Row Crop Navigation by Autonomous Ground Vehicle for Crop Scouting

by

Austin Schmitz

B.S., Kansas State University, 2017

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Biological and Agricultural Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Approved by:

Major Professor
Dr. Daniel K. Flippo

Copyright

Abstract

Robotic vehicles have the potential to play a key role in the future of agriculture. For this to happen designs that are cost effective, robust, and easy to use will be necessary. Robotic vehicles that can pest scout, monitor crop health, and potentially plant and harvest crops will provide new ways to increase production within agriculture. At this time, the use of robotic vehicles to plant and harvest crops poses many challenges including complexity and power consumption. The incorporation of small robotic vehicles for monitoring and scouting fields has the potential to allow for easier integration of robotic systems into current farming practices as the technology continues to develop. Benefits of using unmanned ground vehicles (UGVs) for crop scouting include higher resolution and real time mapping, measuring, and monitoring of pest location density, crop nutrient levels, and soil moisture levels. The focus of this research is the ability of a UGV to scout pest populations and pest patterns to complement existing scouting technology used on UAVs to capture information about nutrient and water levels. There are many challenges to integrating UGVs in conventionally planted fields of row crops including intra-row and inter-row maneuvering. For intra-row maneuvering; i.e. between two rows of corn, cost effective sensors will be needed to keep the UGV between straight rows, to follow contoured rows, and avoid local objects. Inter-row maneuvering involves navigating from long straight rows to the headlands by moving through the space between two plants in a row. Oftentimes headland rows are perpendicular to the row that the UGV is within and if the crop is corn, the spacing between plants can be as narrow as 5”. A vehicle design that minimizes or eliminates crop damage when inter-row maneuvering occurs will be very beneficial and allow for earlier integration of robotic crop scouting into conventional farming practices. Using three fixed HC-SR04 ultrasonic sensors with LabVIEW programming proved to be a cost effective, simple, solution for intra-row maneuvering of an unmanned ground vehicle through a simulated corn row. Inter-row maneuvering was accomplished by designing a transformable tracked vehicle with the two configurations of the tracks being parallel and linear. The robotic vehicle operates with tracks parallel to each other and skid steering being the method of control for traveling between rows of corn. When the robotic vehicle needs to move through narrow spaces or from one row to the next, two motors rotate the frame of the tracks to a linear configuration where one track follows the other track. In the linear configuration the vehicle has a width of 5 inches which allows it to move between corn plants in high population fields for minimally invasive maneuvers.

Fleets of robotic vehicles will be required to perform scouting operations on large fields. Some robotic vehicle operations will require coordination between machines to complete the tasks assigned. Simulation of the path planning for coordination of multiple machines was studied within the context of a non-stationary traveling salesman problem to determine optimal path plans.

Keywords: Inter-row maneuvering, robotic crop scouting, row crop navigation, simulated path planning, unmanned ground vehicle

Table of Contents

List of Figures

List of Tables

Acknowledgements

Research made possible by the sponsorship of the Kansas Corn Commission.

Dedication

Dedicated to my parents, Joe and Amy Schmitz.

to Dad – Thank you for inspiring me with your love for agriculture, especially our cows.

to Mom – Thank you for inspiring me to become an engineer and musician.

Chapter 1 - Introduction

There is a need to collect more information about the growing conditions of agricultural crops throughout the growing season in order to increase the output of cropping systems within production agriculture. Providing farmers and researchers with more information allows them to develop crop management strategies and make informed decisions that lead to increased production. A better understanding of soil moisture, nutrient levels, and pest concentrations will allow farmers and researches to make management decisions for their cropping system that will lead to healthier and more efficient crops thus increasing production. As farms become larger and larger it can also become harder for famers to scout all of their fields effectively, thus many technologies have emerged to help farmers collect the needed information to make management decisions.

Over the last several years the use of UAVs has become more prevalent in crop scouting, particularly for monitoring soil moisture levels and nutrient levels thru NDVI (Normalized Difference Vegetation Index) images to complement satellite imagery. Advancements within precision agriculture such as higher resolution cameras, big data, and data processing have opened the door for improved crop scouting for up to date information in real-time. Most research has focused on improving information regarding soil moisture and nutrient levels of cropland. Little research has focused on understanding another challenge that farmers and researchers must face; management decisions concerning harmful insects in crops.

Information regarding insects that are found within corn fields are of particular interest, especially with regards to identification, population levels, concentration per area, migration patterns, and behaviors. To better understand these variables, information collection needs higher precision than current practices allow. Current practices for pest scouting involves collecting pest samples by hand to determine population density or observing plants for damage to the roots, stems, or leaves. Neither farmers nor researchers have the time or resources to walk entire fields, thus decisions are made based upon information found within the scouted area. If a pest control management practice is chosen for a field it is possible that pesticides are sprayed in areas where there are very few or no pests at all, therefore expending unnecessary resources.

Unmanned aerial vehicle (UAV) technology has been developed to allow researchers and farmers to monitor crop water stress, nutrient levels, and weed densities, in a fast and efficient manner by flying over the crop to complement satellite imagery (Conesa-Munoz, Valente, del Cerro, Barrientos, & Riberio, 2016). However, sub-canopy scouting in crops such as corn, sorghum, and beans is much more challenging as the UAVs cannot "see" under the canopy after its closure. It is important to be able to scout the sub-canopy as this is where many harmful insects reside. In corn, the canopy is typically closed by V10 (ten leaves formed) leaving the producer unable to collect information about pests found on the underside of leaves.

Unmanned ground vehicles (UGVs) provide a way to move below the canopy and collect information at the desired precision thus complimenting and providing value to a scouting system already using UAVs or satellite imagery for real-time information about their crops. Fleets of small UGVs can be deployed to scout sub-canopy and monitor large fields in real-time, collecting and sharing the information back to the farm database allowing famers to make better management decisions for their crops. These management decisions will ensure that the crops are allowed to grow with minimal insect stress for maximum growth while applying the needed control methods in precise locations.

An ongoing research project within the Biological and Agricultural Engineering Department at Kansas State Unversity, funded by the Kansas Corn Comission and titled

CareTaker, is focusing on developing an autonomous system for sustainable precision pest mitigation, Figure 1.

