                if(val < 2000000)
                {
                    //val -= 1000000;
                    printf("S ");
                }
                else if(val >= 3000000)
                {
                    //val -= 3000000;
                    printf("T ");
                }
                else
                {
                    //val -= 2000000;
                    printf("M ");
                }
            }
        }
        printf("\n");
    }
}

void freeIntArray(int** arr)
{
    int i;
    for (i = 0; i < rows; i++)
```

```
        free(arr[i]);

    free(arr);
}

void freeBoolArray(bool** arr)
{
    int i;
    for (i = 0; i < rows; i++)
        free(arr[i]);

    free(arr);
}

void freeLocationsArray(Locations** arr)
{
    int i;
    for (i = 0; i < rows; i++)
        free(arr[i]);

    free(arr);
}

Locations getCurrentLoc(int r, int c)
{
    //later modify to get coordinate inputs of current location from mesh network
    Locations cur(r, c);
    return cur;
}

Locations getProxSensorLoc()
{
    int i, SenSize = SList.size();
    Sensor closestSensors[3];
    double* SensDistArray = (double*)malloc(sizeof(double) * SenSize);
    for(i = 0; i < SenSize; i++)  //make and fill array by sensor distances
        SensDistArray[i] = SList[i].dist;

    sort(SensDistArray, SensDistArray + SenSize); //sort by distance in ascending order
    if(SList.size() > 2)  //find the three closest sensors to calculate position from
    {
        for(i = 0; i < SenSize; i++)
        {
            if(SList[i].dist == SensDistArray[0])
            {
                closestSensors[0] = SList[i];
                break;
```

```
            }
        }

        for(i = 0; i < SenSize; i++)
        {
            if((SList[i].dist == SensDistArray[1]) && (SList[i].id != closestSensors[0].id))
            {
                closestSensors[1] = SList[i];
                break;
            }
        }

        for(i = 0; i < SenSize; i++)
        {
            if((SList[i].dist == SensDistArray[2]) && (SList[i].id != closestSensors[0].id) &&
(SList[i].id != closestSensors[1].id))
            {
                closestSensors[2] = SList[i];
                break;
            }
        }
    }

    //for(i = 0; i < 3; i++)
        //cout << "sensor = " << closestSensors[i].id << " at " << closestSensors[i].dist <<
"\n";

    Locations cur = Trilocalization(closestSensors);
    return cur;
}

Locations Trilocalization(Sensor* sensArray)
{
    double A = (double)((-2 * sensArray[0].r) + (2 * sensArray[1].r));
    double B = (double)((-2 * sensArray[0].c) + (2 * sensArray[1].c));
    double C = (double)((sensArray[0].dist * sensArray[0].dist)
    - (sensArray[1].dist * sensArray[1].dist)
    - (sensArray[0].r * sensArray[0].r) + (sensArray[1].r * sensArray[1].r)
    - (sensArray[0].c * sensArray[0].c) + (sensArray[1].c * sensArray[1].c));
    double D = (double)((-2 * sensArray[1].r) + (2 * sensArray[2].r));
    double E = (double)((-2 * sensArray[1].c) + (2 * sensArray[2].c));
    double F = (double)((sensArray[1].dist * sensArray[1].dist)
    - (sensArray[2].dist * sensArray[2].dist)
    - (sensArray[1].r * sensArray[1].r) + (sensArray[2].r * sensArray[2].r)
    - (sensArray[1].c * sensArray[1].c) + (sensArray[2].c * sensArray[2].c));
    int curR = (int)(((C * E) - (F * B)) / ((E * A) - (B * D)));
    int curC = (int)(((C * D) - (F * A)) / ((B * D) - (E * A)));
```

```
    Locations cur(curR, curC);
    return cur;
}

vector<Midpoint> findMidpoints(int neighborRows, int neighborCols)
{
    int i, j, mr, mc, offset, count = 0;
    vector<Midpoint> MList;
    for (i = 0; i < neighborCols; i++)
    {
        offset = i * neighborRows;//for each column
        for (j = 0; j < (neighborRows - 1); j++)
        {
            //calcmidpointplot(mappy, neighborMap, count, J + offset, J + 1 + offset);
            //printf("midpoint %d calculated\n", count);
            mr = calcMid(SList[j + offset].r, SList[j + 1 + offset].r);
            mc = calcMid(SList[j + offset].c, SList[j + 1 + offset].c);
            Midpoint cur(count, mr, mc, SList[j + offset], SList[j + 1 + offset]);
            MList.push_back(cur);
            SList[j + offset].addMids(cur);
            SList[j + 1 + offset].addMids(cur);
            count++;
        }

        if (i < (neighborCols - 1)) //to avoid out of bounds for between columbs
        {
            for (j = 0; j < neighborRows; j++)
            {
                //calcmidpointplot(mappy, neighborMap, count, J + offset, J + rows + offset);
                //printf("midpoint %d calculated\n", count);
                mr = calcMid(SList[j + offset].r, SList[j + neighborRows + offset].r);
                mc = calcMid(SList[j + offset].c, SList[j + neighborRows + offset].c);
                Midpoint cur(count, mr, mc, SList[j + offset], SList[j + neighborRows + offset]);
                MList.push_back(cur);
                SList[j + offset].addMids(cur);
                SList[j + neighborRows + offset].addMids(cur);
                count++;
            }
        }
    }
    return MList;
}

vector<Locations>buildPath(Locations** LocMap, Locations end)
{
    vector<Locations> path;
```

```
    //printPrevMap(mappy, length, width);
    int row = end.r, col = end.c, temp;
    while (!(LocMap[row][col].prev_r == row && LocMap[row][col].prev_c == col))
    {
        path.push_back(LocMap[row][col]);
        temp = row;
        row = LocMap[row][col].prev_r;
        col = LocMap[temp][col].prev_c;
    }
    path.push_back(LocMap[row][col]);
    return path;
}

vector<Locations> AStarSearch(Locations start, Locations end)
{
    int i, j;
    double mult;
    bool pathFound = false;
    vector<Locations> path, BestPath;
    queue<Locations> q;

    /*Locations** LocMap = (Locations**)malloc(sizeof(Locations*) * rows);
    for (j = 0; j < rows; j++)
        LocMap[j] = (Locations*)malloc(sizeof(Locations) * cols);*/

    bool** visited = (bool**)malloc(sizeof(bool*) * rows);
    for (int j = 0; j < rows; j++)
        visited[j] = (bool*)malloc(sizeof(bool) * cols);

    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            visited[i][j] = false;
            mappy[i][j].r = i;
            mappy[i][j].c = j;
            mappy[i][j].prev_r = -1;
            mappy[i][j].prev_c = -1;
            mappy[i][j].f = FLT_MAX;
            mappy[i][j].g = FLT_MAX;
            mappy[i][j].h = FLT_MAX;

            /*if (isSingleDigit(mappy[i][j].hazardLevel))
                LocMap[i][j].hazardLevel = mappy[i][j].hazardLevel;
            else
                LocMap[i][j].hazardLevel = 0;*/
        }
```

```
    }
    mappy[start.r][start.c].prev_r = start.r;
    mappy[start.r][start.c].prev_c = start.c;
    mappy[start.r][start.c].g = 0.0;
    mappy[start.r][start.c].h = findDistance(start.r, start.c, end.r, end.c);
    mappy[start.r][start.c].f = mappy[start.r][start.c].g + mappy[start.r][start.c].h;
    mult = mappy[start.r][start.c].h;
    q.push(start);

    while (!q.empty())
    {
        double newG, newH, newF;
        Locations current = q.front();
        q.pop();
        //printf("current is at r = %d and c = %d from parent r = %d and c = %d\n",
current.r, current.c, current.prev_r, current.prev_c);
        visited[current.r][current.c] = true;

        if (isGoal(current, end))
        {
            //printf("The destination locations is reached from current at r = %d and c = %d
from parent r = %d and c = %d\n", current.r, current.c, current.prev_r, current.prev_c);
            //printPrevMap(mappy, length, width);
            path = buildPath(mappy, current);
            pathFound = true;
            freeBoolArray(visited);
            return path;
        }

        for (i = -1; i < 2; i++) // double loop is only 8 checks for adjacent locations
        {
            for (j = -1; j < 2; j++)
            {
                if (inBounds(current.r + i, current.c + j) && !isSame(current.r, current.c, i, j))
                {
                    if (isObstacle(mappy, current.r + i, current.c + j))
                        visited[current.r + i][current.c + j] = true;

                    if (!visited[current.r + i][current.c + j])
                    {
                        //printf("not an obstacle and visited at x = %d and y = %d\n", current.x +
i, current.y + j);
                        newG = mappy[current.r][current.c].g + 1.0;
                        newH = findDistance(current.r + i, current.c + j, end.r, end.c);
                        newF = ((newG + newH + mappy[current.r][current.c].f) / 2) +
(double)(mappy[current.r + i][current.c + j].hazardLevel);
```