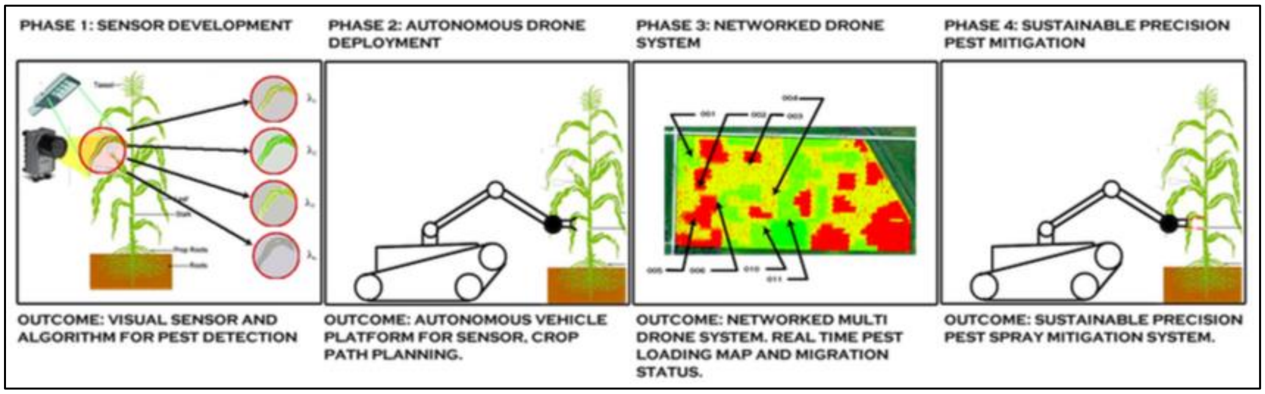

PHASE 1: SENSOR DEVELOPMENT
PHASE 2: AUTONOMOUS DRONE DEPLOYMENT
PHASE 3: NETWORKED DRONE SYSTEM
PHASE 4: SUSTAINABLE PRECISION PEST MITIGATION
OUTCOME: VISUAL SENSOR AND ALGORITHM FOR PEST DETECTION
OUTCOME: AUTONOMOUS VEHICLE PLATFORM FOR SENSOR, CROP PATH PLANNING.
OUTCOME: NETWORKED MULTI DRONE SYSTEM. REAL TIME PEST LOADING MAP AND MIGRATION STATUS.
OUTCOME: SUSTAINABLE PRECISION PEST SPRAY MITIGATION SYSTEM.

Figure 1 - CareTaker Project Timeline

Before control and mitigation of pests can occur, these pests have to be identified. The first phase of the project has been to identify pests by a visual sensor with the ability to detect differences in reflected light wavelengths. The difference in wavelengths between plants and pests are to be filtered through a image analysis algorithm, which gives a confidence level of mapped pests in the field. The effort to distinguish pests from plants and other foreign obstacles will be challenging, with a reward of providing specific information allowing farmers to control pests before they can cause an economic loss. The second phase of the project is to develop an autonomous UGV to carry the sensor through corn fields. The challenge for the autonomous vehicle will be to navigate successfully through its enviroment and canvass a field. A vehicle that is minimally invasive and can scout large acreages daily will provide farmers with information about the dynamics of the pest populations. Phase three of the research project involves the coordination and path planning of multiple UGV's as they work together to map the pest conditions within a field. Researching how to coordinate multiple UGV's within an agricultural field will have applications for all agricultural robotic systems beyond pest scouting.

Phase four includes researching unconventional control techniques such as micro spray and laser eradication methods. The uniqueness of these control techniques will be take time to research, but have potential to be environmentaly friendly than complete coverage of fields with chemicals.

From July 2015-July 2016 sensor development was completed and a preliminary design proposed. The sensor now needs to be carried upon a robotic vehicle through corn fields for field testing. Research started in January 2016 to study the challenges that an unmanned ground vehicle will have to address within a cornfield to carry the pest identification sensor for the development of an autonomous UGV in Phase Two.

This research is focused on studying unmanned ground vehicle solutions that can carry sensors for collecting information within corn fields. For a rover traveling between corn rows it can be expected that the corn rows will be spaced 30 inches apart and that a field will have headlands. Headlands shown in Figure 2, also known as turn rows, are the outer perimeter of a field where large equipment, tractor and nitrogen applicator in Figure 3, can turn around between passes through the field.

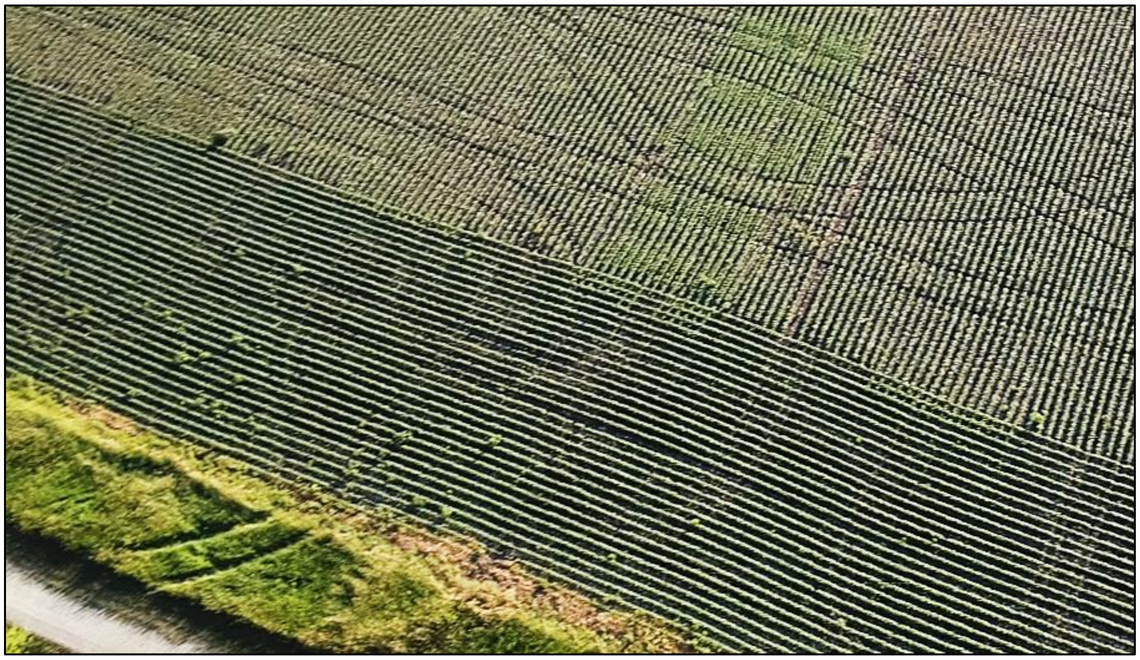


Figure 2 - Headlands of Bean Field (going diagonal from top left to bottom right)