```
                    //printf("at r = %d and c = %d newG = %lf and newH %lf and newF %lf
and old F %lf\n", current.r + i, current.c + j, newG, newH, newF, mappy[current.r +
i][current.c + j].hazardLevel);

                    if (mappy[current.r + i][current.c + j].f > newF)
                    {
                        mappy[current.r + i][current.c + j].g = newG;
                        mappy[current.r + i][current.c + j].h = newH;
                        mappy[current.r + i][current.c + j].f = newF;
                        mappy[current.r + i][current.c + j].prev_r = current.r;
                        mappy[current.r + i][current.c + j].prev_c = current.c;
                        q.push(mappy[current.r + i][current.c + j]);
                        //printf("adding to queue r = %d and c = %d\n", current.r + i, current.c
+ j);
                    }
                }
            }
        }
    }

    if (!pathFound)
    {
        printf("Failed to find the path to the Destination location\n");
        pathExists = false;
    }

    freeBoolArray(visited);
    //freeLocationsArray(LocMap);
    return path;
}

vector<Locations> FindLongPath(Locations initial, int neighborRows)
{
    int i, cur, left, center, right, next = 0, DryCount = 0, MSize = MList.size(), SenSize =
SList.size();
    bool needsWater = false;
    vector<Locations> LongPath;
    cur = FindStart(initial);
    Locations start(MList[cur].r, MList[cur].c);
    LongPath.push_back(initial);
    LongPath.push_back(start);
    dir = InitialDir(cur, neighborRows);
    //printf("current dir is %d\n", dir);

    bool* SReading = (bool*)malloc(sizeof(bool) * MSize);
    int* MPDry = (int*)malloc(sizeof(int) * MSize);
```

```
int* neighbors = (int*)malloc(sizeof(int) * 4);

for (i = 0; i < SenSize; i++) //set up midpoint dryness array
    SReading[i] = SList[i].wet;

for (i = 0; i < MSize; i++) //set up midpoint dryness array
    MPDry[i] = MList[i].dryness;

MPDry = DrynessCheck(SReading, MPDry, cur); // for initial position
SReading = SensorCheck(SReading, cur);
DryCount = countDry(MPDry);

while (DryCount > 0)
{
    //printf("current dir is %d at midpoint %d\n", dir, cur);
    neighbors = setNeighbors(cur, neighborRows, neighbors);
    left = neighbors[1];
    center = neighbors[2];
    right = neighbors[3];
    //for (i = 0; i < MSize; i++)
    //printf("dryness level at midpoint %d is %d\n", i, MPDry[i]);

    if ((MPDry[left] == MPDry[right]) && (MPDry[left] != 0)) // head to center
    {
        next = center;
        dir = FindDir(next, left, right);
        needsWater = true;
        if (atEdge(center, neighborRows))
            dir = (dir + 2) % 4;
    }
    else if (MPDry[left] > MPDry[right])  //head to left
    {
        next = left;
        dir = FindDir(next, left, right);
        needsWater = true;
        if (atEdge(left, neighborRows))
            dir = (dir + 2) % 4;
    }
    else if (MPDry[left] < MPDry[right])  // head to right
    {
        next = right;
        dir = FindDir(next, left, right);
        needsWater = true;
        if (atEdge(right, neighborRows))
            dir = (dir + 2) % 4;
    }
    if ((MPDry[left] == MPDry[right]) && (MPDry[left] == 0)) // look for nearest neighbor
```

```
        {
            //next = 3; //nearest neightbor here
            next = NearestDryDestination(neighbors, MPDry);
            //printf("need to put nearest neighbor algorithm here\n");
            needsWater = false;
            dir = FindDir(next, left, right);
            if (atEdge(next, neighborRows))
                dir = (dir + 2) % 4;
        }
        Locations NextMid(MList[next].r, MList[next].c);
        LongPath.push_back(NextMid);
        if(needsWater == true)
            WaterList.push_back(MList[next]);  //for destinations that need water

        cur = next;
        //printf("current dir is %d at midpoint %d\n", dir, cur);

        MPDry = DrynessCheck(SReading, MPDry, cur);  //update dryness
        SReading = SensorCheck(SReading, cur);
        DryCount = countDry(MPDry);
    }
    free(neighbors);
    free(MPDry);
    free(SReading);
    return LongPath;
}

int main()
{
    int j, k, waterIndex, neighborRows = 0, neighborCols = 0, LPLSize, MSize;
    bool obsDetect = false, obsDetectUltra = false;
    pathExists = true;
    Json::Reader reader;
    Json::Value obj;

    //int i, testCases,  sensorNum = 0,sid, srow, scol;
    //bool w;
    //Delay Here
    //int InitialTimeDelayer = usleep(60000000);

    commSetup();
    LidarSetup();

    //detectIMU();
    //enableIMU();
    //int compassOffSet = getCompassDirection();
```

```
    //LOW turns on led | HIGH turns on led
    pinMode(7,OUTPUT); //RED led
    pinMode(0,OUTPUT); //GREEN led
    pinMode(2,OUTPUT); //BLUE led
    pinMode(11,INPUT); //button

    //serialPutchar (fl, '0');  //set scanning to static
    //int StaticDelayer = usleep(500000);
    int prevButton = HIGH;
    for(;;)
    {
        if(prevButton == HIGH && digitalRead(11) == LOW)  // a falling edge
        {
            break;
        }
    }

    digitalWrite(0, HIGH); // turn green LED on
    digitalWrite(2, LOW); // turn blue LED on
    digitalWrite(7, LOW); // turn red LED on
    int LED = 0;  //label to show blue is on

    //cin >> testCases;
    //for (i = 0; i < testCases; i++)  //for multiple testcases
    //{
    //printf("testcase %d\n", i);
    rows = 0;  //initialize dimensions
    cols = 0;
    //setup parsing
    //cin >> sensorNum;
    //sensorNum = 6;  //hard coded testcase for summer without mesh network
                //cin >> rows;
    rows = 20;  //hard coded testcase for summer without mesh network
              //cin >> cols;
    cols = 30;  //hard coded testcase for summer without mesh network

    mappy = (Locations**)malloc(sizeof(Locations*) * rows); //alocate memory for the
map
    for (j = 0; j < rows; j++)
        mappy[j] = (Locations*)malloc(sizeof(Locations) * cols);

    for (j = 0; j < rows; j++) //initialize map to all 0's
        for (k = 0; k < cols; k++)
            mappy[j][k].hazardLevel = 0;

    //vector<Sensor> SList;
    SList.clear();
```

```
    /*for (j = 0; j < sensorNum; j++)   //for when mesh network can supply sensor data
    {
    cin >> sid;
    cin >> srow;
    cin >> scol;
    cin >> w;
    Sensor cur(sid, srow, scol, w);
    SList.push_back(cur);
    mappy[srow][scol].hazardLevel = 1000000 + sid;  //plot sensors on map with 1000000
flag
    }*/
    Locations InitialStart = getCurrentLoc(rows / 2, 0); //initial location of robot

    //hard codes sensor info for summer demo testcase without mesh network
    Sensor cur0(0, 5, 5, 1);  //sensor 0
    SList.push_back(cur0);
    mappy[5][5].hazardLevel = 1000000 + 0;  //plot sensors on map with 1000000 flag
    ifstream ifs0("ble/node_modules/bleacon/mint.json");
    reader.parse(ifs0, obj); // reader can also read strings
    SList[0].proxID = obj["uuid"].asString();
    SList[0].dist = obj["accuracy"].asDouble() * metersToFeet;
    SList[0].distOffset = SList[0].dist - findDistance(InitialStart.r, InitialStart.c,  SList[0].r,
SList[0].c);
    cout << "Sensor 0 proxID = " << SList[0].proxID << " and dist = " << SList[0].dist << "
and offset " << SList[0].distOffset << "\n";
    //SList[0].wet = true;
    //SList[0].wet = false;

    Sensor cur1(1, 15, 5, 1);  //sensor 1
    SList.push_back(cur1);
    mappy[15][5].hazardLevel = 1000000 + 1;  //plot sensors on map with 1000000 flag
    ifstream ifs1("ble/node_modules/bleacon/blueberry.json");
    reader.parse(ifs1, obj); // reader can also read strings
    SList[1].proxID = obj["uuid"].asString();
    SList[1].dist = obj["accuracy"].asDouble() * metersToFeet;
    SList[1].distOffset = SList[1].dist - findDistance(InitialStart.r, InitialStart.c,  SList[1].r,
SList[1].c);
    cout << "Sensor 1 proxID = " << SList[1].proxID << " and dist = " << SList[1].dist << "
and offset " << SList[1].distOffset << "\n";
    //SList[1].wet = true;
    //SList[1].wet = false;

    Sensor cur2(2, 5, 15, 0);  //sensor 2
    SList.push_back(cur2);
    mappy[5][15].hazardLevel = 1000000 + 2;  //plot sensors on map with 1000000 flag
    ifstream ifs2("ble/node_modules/bleacon/ice.json");
    reader.parse(ifs2, obj); // reader can also read strings
```

```
    SList[2].proxID = obj["uuid"].asString();
    SList[2].dist = obj["accuracy"].asDouble() * metersToFeet;
    SList[2].distOffset = SList[2].dist - findDistance(InitialStart.r, InitialStart.c,  SList[2].r,
SList[2].c);
    cout << "Sensor 2 proxID = " << SList[2].proxID << " and dist = " << SList[2].dist << "
and offset " << SList[2].distOffset << "\n";
    //SList[2].wet = true;
    //SList[2].wet = false;

    Sensor cur3(3, 15, 15, 1);  //sensor 3
    SList.push_back(cur3);
    mappy[15][15].hazardLevel = 1000000 + 3;  //plot sensors on map with 1000000 flag
    ifstream ifs3("ble/node_modules/bleacon/coconut.json");
    reader.parse(ifs3, obj); // reader can also read strings
    SList[3].proxID = obj["uuid"].asString();
    SList[3].dist = obj["accuracy"].asDouble() * metersToFeet;
    SList[3].distOffset = SList[3].dist - findDistance(InitialStart.r, InitialStart.c,  SList[3].r,
SList[3].c);
    cout << "Sensor 3 proxID = " << SList[3].proxID << " and dist = " << SList[3].dist << "
and offset " << SList[3].distOffset << "\n";
    //SList[3].wet = true;
    //SList[3].wet = false;

    Sensor cur4(4, 5, 25, 0);  //sensor 4
    SList.push_back(cur4);
    mappy[5][25].hazardLevel = 1000000 + 4;  //plot sensors on map with 1000000 flag
    ifstream ifs4("ble/node_modules/bleacon/mint2.json");
    reader.parse(ifs4, obj); // reader can also read strings
    SList[4].proxID = obj["uuid"].asString();
    SList[4].dist = obj["accuracy"].asDouble() * metersToFeet;
    SList[4].distOffset = SList[4].dist - findDistance(InitialStart.r, InitialStart.c,  SList[4].r,
SList[4].c);
    cout << "Sensor 4 proxID = " << SList[4].proxID << " and dist = " << SList[4].dist << "
and offset " << SList[4].distOffset << "\n";
    //SList[4].wet = true;
    //SList[4].wet = false;

    Sensor cur5(5, 15, 25, 0);  //sensor 5
    SList.push_back(cur5);
    mappy[15][25].hazardLevel = 1000000 + 5;  //plot sensors on map with 1000000 flag
    ifstream ifs5("ble/node_modules/bleacon/blueberry2.json");
    reader.parse(ifs5, obj); // reader can also read strings
    SList[5].proxID = obj["uuid"].asString();
    SList[5].dist = obj["accuracy"].asDouble() * metersToFeet;
    SList[5].distOffset = SList[5].dist - findDistance(InitialStart.r, InitialStart.c,  SList[5].r,
SList[5].c);
```

```
    cout << "Sensor 5 proxID = " << SList[5].proxID << " and dist = " << SList[5].dist << "
and offset " << SList[5].distOffset << "\n";
    //SList[5].wet = true;
    //SList[5].wet = false;
    //end of hard coded summer testcase data

    //cin >> neighborRows;
    neighborRows = 2;
    //cin >> neighborCols;
    neighborCols = 3;

    MList = findMidpoints(neighborRows, neighborCols);
    MSize = MList.size();
    //printf("/n MSize = %d \n", MSize);
    for (j = 0; j < MSize; j++)  //plot midpoints on map with 2000000 flag
        mappy[MList[j].r][MList[j].c].hazardLevel = 2000000 + MList[j].id;

    vector<Locations> LongPathList = FindLongPath(InitialStart, neighborRows);
    //printLPList(LongPathList);

    LPLSize = LongPathList.size();
    Locations curLoc = InitialStart, nextLoc = InitialStart, dectectedCur, goal =
LongPathList[LPLSize - 1]; //start from initial location, end at last midpoint
    vector<Locations> ShortPath, ReversePath;
    int robotDirection = 0, obstacleDirection = 0, obstacleDist = 1000, ultraDist = 1000,
lastDist;

    bool toggleLed = true;
    for(j = 0; j < 360; j++) //intial sweep here
    {
        if(toggleLed == true)
            digitalWrite(0, LOW); // turn green LED off
        else
            digitalWrite(0, HIGH); // turn green LED on

        toggleLed = !toggleLed; // toggle LED

        getObsDataArr();
        obstacleDirection = obstacleDetect.array[0] - 90;
        obstacleDirection *= -1;
        obstacleDist = obstacleDetect.array[1];
        //ultraDist = obstacleDetect.array[2];

        if(obstacleDist > 10)
            lastDist = obstacleDist;

        if(obstacleDist < 10)
```

```
            obstacleDist += lastDist;

        double DDist = ((double)obstacleDist / cmToFeet);
        obstacleDist = (int)DDist;
        //ultraDist /= cmToFeet;
        printf("obstacleDirection = %d, obstacleDist = %d, ult = %d\n", obstacleDirection,
obstacleDist, obstacleDetect.array[2]);
        if(obstacleDist > 1)
            obsDetect = findObstacleLoc(curLoc.r, curLoc.c, robotDirection, obstacleDist,
obstacleDirection); //for lidar

        //Ultra Hedgehog The Sonic, Gotta have them ALL!
        //obsDetect = findObstacleLoc(curLoc.r, curLoc.c, robotDirection, ultraDist, 0); //for
ultrasonic


        int ScanTimeDelayer = usleep(20000);
    }

    serialPutchar (fl, '0');  // set scanning to dynamic
    for(j = 0; j < 10; j++)
    {
        if(toggleLed == true)
            digitalWrite(0, LOW); // turn green LED off
        else
            digitalWrite(0, HIGH); // turn green LED on

        toggleLed = !toggleLed; // toggle LED

        int DynamicDelayer = usleep(1100000);
    }
    //fflush(stdout);

    //int ScanTimeDelayer = usleep(3600000);
    LED = 0;
    digitalWrite(0, HIGH); // turn green LED on
    //mappy = updatemap(17, 17);
    //mappy = updatemap(18, 17);
    //mappy = updatemap(17, 18);
    //mappy = updatemap(8, 7);

    //printf("cur is at row %d and col %d\n", curLoc.r, curLoc.c);
    //printf("goal is at row %d and col %d\n", goal.r, goal.c);

    bool stop = false, accel = false, watered = false, hasWaterSpot = false;
    waterIndex = 0;
    Midpoint toWater;
```