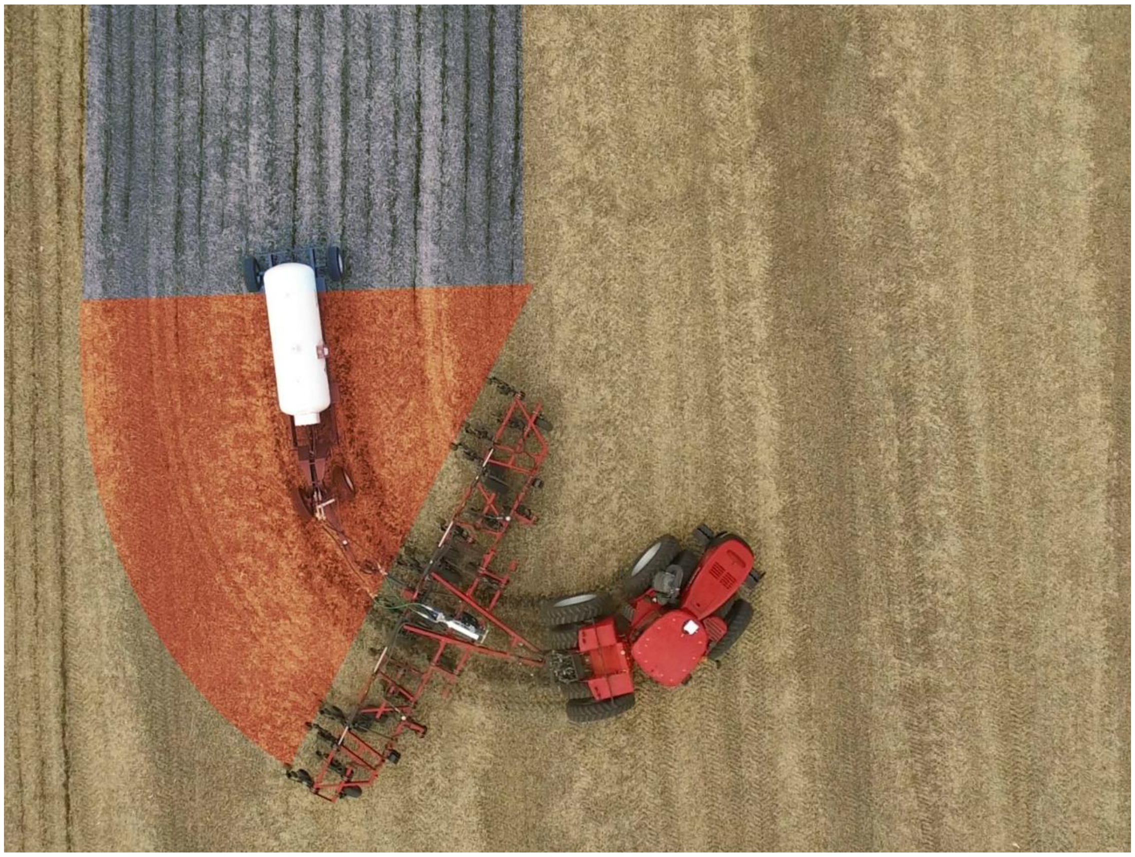


Figure 3 – Red Area is Headland Where Case IH Tractor Turns (caseih.com)

The spacing between corn plants is determined by planting population which can vary from 12,000-36,000 plants per acre (Kansas State University, 2007). This results in individual plant spacing from 17.4 inches to 5.8 inches respectively. However, the spacing between corn plants often varies due to the planter configuration. This can lead to a decrease in the accuracy of spacing and will have an effect upon navigating a rover through the headland rows as the plants are not always spaced exactly 6 inches apart. Knowledge about the basic arrangement of a corn field means that robot sensors required to navigate the field do not have to be as sophisticated or fully aware of every single point in space to be able to operate. An autonomous vehicle must be able to scout a corn field between the growth stages of V3 (emergence of third leaf) to R1 (silking) which is during the growth stages of various insects that can cause damage to corn (Kansas State University, 2007).

Vehicle Design Challenges

Scouting large acreages will require multiple robots, assembled together as a fleet and passing information between each other. The design of the robots will need to be cost effective so that many can be produced and remain an economical solution for producers or researchers. The cost of the robots will be offset by their ability to adapt to other scouting or monitoring tasks. The robots must also be able to detect unforeseen obstacles such as pivot ruts, washed out gullies, large exposed rock formations, or trees that could have fallen into fields from the edges. Therefore, the sensors and obstacle detections systems will need to be simple, robust, and inexpensive. In addition to avoiding obstacles the robots must be minimally invasive as they travel around the field and thru headlands, which will allow the robots to conserve energy and avoid damaging the crop.

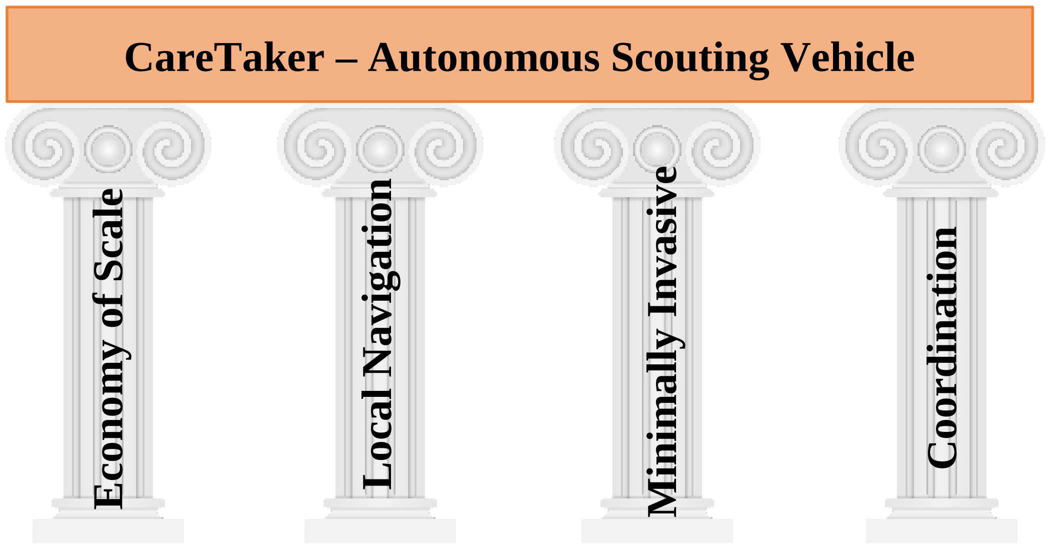


Figure 4 - Primary Challenges for CareTaker Robotic Vehicle

Cost Effectiveness - Economy of Scale

The average size of a farm in Kansas, according to the United States Department of Agriculture 2012 Census, is 750 acres. The average harvested cropland acres for the largest 20% of farms in Kansas is approximately 3000 acres (USDA NASS, 2012). For robots to be integrated into large farming operations they will need to be able to economically cover a lot of ground so that fleets of robots can scout the required acres. One factor determining the number of robots for a field, is the resolution of the scouting desired by the farmer or researcher. The second factor will be the speed at which the robots travel, thus determining the acreage that the robots can cover in a day. Related to this second factor will be the power capacity and the rate of power usage.