```
    int WSize = WaterList.size();
    if(waterIndex < WSize)
    {
        toWater = WaterList[waterIndex];
        hasWaterSpot = true;
    }

    k = 0;
    serialPutchar (fd, '9');
    int travelTimeDelayer = usleep(500000);
    while (!sameLocation(curLoc, goal))  //until goal is reached
    {
        //printf("entering loop\n");
        obsDetect = false; //set new obstacle detected to false to procede
        //obsDetectUltra = false;
        //printf("starting AStar\n");
        printMap();
        ShortPath = AStarSearch(LongPathList[k], curLoc);  //build path from current
location to next midpoint
        if((pathExists == false) || (obsDetectUltra == true))
        {
            printf("path doesn't exist\n");
            serialPutchar (fd, '0');
            printMap();

            if(LED == 0)   // if green is on
            {
                digitalWrite(0, LOW);  // turn off green
                digitalWrite(7, HIGH);   // turn on red
                LED = 7;               //label to show red is on
            }

            if(LED == 2)  // if blue is on
            {
                digitalWrite(2, LOW);  // turn off blue
                digitalWrite(7, HIGH);   // turn on red
                LED = 7;               //label to show red is on
            }
            freeLocationsArray(mappy);
            return 0;
        }
        //printf("ending AStar\n");
        int SPSize = ShortPath.size();

        //serialPutchar (fd, '9');
        //accel = true;
```

```
j = 1;
while (j < SPSize)  //itterate through each location in ShortPath
{
   if(hasWaterSpot == true)
   {
      double waterDist = findDistance(curLoc.r, curLoc.c, toWater.r, toWater.c);

      if(sameCoordinate(curLoc.r, curLoc.c, toWater.r, toWater.c) == true)
      {
         serialPutchar (fd, '0');
         int waterDelayer = usleep(3000000);  //stop to water
         serialPutchar (fd, '9');
         //int travelTimeDelayer = usleep(500000); // to compensate for start delay
      }

      if(waterDist < SprinklerRadius)
      {
         printf("watering midpoint at (%d, %d)\n", toWater.r, toWater.c);
         if(LED == 0)   // if green is on
         {
            digitalWrite(0, LOW);  // turn off green
            digitalWrite(2, HIGH);   // turn on blue
            LED = 2;              //label to show blue is on
         }

         if(LED == 7)  // if red is on
         {
            digitalWrite(7, LOW);  // turn off red
            digitalWrite(2, HIGH);   // turn on blue
            LED = 2;              //label to show blue is on
         }
         watered = true;
      }
      else
      {
         if(LED == 2)   // if blue is on
         {
            digitalWrite(2, LOW);  // turn off blue
            digitalWrite(0, HIGH);   // turn on green
            LED = 0;              //label to show green is on
         }

         if(LED == 7)  // if red is on
         {
            digitalWrite(7, LOW);  // turn off red
            digitalWrite(0, HIGH);   // turn on green
            LED = 0;              //label to show green is on
```

```
        }

        if(watered == true)
        {
            watered = false;
            waterIndex++;
            if(waterIndex < WSize)
                toWater = WaterList[waterIndex];
        }
    }
}

nextLoc = ShortPath[j]; //select next location in path
                    //if (j < SPSize - 1)
                    //nextLoc = ShortPath[j];

//to get speed and angle for drive controls
int rDiff = curLoc.r - nextLoc.r;
int cDiff = nextLoc.c - curLoc.c;
int DriveAngle = getDriveAngle(rDiff, cDiff, robotDirection);
float travelDist = (float)sqrt((rDiff * rDiff) + (cDiff * cDiff));
//printf("starting top of loop drive angle == %d\n", DriveAngle);

//turning
//char t = getAngleChar(DriveAngle);
if((DriveAngle >= 124) && (135 >= DriveAngle))
{
    printf("hard right\n");
    serialPutchar (fd, '2');//for 135 degrees
}

if((DriveAngle >= 101) && (124 > DriveAngle))
{
    printf("soft right\n");
    serialPutchar (fd, '3');//for 112 degrees
}

if((DriveAngle >= 79) && (101 > DriveAngle))
{
    serialPutchar (fd, '4');//for 90 degrees
}

if((DriveAngle > 56) && (79 > DriveAngle))
{
    printf("soft left\n");
    serialPutchar (fd, '5');//for 68 degrees
}
```

```
        if((DriveAngle >= 45) && (56 >= DriveAngle))
        {
            printf("hard left\n");
            serialPutchar (fd, '6');//for 45 degrees
        }

        if (accel == true)
        {
            if(stop == false) //go
            {
                //Ramp up Speed
                serialPutchar (fd, '9');
                //serialPutchar (fd, '9'); // second one to deal with inconsistent motor
                printf("moving forward\n");
                //
                //accChange = true;
            }
            else if (stop == true) //stop
            {
                //Ramp down Speed
                serialPutchar (fd, '0');
                //
                //accChange = true;
            }
        }

        //if(accChange == true)
        accel = false;
        //int robotDirection = 180;  //find direction
        //compass goes here
        robotDirection = getPsuedoAngle(curLoc, nextLoc, robotDirection);  //calculate
(or get) current direction

        /*robotDirection = getCompassDirection() - compassOffSet;

        if(robotDirection < 0)
            robotDirection += 360;

        if(robotDirection > 360)
            robotDirection -= 360;
        */

        //cout << "robotDirection = " << robotDirection  << "\n";

        //printf("robotDirection angle == %d\n", robotDirection);
```

```
        //get sensor distance
        int i; //sensor0
        double proxDistAve = 0.0;
        for(i = 0; i < 5; i++)
        {
            ifstream uifs0("ble/node_modules/bleacon/mint.json");
            reader.parse(uifs0, obj);
            proxDistAve += obj["accuracy"].asDouble();
        }
        SList[0].dist = ((proxDistAve / 5.0) * metersToFeet) - SList[0].distOffset;
        cout << "updating Sensor 0 proxID = " << SList[0].proxID << " and dist = " <<
SList[0].dist << "\n";

        proxDistAve = 0.0; //sensor1
        for(i = 0; i < 5; i++)
        {
            ifstream uifs1("ble/node_modules/bleacon/blueberry.json");
            reader.parse(uifs1, obj);
            proxDistAve += obj["accuracy"].asDouble();
        }
        SList[1].dist = ((proxDistAve / 5.0) * metersToFeet)  - SList[1].distOffset;
        cout << "updating Sensor 1 proxID = " << SList[1].proxID << " and dist = " <<
SList[1].dist << "\n";

        proxDistAve = 0.0; //sensor2
        for(i = 0; i < 5; i++)
        {
            ifstream uifs2("ble/node_modules/bleacon/ice.json");
            reader.parse(uifs2, obj);
            proxDistAve += obj["accuracy"].asDouble();
        }
        SList[2].dist = ((proxDistAve / 5.0) * metersToFeet) - SList[2].distOffset;
        cout << "updating Sensor 2 proxID = " << SList[2].proxID << " and dist = " <<
SList[2].dist << "\n";

        proxDistAve = 0.0; //sensor3
        for(i = 0; i < 5; i++)
        {
            ifstream uifs3("ble/node_modules/bleacon/coconut.json");
            reader.parse(uifs3, obj);
            proxDistAve += obj["accuracy"].asDouble();
        }
        SList[3].dist = ((proxDistAve / 5.0) * metersToFeet) - SList[3].distOffset;
        cout << "updating Sensor 3 proxID = " << SList[3].proxID << " and dist = " <<
SList[3].dist << "\n";

        proxDistAve = 0.0; //sensor4
```

```
        for(i = 0; i < 5; i++)
        {
            ifstream uifs4("ble/node_modules/bleacon/mint2.json");
            reader.parse(uifs4, obj);
            proxDistAve += obj["accuracy"].asDouble();
        }
        SList[4].dist = ((proxDistAve / 5.0) * metersToFeet) - SList[4].distOffset;
        cout << "updating Sensor 4 proxID = " << SList[4].proxID << " and dist = " <<
SList[4].dist << "\n";

        proxDistAve = 0.0; //sensor5
        for(i = 0; i < 5; i++)
        {
            ifstream uifs5("ble/node_modules/bleacon/blueberry2.json");
            reader.parse(uifs5, obj);
            proxDistAve += obj["accuracy"].asDouble();
        }
        SList[5].dist = ((proxDistAve / 5.0) * metersToFeet) - SList[5].distOffset;
        cout << "updating Sensor 5 proxID = " << SList[5].proxID << " and dist = " <<
SList[5].dist << "\n";