Because many robots will be needed, the sensors, drivetrain, and computational hardware will need to be economical, simple, and robust to keep the overall cost of the robots low and the duty cycle high. The cost of the robots cannot outweigh the value provided to the farming operation during pest scouting operations. By developing a base vehicle that can also be used for additional scouting and monitoring tasks, the overall value of the robot system to the farmer will increase. If the robots have additional payload capabilities to carry control methods for insect or plant pests, their value will also increase. Future development of robotic vehicles will include these capabilities thus enhancing the integration of unmanned ground vehicles into current farming practices.

Obstacle Avoidance – Local Navigation

Using an infra-red (IR) sensor array, researchers at Kentucky were able to develop and test a system that used six IR sensors in a manner that allowed them to effectively detect obstacles at distances up to 5 m (Pitla, Luck, & Shearer, 2010).

In addition to avoiding random obstacles, autonomous pest scouting robots will have to follow the crop rows to maneuver through the field, without damaging the valuable crop. Researching existing technologies for intra-row maneuvering led to machine based vision and LIDAR (Light Detection and Ranging). With advancements in machine based vision, agricultural robots have been developed to follow rows of corn with image processing algorithms imposed upon image taken by cameras as the robot travels through the field (Zhang, 2010) and (Lui, Mei, Niu, Lui, & Chu, 2016). LIDAR technology has experienced a resurgence in the last several year as the price of the sensors and processing technology has decreased. After collecting a point cloud of all the obstacles within view, complex algorithms are required to extract the curvature and spacing of the corn plants to then give direction to the robotic vehicle (Santosh A. Hiremath, 2013). Both systems, LIDAR and image processing, are costly and require large amounts of computational power to navigate a robotic vehicle between corn rows.

Ultrasonic sensors have the potential to be a less costly alternative. These distance sensors work by sending out a high frequency sound wave and then measuring the time it takes the sound to come back. Individual HC-SR04 ultrasonic sensors have costs that range from $2.50 to $35. Research conducted at the University of Kentucky showed that ultrasonic sensors can be effective a measuring the distance to obstacles common in an agricultural field (Dvorak, Stone, & Self, 2016). When presented with a variety of obstacles the ultrasonic sensors were able to detect and accurately report the distance from the sensor to the obstacle.

As mentioned in the previous section, many UGVs will be tasked with operating in large fields operating as a fleet to allow them to collect information about the whole field in a shorter amount of time. To reduce the overall cost of a fleet of UGVs, economical and robust sensors are desired to navigate the robots in an environment that is semi-known. The larger a robotic vehicle or autonomous system becomes, more sensor redundancy is needed to ensure that the robots do not cause any harm to surroundings or people (Kohanbash, Bergerman, Lewis, & Moorehead, 2012) and (Conesa-Munoz, Valente, del Cerro, Barrientos, & Riberio, 2016).

The concept of simplicity to overcome difficult autonomous vehicle problems is not new. A simple robot named Scarecrow competed in the Association for the Advancement of Artificial Intelligence Robot Competition in 1992 against robots with far more complicated systems and much more expensive sensors (Miller, Milstein, & Stein, 2007). The computational power required for the complicated robots was much higher than those found upon the Scarecrow, which had no camera and no sensors for detecting obstacles at different ranges. Scarecrow performed very well in the competition placing behind Flakey and CARMEL, but ahead of eight other robots that had funding in excess of $10,000. The approximate cost for the top three performing robots were; Scarecrow (~$200), Flakey (~$500,000), and CARMEL (~$1,000,000). Designing for the specific task at hand allowed the design of Scarecrow to be simple and robust.

This can be applied to other vehicle designs by identifying the information that is required to be collected by the vehicle in the environment in which it is placed.

Headlands – Minimally Invasive

Different researchers around the world have developed UGVs to perform scouting operations within row crop fields. Many of these vehicle configurations constrain them to operation in fields that are planted without headlands. These vehicle configurations work well for research test plots and for fields that are planted straight to the edges. However, if tasked with scouting a field with headlands, these configurations would be required to run plants over while performing a turning maneuver before the next scouting pass through a field.

The Rowbot, Figure 5, is a large robotic vehicle being used in corn fields to monitor and apply nitrogen in-season in addition to being able to inter-seed cover crops during the growing season (Cavendar-Bares, Bares, & Bares, 2014). The Rowbot has a large turning radius and cannot operate in conventionally planted corn fields that have headlands without incurring some crop damage.

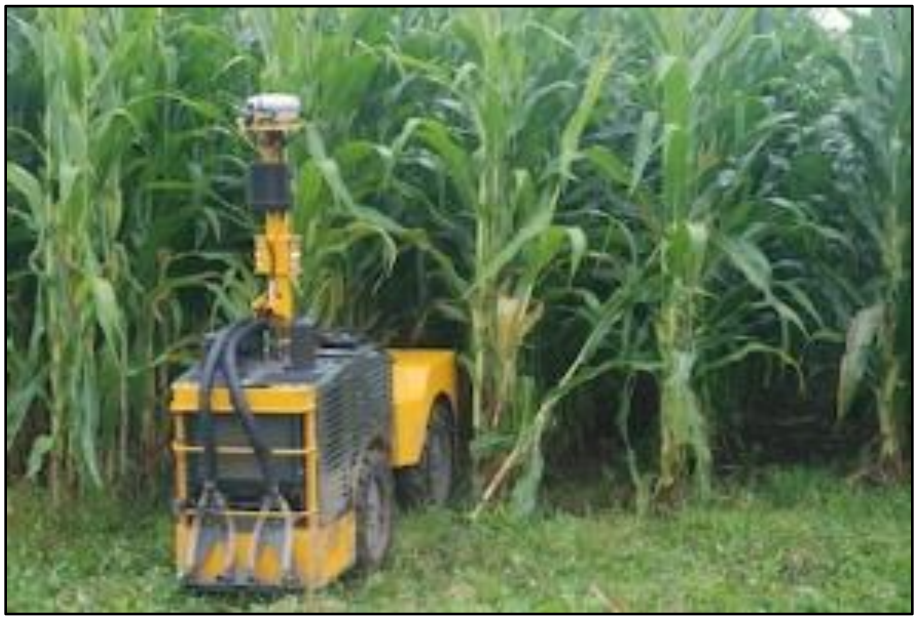


Figure 5 - ROWBOT Entering a Corn Field (www.rowbot.com)

Another design is BoniRob, Figure 6, a four legged multi-row vehicle that straddles two rows of crops collecting data information as it drives down the rows (Bawden, Gall, Kulk, Perez, & Russell). The BoniRob can exchange different sensor packages for different management tasks. Like the Rowbot, this vehicle is limited in its abilities to integrate into field with headlands as it is required to run over crop when presented with a path that requires going through the headlands.

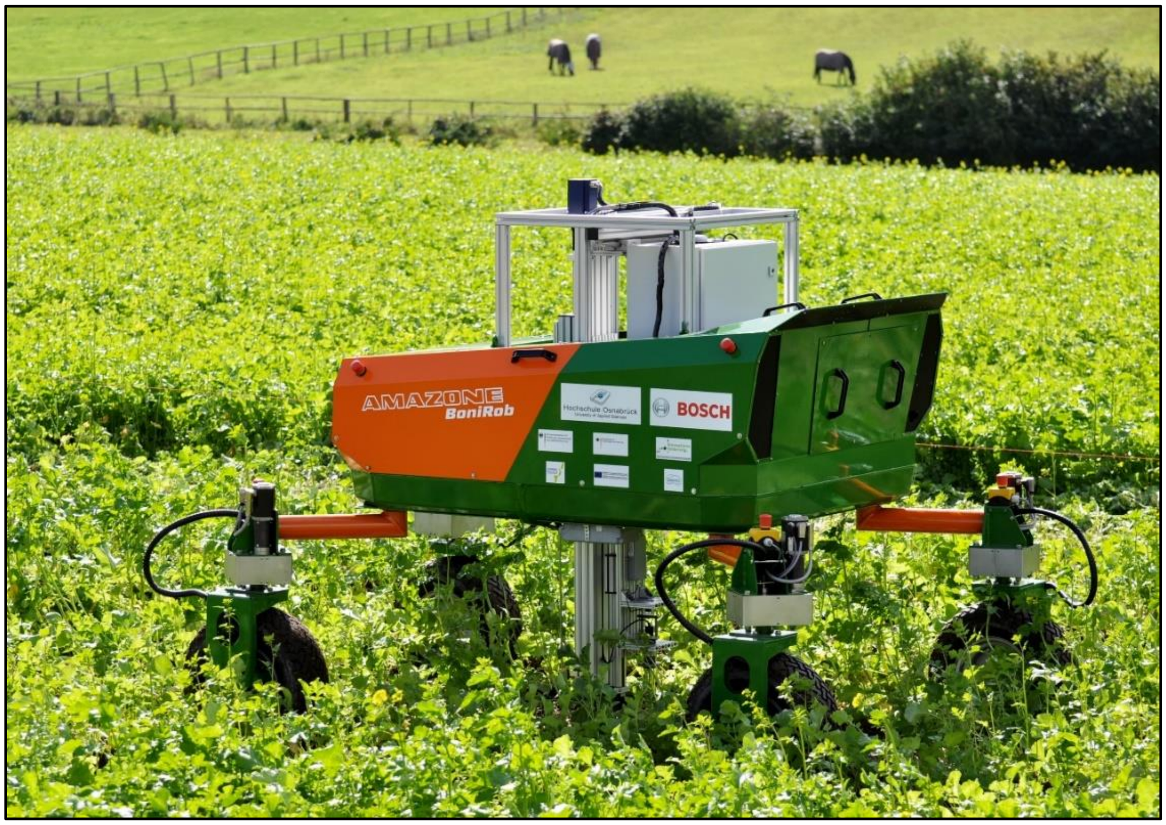


Figure 6 - BoniRob Collecting Data in Field Operation (www.deepfield-robotics.com)

Researching vehicle designs outside the realm of agriculture led to shape changing robots similarly tasked with challenging objectives, particularly search and rescue robots and surveillance robots. These robots feature the ability to maneuver through confined spaces while carrying sensors to survey and map their environment. The Guardian S, show in Figure 7, can climb metal walls, stairs, and enter tunnels while being controlled via tether or wireless remotes.

The Guardian S's duration of operation is 18 hours in inspection mode and 4 hours of continuous driving. The base cost of Guardian S is $60,000 with additional costs for customizations (Strickland, 2017).

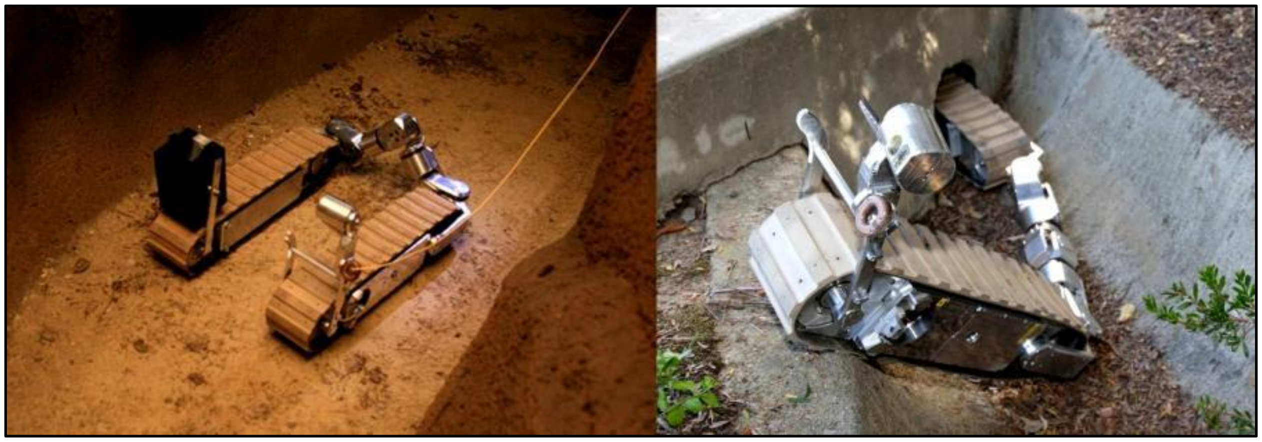


Figure 7 – Guardian S Robot (Left–tank configuration, Right–snake configuration)

A continued search into robotic vehicle designs led to tunnel exploration robots that can change shape to overcome obstacles and enter narrow passageways (Larson, Okorn, Pastore, Hooper, & Edwards, 2014). A transformable tracked vehicle used to study torque requirements for shape-shifting found that moving while shape-shifting reduced the torque load on the motor turning the mobile unit (Li, Ma, Li, Wang, & Wang, 2010).