        //depends on proximity sensors
        dectectedCur = getProxSensorLoc();   //to ensure the vehicle in on course using
proximity sensors
        if((dectectedCur.r < 0) || (dectectedCur.r >= rows))
            dectectedCur.r = curLoc.r;

        if((dectectedCur.c < 0) || (dectectedCur.c >= cols))
            dectectedCur.c = curLoc.c;
        double detectDist = findDistance(dectectedCur.r, dectectedCur.c,  curLoc.r,
curLoc.c);
        cout << "detectect location at (" << dectectedCur.r << ", " << dectectedCur.c <<
")\n";
        /*
        if((curLoc.r != dectectedCur.r) && (curLoc.c != dectectedCur.c) && (detectDist <
2))
        {
            printf("at wrong location, changing course\n");

            cout << "detectect location at (" << dectectedCur.r << ", " << dectectedCur.c
<< ")\n";
            curLoc = dectectedCur;
            serialPutchar (fd, '0');
            accel = true;
            break;
        }
        else
```

```
        {
            cout << "detectect location at (" << dectectedCur.r << ", " << dectectedCur.c
<< ")\n";
        }*/

        ReversePath.push_back(curLoc);  //for back tracking once goal is reached

        //
        //tell vehicle drive controls to go to nextLoc from curLoc
        //
        curLoc = nextLoc;
        if(mappy[curLoc.r][curLoc.c].hazardLevel < 1000000)
            mappy[curLoc.r][curLoc.c].hazardLevel += 3000000;  //mark as visited

        //printf("moving to (%d, %d) in direction %d ", curLoc.r, curLoc.c, robotDirection);
        printf("(%d, %d) at %d at speed %f turning %d \n", curLoc.r, curLoc.c,
robotDirection, travelDist, DriveAngle);
        //printf(" at (%d, %d) -> ", curLoc.r, curLoc.c);

        int p, q, r, numTest, AveCount = 0, UltraAve = 0;
        char highBits, lowBits;
        fflush(stdout);
        for(r = 0; r < 25; r++) // 5 * 5 = 25 for 5 averages
        {
            UltraAve = 0;
        for(p = 0; p < 10; p++)   // 2 lidar/ultra scans
        {
            fflush(stdout);
            for(q = 0; q < 3; q++)  // 1 for angle, distance, and ultra
            {
                if(serialDataAvail (fl))
                {
                    highBits = serialGetchar (fl);
                    lowBits = serialGetchar (fl);

                    numTest = lowBits | highBits << 8;

                    //setvalue depending on position in array
                    obstacleDetect.array[(obstacleDetect.index) % 3] = numTest;
                    //printf("lidar obstacleDirection numTest = %d\n", numTest);
                    //increment the index
                    (obstacleDetect.index)++;

                    //printf("%d\n", numTest);
                    //printf("%c", highBits);
                    //fflush(stdout);
                }
```

```
            //printf("raw data obstacleDetect[%d] = %d\n", q, obstacleDetect.array[q]);


        }

        //obstacleDirection = obstacleDetect.array[0] - 90;
        //obstacleDirection *= -1;
        //obstacleDist = obstacleDetect.array[1];
        //ultraDist = obstacleDetect.array[2];
        int rawUltra;
        int l;
        for(l = 0; l < 3; l++)
        {
            if((obstacleDetect.array[l] != 90) && (obstacleDetect.array[l] != 25000))
                rawUltra = obstacleDetect.array[l];
        }
        printf("raw ultraDist = %d\n", rawUltra);
        //ultraDist = numTest;

        //if(obstacleDirection < 0)
        //obstacleDirection += 360;

        //if(obstacleDist > 10)
            //lastDist = obstacleDist;

        //if(obstacleDist < 10)
            //obstacleDist += lastDist;

        //obstacleDist /= cmToFeet;
        //ultraDist /= cmToFeet;

        //printf("obstacleDirection = %d, obstacleDist = %d, ultraDist = %d\n",
obstacleDirection, obstacleDist, ultraDist);
        //if(obstacleDist < 4)
            //obsDetect = findObstacleLoc(curLoc.r, curLoc.c, robotDirection,
obstacleDist, obstacleDirection); //for lidar

        UltraAve += rawUltra;
        /*if(ultraDist > 2) // remove false zero values
        {
            ultraDist /= cmToFeet;
            printf("ultraDist = %d\n", ultraDist);
            if(ultraDist < 4)
            {
                obsDetectUltra = findObstacleLoc(curLoc.r, curLoc.c, robotDirection,
ultraDist, 0); //for ultrasonic
                //printf("ultraDist = %d\n", ultraDist);
```

```
                //ultraDet = true;
            }
        }*/
        //int DtravelTimeDelayer = usleep(20000);

    }
    ultraDist = (UltraAve / 10);


    if(ultraDist > 2) // remove false zero values
    {
        ultraDist /= cmToFeet;
        printf("detected ultraDist = %d\n", ultraDist);
        if(ultraDist < 4)
        {
            obsDetectUltra = findObstacleLoc(curLoc.r, curLoc.c, robotDirection,
ultraDist, 0); //for ultrasonic
            //printf("ultraDist = %d\n", ultraDist);
            //ultraDet = true;
        }
    }
    if (obsDetectUltra ) //stop moving along path and recalculate
    {
        printf("found obstacle\n");
        serialPutchar (fd, '0');
        accel = true;
        break;
    }
    travelTimeDelayer = usleep(10000); // 0.1 second per ultra scan
    } //for the 5 * 5 = 25
    //travelTimeDelayer = usleep(500000); // 0.1 second per ultra scan

    //travelTimeDelayer = usleep(250000);
    //nanosleep(1000000000);


    if (obsDetectUltra ) //stop moving along path and recalculate
    {
        printf("found obstacle\n");
        serialPutchar (fd, '0');
        accel = true;
        break;
    }

    /*if (obsDetect) //stop moving along path and recalculate
    {
        printf("found obstacle\n");
```

```
            serialPutchar (fd, '0');
            accel = true;
            break;
        }*/

        /*if (obsDetect || obsDetectUltra)//stop moving along path and end program
        {
            printf("found obstacle ending path\n");
            serialPutchar (fd, '0');
            accel = true;

            serialPutchar (fd, '4');//for 90 degrees
            serialPutchar (fd, '0'); // to stop
            printMap();
            if(LED == 0)   // if green is on
            {
                digitalWrite(0, LOW);  // turn off green
                digitalWrite(7, HIGH);   // turn on red
                LED = 7;               //label to show red is on
            }

            if(LED == 2)  // if blue is on
            {
                int waterDelayerEnd = usleep(5000000);
                digitalWrite(2, LOW);  // turn off blue
                digitalWrite(7, HIGH);   // turn on red
                LED = 7;               //label to show red is on
            }
            freeLocationsArray(mappy);
            return 0; // exit program
        }*/

        //travelTimeDelayer = usleep(250000);
        //nanosleep(1000000000);

        j++;
    }
    if (!obsDetect) //if no obstacle detected
        k++; //increment to next midpoint

    //serialPutchar (fd, '9');
    printf("\n");
}
ReversePath.push_back(curLoc); //add last Location (goal)

//Ramp down Speed
serialPutchar (fd, '4');//for 90 degrees
```

```
    serialPutchar (fd, '0');

    if(LED == 0)   // if green is on
    {
        digitalWrite(0, LOW);  // turn off green
        digitalWrite(7, HIGH);   // turn on red
        LED = 7;               //label to show red is on
    }

    if(LED == 2)  // if blue is on
    {
        int waterDelayerEnd = usleep(5000000);
        digitalWrite(2, LOW);  // turn off blue
        digitalWrite(7, HIGH);   // turn on red
        LED = 7;               //label to show red is on
    }

    // read signal
    /*if(serialDataAvail (fd))
    {
        char newChar = serialGetchar (fd);
        printf("%c", newChar);
        fflush(stdout);
    }*/

    /*retractable hose needed for this to not cause problems (Mechanical Team)
    for (j = ReversePath.size() - 1; j > -1; j--)  //backtracking to chargining station
    {
        curLoc = ReversePath[j];
        //
        //tell vehicle drive controls to go in reverse from goal to InitialStart
        //
        printf("(%d, %d) ", curLoc.r, curLoc.c);
    }*/

    //printf("after mods\n");
    printMap();
    freeLocationsArray(mappy);  //frees memory used for global map
    fflush(stdout);

    return 0;
}
#endif
```

Program 1: LongPath2.cpp

```
#ifdef RaspberryPi // just that the Arduino IDE doesnt compile these files.