While each of the different designs reviewed had many positive aspects that are required for the scouting job, none were able to meet the challenge of being a cost effective solution that was minimally invasive to the scouted crop. The Rowbot and BoniRob require a different planting strategy or running over crops to complete their scouting in conventionally planted field. While the Guardian S would be able to scout a field while being minimally invasive, the low continuous run time would require large fleets which would be cost-prohibitive for farmers. An economical solution is desired to integrate into existing farming practices to complete minimally invasive scouting.

Path Planning Simulation

Within conventional agricultural operations there is often coordination between equipment to complete various jobs. There are tasks that require cooperation between like machines (i.e. homogeneous coordination); while other operations require coordination between different machines (i.e. heterogeneous coordination). An example of homogeneous coordination is spraying a field with chemicals using multiple sprayers operating at the same time in different areas of the field as they share information about coverage. Coordination between a combine and tractor-grain cart for the transfer of grain is an example of heterogeneous coordination.

Heterogeneous coordination can be broken down further into two different categories; indirect coordination and direct coordination (Figure 8). Indirect coordination can occur between a UAV that scouts a field for weeds and then generates a herbicide treatment to be applied by unmanned ground vehicles as proposed by Jesus Conesa-Munoz in a multi-robot sense-act system (Conesa-Munoz, Valente, del Cerro, Barrientos, & Riberio, 2016). Direct coordination occurs when different machines interact in a way that requires physical transfer of material; transferring harvested grain, refueling a vehicle, swapping batteries, or refilling spray tanks.

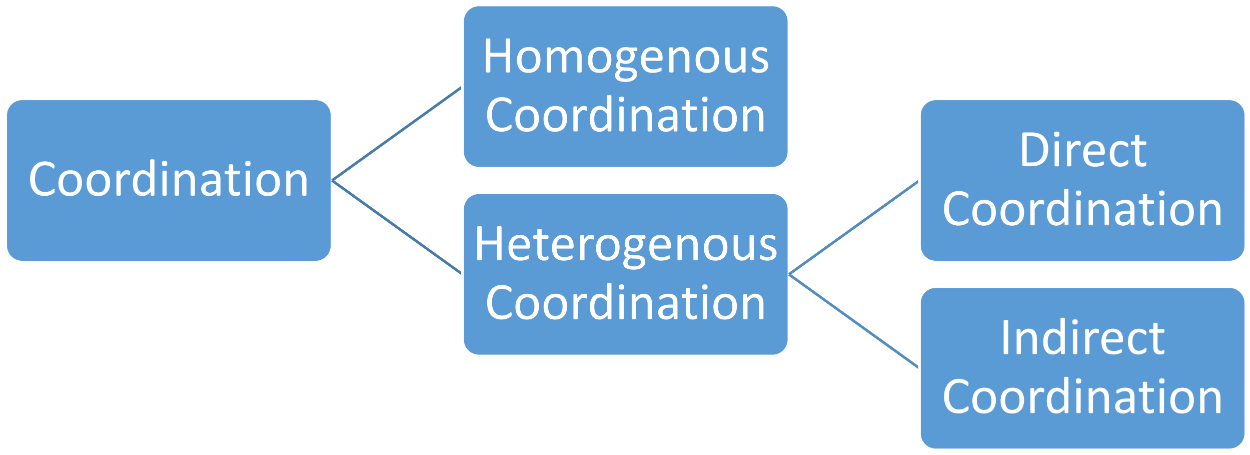


Figure 8 - Different Designations for Coordination within Agriculture

As information technology has developed, the sharing of location and status information provides the opportunity to improve operations in agriculture, specifically coordination between equipment. Improving coordination routines can reduce the time required to complete tasks as well as conserve resources such as fuel and fertilizer.

With advancements in GPS it is possible to plan the paths of machines through a field and share the information between machines to know what has been covered. For example, during harvest two combines might operate in the same field, maximizing their coverage area with minimal overlap between passes. Kinze Manufacturing Inc. is developing autonomous capabilities for tractors that pull grain carts so in field operation efficiency can be improved (Kinze Manufacturing Inc, 2014). As autonomous equipment for agriculture develops it will rely intensely on coordination between homogeneous- and heterogeneous- machines to complete their tasks. For small autonomous equipment where multiple machines will be required to operate in the same field, path planning and coordination will be very critical for application success, efficiency, and safety.

Simulation of homogeneous spraying by autonomous vehicles allowed researchers at the Centre for Automation and Robotics to study the optimization of routes for a fleet of autonomous vehicles. The simulation allowed them to consider different vehicle characteristics, optimization criteria, variability of the field, and refilling operations vs more robots. Simulated annealing was used to find the optimal solution for the various configurations (Conesa-Munoz, Bengochea-Guevara, Andujar, & Ribeiro, 2016).

Operating many homogeneous machines in the same field has been successful due to the improvements with GPS and the sharing of work zones between machines. The challenge with coordinating autonomous heterogeneous machines in the same field is that the coordination may require moving while the coordination operation occurs. This challenge daunting due to the fact that it is a dynamic coordination of current, past, and future events happening in real-time between machines.

The problem of heterogeneous-coordination can be characterized as a Non-stationary Traveling Salesman Problem (NTSP). The NTSP is a variation of the traveling salesman problem (TSP) which involves a salesman finding the shortest path between multiple cities (Potvin, 1996). The variation made to the TSP to become a NTSP is that there is a dynamic element, either a changing number of cities or the cities themselves have a changing position. For an agricultural application moving cities become analogous to moving equipment within a field.

Research conducted by Jiang (Jiang, Sarker, & Abbass, 2005) regarding the NTSP focused on finding solutions for scenarios where the targets have a constant velocity vector. A genetic algorithm was utilized to find the best solution in a heuristic approach. Simulation of coordination between scouting UAVs and mobile charging station UGVs was conducted to understand how modifications could be made to solve the problem using the traveling salesman approach (Mathew, Smith, & Waslander, 2015).

The primary difference between the problems presented by both Jiang and Mathew, is that vehicles in agricultural fields change their velocity vector when turning around at the end of a field after completing every pass thru the field. Preliminary research regarding the application of the traveling salesman problem to an agricultural field with changing velocity vectors will be critical in further development of the CareTaker project for future phases as well as optimization and coordination scheduling for other agricultural tasks.

Chapter 2 - Robotic Vehicle Design

Design Goals and Objectives

After completing literature review a list of vehicle design goals was compiled. This list provided direction for decisions made regarding vehicle features. Design goals include:

- Duty Cycle – Coverage Area
- Lightweight – No soil compaction
- Maneuverability – Minimally Invasive
- Localized Navigation – Obstacle Avoidance
- Sensor Package Payload – Carry pest identification sensor

A small robotic vehicle capable of driving between two rows of a row crop and maneuvering through the headlands with minimal or zero damage would be beneficial to researchers and producers who have fields to be scouted that are planted with or without headlands.

Materials and Methods

Preliminary testing of the visual pest sensor was conducted in the summer of 2016 with an existing robotic vehicle, Prototype 1 shown in Figure 9. Prototype 1 had been built to demonstrate how a pest identification sensor could be carried through the field and operated under remote control. The pest identification sensor that had been in development and lab tests for the previous year was taken to the field after dusk to test its capabilities. Three LED lights were mounted on the machine to provide the light that would reflect off of the plants and the exoskeleton of the pests. During the testing of the pest identification sensor observations were also made and recorded regarding the challenges that the robotic vehicle encountered in the field. The frame of Prototype 1 had dimensions of 21"x11" and used four PLA printed tires resulting in outside dimensions of 21"x 21". The rover was controlled by a HiTEC OPTIC 5 remote and HiTEC Minima 6T receiver that sends signals to a NI myRIO-1900 (National Instruments, Austin TX) which controls a Sabertooth 2x12 motor controller (Dimension Engineering, Hudson

OH). The robot was powered by two RIGID 18V 2.0 Ah Lithium-Ion rechargeable batteries and utilized skid steering for simplicity and agility. The vehicle was remote controlled and had several cameras mounted on it to record the challenges experienced, Figure 10.