#include <bits/stdc++.h>
#include <iostream>
#include <stdio.h>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <queue>
#include <cmath>
#include <unistd.h>
#include <string.h> //for errno
#include <errno.h> //error output
#include <stdint.h> //uint8_t definitions
#include <fcntl.h>
#include <termios.h>
#include <fstream>
#include <wiringPi.h> //wiring Pi
#include <wiringSerial.h>

using namespace std;

#define PI 3.14159265
#define cmToFeet 30.48
#define metersToFeet 3.28084
#define SprinklerRadius 3.0

#define magXmax 2378
#define magYmax 775
#define magZmax -279
#define magXmin -1282
#define magYmin -322
#define magZmin -1268

//class definitions
class Locations;
class Sensor;
class Midpoint;

//struct definitions
struct obstacleData
{
    int array[3];
    int index;

};
```

```
//global variables and structures
struct obstacleData obstacleDetect;
Locations** mappy;
int dir;
int rows;
int cols;
bool pathExists;

// Find Serial device on Raspberry with ~ls /dev/tty*
// ARDUINO_UNO "/dev/ttyACM0"
// FTDI_PROGRAMMER "/dev/ttyUSB0"
// HARDWARE_UART "/dev/ttyAMA0"
char device[]= "/dev/ttyACM0";  //drive
// filedescriptor
int fd;    // drive

char Lidar[]= "/dev/ttyS0";  // lidar
// filedescriptor
int fl;      // lidar

unsigned long baud = 9600;
unsigned long highBaud = 115200;
//unsigned long time=0;

//classes
class Locations
{

public:

    int r, c, hazardLevel, prev_r, prev_c;
    double f, g, h;

    Locations();
    Locations(int, int);
    void setCoords(int, int);
    ~Locations();
};

//constructors
Locations::Locations()
{
    r = 0;
    c = 0;
}

Locations::Locations(int rLoc, int cLoc)
```

```
{
    setCoords(rLoc, cLoc);
    prev_r = r;
    prev_c = c;
    f = 0.0;
    g = 0.0;
    h = 0.0;
}

void Locations::setCoords(int rLoc, int cLoc)
{
    r = rLoc;
    c = cLoc;
}

Locations::~Locations()
{
    ;
}

//helper function prototypes
char * serialGets(char *buf, const int n, const int fg);

int FindStart(Locations cur);

int calcMid(int n1, int n2);

int getPsuedoAngle(Locations cur, Locations next, int curAng);

int getDriveAngle(int rDiff, int cDiff, int ang);

void getObsDataArr();

Locations** updatemap(int tRow, int tCol);

double findDistance(int curR, int curC, int destR, int destC);

bool findObstacleLoc(int VRow, int VCol, int VAng, int obDist, int obAng);

bool inBounds(int row, int col);

bool isSame(int row, int col, int i, int j);

bool sameLocation(Locations curLoc, Locations goal);

bool sameCoordinate(int crow, int ccol, int grow, int gcol);

bool isGoal(Locations current, Locations Goal);
```

```
bool isObstacle(Locations** LocMap, int row, int col);

void commSetup();

void LidarSetup();

void printSList();

void printMap();

void freeIntArray(int** arr);

void freeBoolArray(bool** arr);

void freeLocationsArray(Locations** arr);

Locations getCurrentLoc(int r, int c);

vector<Locations>buildPath(Locations** LocMap, Locations end);

vector<Locations> AStarSearch(Locations start, Locations end);

//helper functions
char * serialGets(char *buf, const int n, const int fg)
{
    int m;
    m = read(fg, buf, n);
    *(buf + m) = '\0';
    return (buf);
}

int calcMid(int n1, int n2)
{
    int m = 0;
    m = (n1 + n2) / 2;
    return m;
}

int getPsuedoAngle(Locations cur, Locations next, int curAng)
{
    if (sameLocation(cur, next))
        return curAng;

    int rDiff = cur.r - next.r;
    int cDiff = next.c - cur.c;
    int calcAng = (int)((atan2(rDiff, cDiff) * 180) / PI);
    //printf(" rDiff = %d, cDiff = %d, calcAng = %lf ", rDiff, cDiff, calcAng);
```

```
    if (calcAng < 0)
        calcAng += 360;

    if (calcAng != curAng)
        return calcAng;

    return curAng;
}

int getDriveAngle(int rDiff, int cDiff, int ang)
{
    int turnAng;
    int posAng = (int)((180 * atan2(rDiff, cDiff)) / PI);

    if (posAng < 0)
        posAng += 360;
    if (ang < 0)
        ang += 360;

    //printf("current angle at %d pos angle at %d\n", ang, posAng);

    turnAng = ang - posAng;
    turnAng = (turnAng / 2) + 90;
    if (turnAng < 0)
        turnAng += 180;

    if (turnAng > 135)
        turnAng -= 180;

    return turnAng;
}

void getObsDataArr()
{
    fflush(stdout);
    int q, numTest;
    char highBits, lowBits;
    for(q = 0; q < 3; q++)
    {
        if(serialDataAvail (fl))
        {
            highBits = serialGetchar (fl);
            lowBits = serialGetchar (fl);

            numTest = lowBits | highBits << 8;

            //setvalue depending on position in array
```

```
            obstacleDetect.array[(obstacleDetect.index) % 3] = numTest;
            //printf("lidar obstacleDirection numTest = %d\n", numTest);
            //increment the index
            (obstacleDetect.index)++;

            //printf("%d\n", numTest);
            //printf("%c", highBits);
            //fflush(stdout);
        }
        //printf("raw data obstacleDetect[%d] = %d\n", q, obstacleDetect.array[q]);
    }
}

Locations** updatemap(int tRow, int tCol)
{
    int i, j, p, q, count;
    mappy[tRow][tCol].hazardLevel = 9;
    for (i = -1; i<2; i++)
    {
        for (j = -1; j<2; j++)
        {
            if ((inBounds(tRow + i, tCol + j) && !(isSame(tRow, tCol, i, j))))
            {
                if ((mappy[tRow + i][tCol + j].hazardLevel != 9) && (mappy[tRow + i][tCol +
j].hazardLevel < 1000000))
                {
                    count = 0;
                    for (p = -1; p <2; p++)
                    {
                        for (q = -1; q <2; q++)
                        {
                            if ((inBounds(tRow + i + p, tCol + j + q) && !(isSame(tRow + i, tCol + j,
p, q))))
                            {
                                if (mappy[tRow + i + p][tCol + j + q].hazardLevel == 9)
                                {
                                    count++;
                                }
                            }
                        }
                    }
                    mappy[tRow + i][tCol + j].hazardLevel = count;

                }
            }

        }
```

```
    }

    return mappy;
}

double findDistance(int curR, int curC, int destR, int destC)
{
    //distance formula
    double rDiff = (double)curR - (double)destR;
    double cDiff = (double)curC - (double)destC;
    double rSquare = rDiff * rDiff;
    double cSquare = cDiff * cDiff;
    double squareSum = rSquare + cSquare;
    double dist = ((double)sqrt(squareSum));
    return dist;
}

bool findObstacleLoc(int VRow, int VCol, int VAng, int obDist, int obAng)
{
    int ORow, OCol, tRow = 0, tCol = 0;
    bool updated = false;
    double RelativeAngle = (double)((VAng + obAng) % 360);
    ORow = (int)round((double)(obDist * sin(RelativeAngle * (PI / 180))));
    OCol = (int)round((double)(obDist * cos(RelativeAngle * (PI / 180))));
    //printf("obstace distance coords: ORow = %d OCol = %d\n", tRow, tCol);

    tRow = VRow - ORow;
    //printf("obstace location coords: tRow = %d ", tRow);
    tCol = VCol + OCol;
    //printf("tCol = %d\n", tCol);

    if (inBounds(tRow, tCol))
    {
        if ((mappy[tRow][tCol].hazardLevel) != 9 && (mappy[tRow][tCol].hazardLevel <
1000000))
        {
            mappy = updatemap(tRow, tCol);
            updated = true;
        }
    }
    return updated;
}

bool isSingleDigit(int num)
{
    if (num > -1 && num < 10)
```

```
        return true;
    else
        return false;
}

bool inBounds(int row, int col)
{
    if ((row > -1) && (row < rows) && (col > -1) && (col < cols))
        return true;
    else
        return false;
}

bool isSame(int row, int col, int i, int j) //determine of locations are the same based on
itteration
{
    if (((row + i) == row) && ((col + j) == col))
        return true;
    else
        return false;
}

bool sameLocation(Locations curLoc, Locations goal) //determining if two defined
locations are the same
{
    if ((curLoc.r == goal.r) && (curLoc.c == goal.c))
        return true;

    return false;
}

bool sameCoordinate(int crow, int ccol, int grow, int gcol) //determining if two locations
are the same by coordinates
{
    if ((crow == grow) && (ccol == gcol))
        return true;

    return false;
}

bool isGoal(Locations current, Locations Goal)
{
    if ((current.r == Goal.r) && (current.c == Goal.c))
        return true;
    else
        return false;
}
```

```
bool isObstacle(Locations** LocMap, int row, int col)
{
   //printf("checking obstacles at r = %d c = %d and hlevel at %d\n", row, col,
LocMap[row][col].hazardLevel);
   if (LocMap[row][col].hazardLevel == 9)
      return true;
   else
      return false;
}

void commSetup()
{

  printf("%s \n", "Raspberry Startup!");
  fflush(stdout);

  //get filedescriptor
  if ((fd = serialOpen (device, baud)) < 0){
    fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
    exit(1); //error
  }

  //setup GPIO in wiringPi mode
  if (wiringPiSetup () == -1){
    fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
    exit(1); //error
  }

}

void LidarSetup()
{

   //initialize array and index to 0
   int i;
   for(i = 0; i < 3; i++)
   {
      obstacleDetect.array[i] = 0;
   }
   obstacleDetect.index = 0;

   printf("%s \n", "Raspberry Startup!");
   fflush(stdout);

   //get filedescriptor
   // ((fl = serialOpen (Lidar, baud)) < 0)
```

```
    if ((fl = serialOpen (Lidar, highBaud)) < 0)
    {
        fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
        exit(1); //error
    }

    //setup GPIO in wiringPi mode
    if (wiringPiSetup () == -1)
    {
        fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
        exit(1); //error
    }

}

void printMap()
{
    int i, j;
    printf("\n printing map \n");
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            int val = mappy[i][j].hazardLevel;
            if(val < 1000000)
                printf("%d ", val);
            else
            {
                if(val < 2000000)
                {
                    //val -= 1000000;
                    printf("S ");
                }
                else if(val >= 3000000)
                {
                    //val -= 3000000;
                    printf("T ");
                }
                else
                {
                    //val -= 2000000;
                    printf("M ");
                }
            }
        }
        printf("\n");
    }
```

```
}

void freeBoolArray(bool** arr)
{
    int i;
    for (i = 0; i < rows; i++)
        free(arr[i]);

    free(arr);
}

void freeLocationsArray(Locations** arr)
{
    int i;
    for (i = 0; i < rows; i++)
        free(arr[i]);

    free(arr);
}

Locations getCurrentLoc(int r, int c)
{
    //later modify to get coordinate inputs of current location from mesh network
    Locations cur(r, c);
    return cur;
}

vector<Locations>buildPath(Locations** LocMap, Locations end)
{
    vector<Locations> path;

    //printPrevMap(mappy, length, width);
    int row = end.r, col = end.c, temp;
    while (!(LocMap[row][col].prev_r == row && LocMap[row][col].prev_c == col))
    {
        path.push_back(LocMap[row][col]);
        temp = row;
        row = LocMap[row][col].prev_r;
        col = LocMap[temp][col].prev_c;
    }
    path.push_back(LocMap[row][col]);
    return path;
}

vector<Locations> AStarSearch(Locations start, Locations end)
{
    int i, j;
```

```
    bool pathFound = false;
    vector<Locations> path, BestPath;
    queue<Locations> q;

    bool** visited = (bool**)malloc(sizeof(bool*) * rows);
    for (int j = 0; j < rows; j++)
        visited[j] = (bool*)malloc(sizeof(bool) * cols);

    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            visited[i][j] = false;
            mappy[i][j].r = i;
            mappy[i][j].c = j;
            mappy[i][j].prev_r = -1;
            mappy[i][j].prev_c = -1;
            mappy[i][j].f = FLT_MAX;
            mappy[i][j].g = FLT_MAX;
            mappy[i][j].h = FLT_MAX;

        }
    }
    mappy[start.r][start.c].prev_r = start.r;
    mappy[start.r][start.c].prev_c = start.c;
    mappy[start.r][start.c].g = 0.0;
    mappy[start.r][start.c].h = findDistance(start.r, start.c, end.r, end.c);
    mappy[start.r][start.c].f = mappy[start.r][start.c].g + mappy[start.r][start.c].h;
    q.push(start);

    while (!q.empty())
    {
        double newG, newH, newF;
        Locations current = q.front();
        q.pop();
        //printf("current is at r = %d and c = %d from parent r = %d and c = %d\n",
current.r, current.c, current.prev_r, current.prev_c);
        visited[current.r][current.c] = true;

        if (isGoal(current, end))
        {
            //printf("The destination locations is reached from current at r = %d and c = %d
from parent r = %d and c = %d\n", current.r, current.c, current.prev_r, current.prev_c);
            path = buildPath(mappy, current);
            pathFound = true;
            freeBoolArray(visited);
            return path;
```

```
        }

        for (i = -1; i < 2; i++) // double loop is only 8 checks for adjacent locations
        {
            for (j = -1; j < 2; j++)
            {
                if (inBounds(current.r + i, current.c + j) && !isSame(current.r, current.c, i, j))
                {
                    if (isObstacle(mappy, current.r + i, current.c + j))
                        visited[current.r + i][current.c + j] = true;

                    if (!visited[current.r + i][current.c + j])
                    {
                        //printf("not an obstacle and visited at x = %d and y = %d\n", current.x +
i, current.y + j);
                        newG = mappy[current.r][current.c].g + 1.0;
                        newH = findDistance(current.r + i, current.c + j, end.r, end.c);
                        newF = ((newG + newH + mappy[current.r][current.c].f) / 2) +
(double)(mappy[current.r + i][current.c + j].hazardLevel);
                        //printf("at r = %d and c = %d newG = %lf and newH %lf and newF %lf
and old F %lf\n", current.r + i, current.c + j, newG, newH, newF, mappy[current.r +
i][current.c + j].hazardLevel);

                        if (mappy[current.r + i][current.c + j].f > newF)
                        {
                            mappy[current.r + i][current.c + j].g = newG;
                            mappy[current.r + i][current.c + j].h = newH;
                            mappy[current.r + i][current.c + j].f = newF;
                            mappy[current.r + i][current.c + j].prev_r = current.r;
                            mappy[current.r + i][current.c + j].prev_c = current.c;
                            q.push(mappy[current.r + i][current.c + j]);
                            //printf("adding to queue r = %d and c = %d\n", current.r + i, current.c
+ j);
                        }
                    }
                }
            }
        }
    }

    if (!pathFound)
    {
        printf("Failed to find the path to the Destination location\n");
        pathExists = false;
    }

    freeBoolArray(visited);
```

```
    return path;
}

int main()
{
    int j, k;
    bool obsDetect = false, obsDetectUltra = false;
    pathExists = true;

    commSetup();
    LidarSetup();

    //LOW turns on led | HIGH turns on led
    pinMode(7,OUTPUT); //RED led
    pinMode(0,OUTPUT); //GREEN led
    pinMode(2,OUTPUT); //BLUE led
    pinMode(11,INPUT); //button

    int prevButton = HIGH;
    for(;;)
    {
        if(prevButton == HIGH && digitalRead(11) == LOW)  // a falling edge
        {
            break;
        }
    }

    digitalWrite(0, HIGH); // turn green LED on
    digitalWrite(2, LOW); // turn blue LED on
    digitalWrite(7, LOW); // turn red LED on
    int LED = 0;  //label to show green is on

    rows = 20;  //hard coded testcase for summer without mesh network
    cols = 30;  //hard coded testcase for summer without mesh network

    mappy = (Locations**)malloc(sizeof(Locations*) * rows); //alocate memory for the
map
    for (j = 0; j < rows; j++)
        mappy[j] = (Locations*)malloc(sizeof(Locations) * cols);

    for (j = 0; j < rows; j++) //initialize map to all 0's
        for (k = 0; k < cols; k++)
            mappy[j][k].hazardLevel = 0;

    Locations InitialStart = getCurrentLoc(rows / 2, 0);
    Locations goal = Locations(rows / 2, cols - 1);
```

```
    Locations curLoc = InitialStart, nextLoc = InitialStart; //start from initial location, end at
last midpoint
    vector<Locations> ShortPath;
    int robotDirection = 0, obstacleDirection = 0, obstacleDist = 1000, ultraDist = 1000,
lastDist;

    //mappy = updatemap(17, 17);
    //mappy = updatemap(18, 17);
    //mappy = updatemap(17, 18);
    //mappy = updatemap(8, 7);

    bool stop = false, accel = false;

    k = 0;
    serialPutchar (fd, '4');//for 90 degrees
    serialPutchar (fd, '9'); // go forward
    int travelTimeDelayer = usleep(500000);
    //printf("starting AStar\n");
    ShortPath = AStarSearch(goal, curLoc);

    if(pathExists == false)
    {
        printf("path doesn't exist\n");
        serialPutchar (fd, '0');
        printMap();

        if(LED == 0)   // if green is on
        {
            digitalWrite(0, LOW);  // turn off green
            digitalWrite(7, HIGH);   // turn on red
            LED = 7;               //label to show red is on
        }

        if(LED == 2)  // if blue is on
        {
            digitalWrite(2, LOW);  // turn off blue
            digitalWrite(7, HIGH);   // turn on red
            LED = 7;               //label to show red is on
        }

        return 0;
    }
        //printf("ending AStar\n");
    int SPSize = ShortPath.size();

    //serialPutchar (fd, '9');
    //accel = true;
```

```
j = 1;
while (j < SPSize)  //itterate through each location in ShortPath
{
   nextLoc = ShortPath[j]; //select next location in path

   //to get speed and angle for drive controls
   int rDiff = curLoc.r - nextLoc.r;
   int cDiff = nextLoc.c - curLoc.c;
   int DriveAngle = getDriveAngle(rDiff, cDiff, robotDirection);
   float travelDist = (float)sqrt((rDiff * rDiff) + (cDiff * cDiff));
   //printf("starting top of loop drive angle == %d\n", DriveAngle);

   //turning
   //char t = getAngleChar(DriveAngle);
   /*if((DriveAngle >= 124) && (135 >= DriveAngle))
   {
      printf("hard right\n");
      serialPutchar (fd, '2');//for 135 degrees
   }

   if((DriveAngle >= 101) && (124 > DriveAngle))
   {
      printf("soft right\n");
      serialPutchar (fd, '3');//for 112 degrees
   }

   if((DriveAngle >= 79) && (101 > DriveAngle))
   {
      serialPutchar (fd, '4');//for 90 degrees
   }

   if((DriveAngle > 56) && (79 > DriveAngle))
   {
      printf("soft left\n");
      serialPutchar (fd, '5');//for 68 degrees
   }

   if((DriveAngle >= 45) && (56 >= DriveAngle))
   {
      printf("hard left\n");
      serialPutchar (fd, '6');//for 45 degrees
   }

   if (accel == true)
   {
      if(stop == false) //go
```

```
        {
            //Ramp up Speed
            serialPutchar (fd, '9');
            //serialPutchar (fd, '9'); // second one to deal with inconsistent motor
            printf("moving forward\n");
            //accChange = true;
        }
        else if (stop == true) //stop
        {
            //Ramp down Speed
            serialPutchar (fd, '0');
            //accChange = true;
        }
    }*/

    //if(accChange == true)
    accel = false;
    //int robotDirection = 180;  //find direction
    //compass goes here
    robotDirection = getPsuedoAngle(curLoc, nextLoc, robotDirection);  //calculate (or
get) current direction

    curLoc = nextLoc;
    if(mappy[curLoc.r][curLoc.c].hazardLevel < 1000000)
        mappy[curLoc.r][curLoc.c].hazardLevel += 3000000;  //mark as visited

    //printf("moving to (%d, %d) in direction %d ", curLoc.r, curLoc.c, robotDirection);
    printf("(%d, %d) at %d at speed %f turning %d \n", curLoc.r, curLoc.c,
robotDirection, travelDist, DriveAngle);
    //printf(" at (%d, %d) -> ", curLoc.r, curLoc.c);

    int p, q, numTest;
    char highBits, lowBits;
    fflush(stdout);
    for(p = 0; p < 25; p++)   // 2 lidar/ultra scans
    {
        bool ultraDet = false;
        fflush(stdout);
        for(q = 0; q < 3; q++)  // 1 for angle, distance, and ultra
        {
            if(serialDataAvail (fl))
            {
                highBits = serialGetchar (fl);
                lowBits = serialGetchar (fl);

                numTest = lowBits | highBits << 8;
```

```
                //setvalue depending on position in array
                obstacleDetect.array[(obstacleDetect.index) % 3] = numTest;
                //printf("lidar obstacleDirection numTest = %d\n", numTest);
                //increment the index
                (obstacleDetect.index)++;

                //printf("%d\n", numTest);
                //printf("       `%c", highBits);
                //fflush(stdout);
            }
            //printf("raw data obstacleDetect[%d] = %d\n", q, obstacleDetect.array[q]);
        }
        //obstacleDirection = obstacleDetect.array[0] - 90;
        //obstacleDirection *= -1;
        //obstacleDist = obstacleDetect.array[1];
        int l;
        for(l = 0; l < 3; l++)
        {
            if((obstacleDetect.array[l] != 90) && (obstacleDetect.array[l] != 25000))
                ultraDist = obstacleDetect.array[l];
        }
        //ultraDist = numTest;

        //if(obstacleDirection < 0)
        //obstacleDirection += 360;

        if(obstacleDist > 10)
            lastDist = obstacleDist;

        if(obstacleDist < 10)
            obstacleDist += lastDist;

        //printf("obstacleDirection = %d, obstacleDist = %d, ultraDist = %d\n",
obstacleDirection, obstacleDist, ultraDist);

        obstacleDist /= cmToFeet;
        if(ultraDist > 2) // remove false zero values
        {
            ultraDist /= cmToFeet;
            printf("ultraDist = %d\n", ultraDist);
            if(ultraDist < 4)
            {
                obsDetectUltra = findObstacleLoc(curLoc.r, curLoc.c, robotDirection,
ultraDist, 0); //for ultrasonic
                ultraDet = true;
            }
        }
```

```
        //obsDetect = findObstacleLoc(curLoc.r, curLoc.c, robotDirection, obstacleDist,
obstacleDirection); //for lidar
        if (ultraDet == true) //stop moving along path
        {
            printf("found Ultra obstacle\n");
            serialPutchar (fd, '0');

            if(LED == 0)   // if green is on
            {
                digitalWrite(0, LOW);  // turn off green
                digitalWrite(7, HIGH);   // turn on red
                LED = 7;               //label to show red is on
            }

            if(LED == 2)  // if blue is on
            {
                digitalWrite(2, LOW);  // turn off blue
                digitalWrite(7, HIGH);   // turn on red
                LED = 7;               //label to show red is on
            }

            accel = true;
            printMap();
            freeLocationsArray(mappy);  //frees memory used for global map

            return 0;

            break; // just incase
        }

    }
    travelTimeDelayer = usleep(500000); // 0.1 second per lidar/ultra scan
    //travelTimeDelayer = usleep(250000);
    //nanosleep(1000000000);

    if (obsDetect || obsDetectUltra) //stop moving along path
    {
        printf("found obstacle\n");
        serialPutchar (fd, '0');
        accel = true;
        printMap();
        freeLocationsArray(mappy);  //frees memory used for global map

        return 0;

        break; // just incase
    }
```

```
        j++;
    }

    //Ramp down Speed
    serialPutchar (fd, '4');//for 90 degrees
    serialPutchar (fd, '0');

    if(LED == 0)   // if green is on
    {
        digitalWrite(0, LOW);  // turn off green
        digitalWrite(7, HIGH);   // turn on red
        LED = 7;               //label to show red is on
    }

    if(LED == 2)  // if blue is on
    {
        int waterDelayerEnd = usleep(5000000);
        digitalWrite(2, LOW);  // turn off blue
        digitalWrite(7, HIGH);   // turn on red
        LED = 7;               //label to show red is on
    }

    //printf("after mods\n");
    printMap();
    freeLocationsArray(mappy);  //frees memory used for global map

    return 0;
}
#endif
```

Program 2: StopPath2.cpp