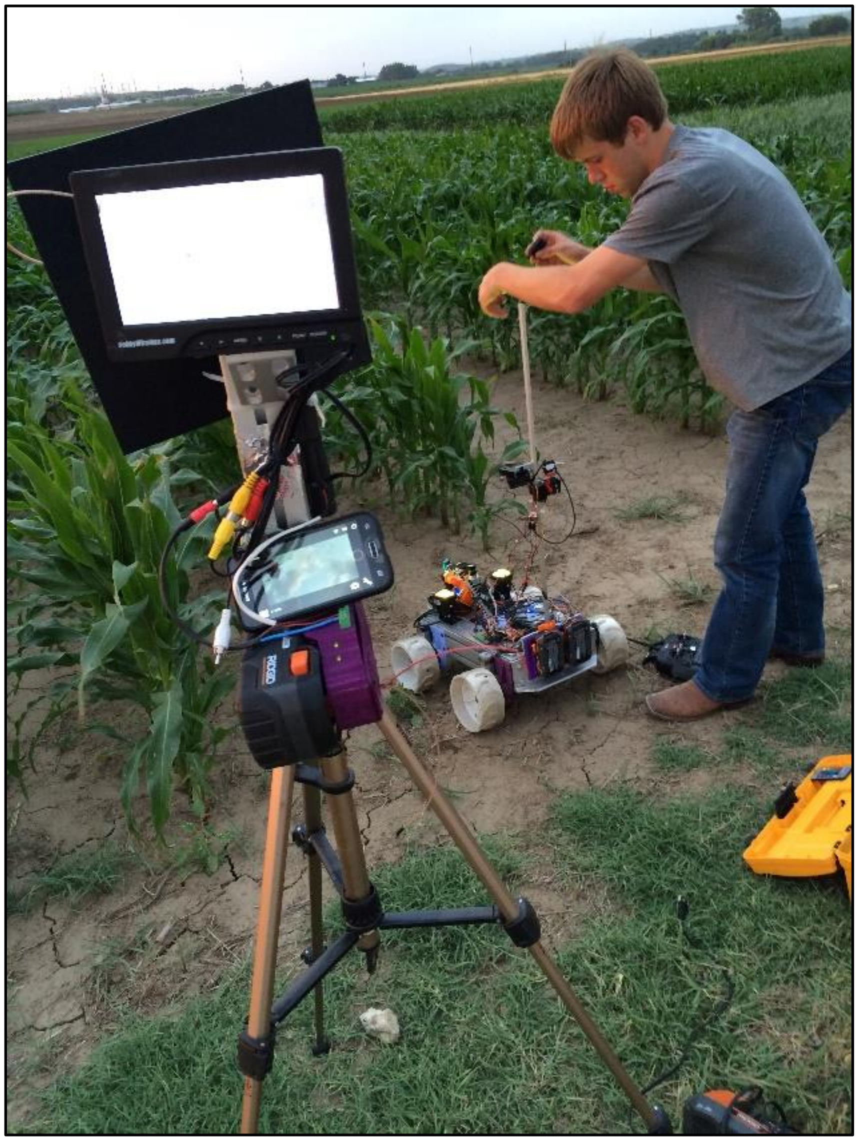


Figure 9 - Prototype I Being Outfitted with Cameras for Data Collection

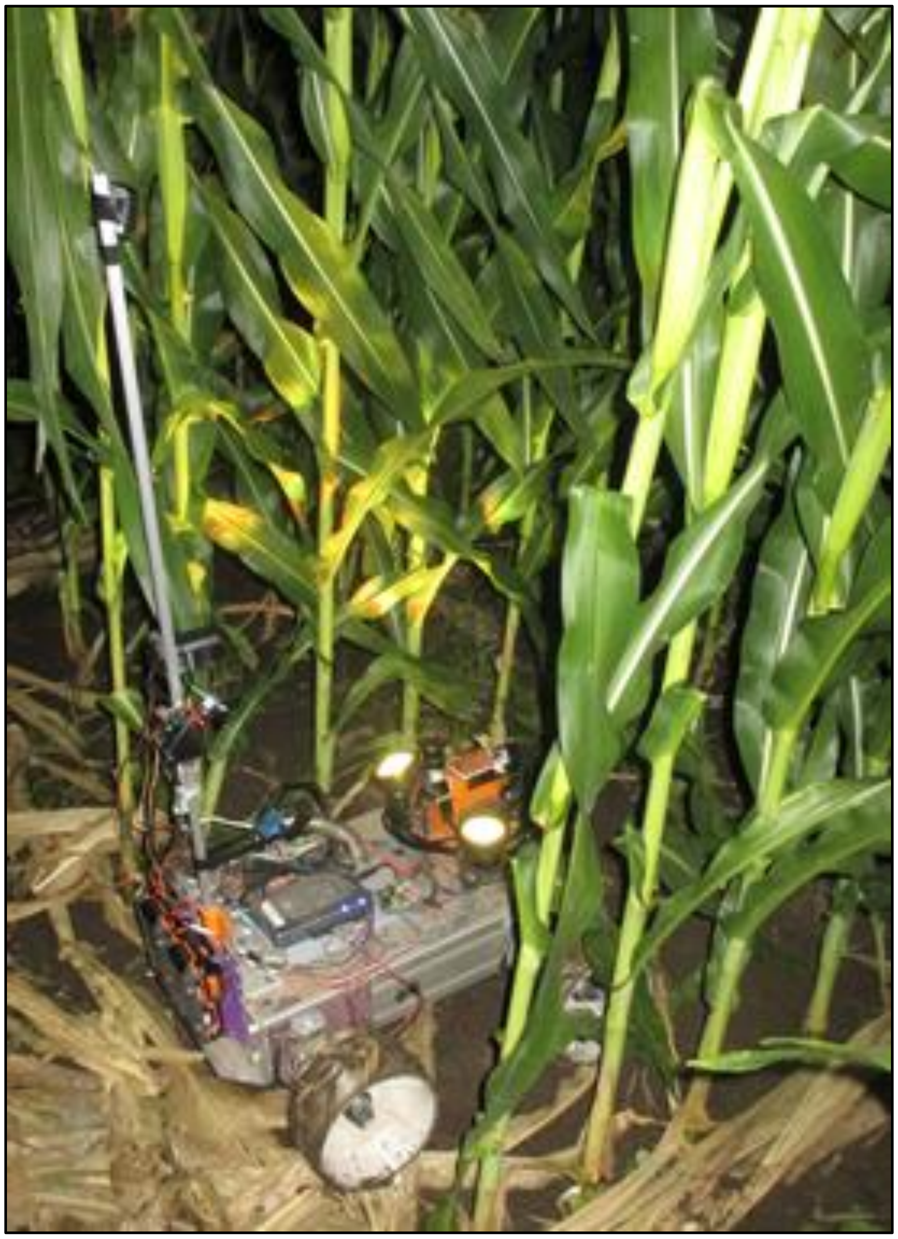


Figure 10 - Operation of Prototype 1 in Field Trials

Preliminary testing of Prototype 1 observations include:

PROS

- Skid steering proved to be effective for vehicle maneuvering
- Sub-canopy scouting was operational and in correct location for pest scouting
- 18V rechargeable batteries worked well for power supply and battery swapping
- 2.4 GHz signal was strong through the dense canopy for video transmission
- Torque from worm gear motors was adequate for application

CONS

- Overall width of vehicle minimized the window for steering corrections
- Wheels occasionally became stuck between stalks if driven off of heading
- Speeds of 3 mph were achieved, but were too fast for pest identification sensors
- Bluetooth transmission thru canopy was poor at long ranges from operating base
- Worm gears consumed a lot of power and quickly drained 18V rechargeable batteries

From the summer testing it was determined that it was common for an operator to drive the rover off of the correct heading and then run the rover into stalks, primarily due to the overall width of the vehicle. When the rover was driven off of the correct heading it had very little space or time for correction before it ran into the stalks.

Current challenges for UGVs reside in integration into current farming practices (headlands), minimally invasive (not running crop over), and energy consumption (small robot running over corn > going through gaps). Research was conducted regarding two maneuvering methods, intra-row navigation and inter-row navigation, to address these challenges facing scouting vehicles. Intra-row navigation occurs when a robotic vehicle travels between two rows of crop following straight or contoured rows depicted by the yellow arrows in Figure 11. Inter-row navigation happens when a robotic vehicle moves from one row to another. Inter-row navigation can take place in two ways; #2a, red arrows, exiting the row by moving into the headlands and then entering into the next row, or by #2b, orange arrows, moving directly into the next row by moving between plants.

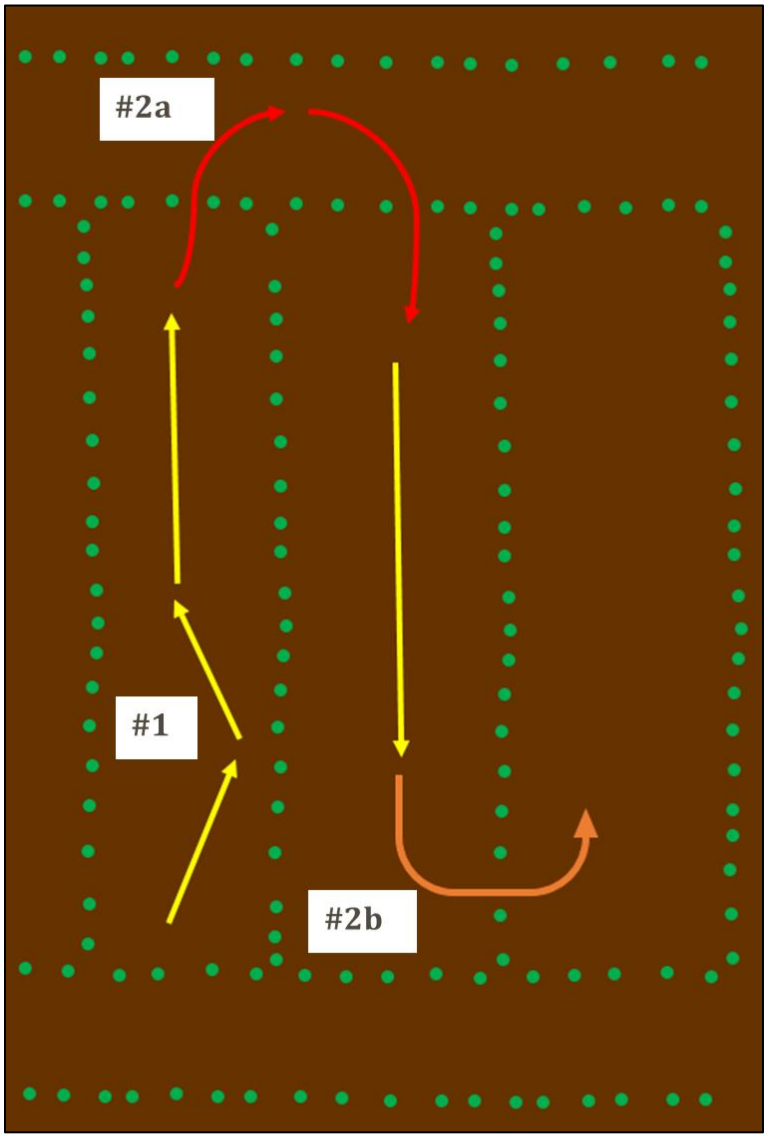


Figure 11 - Depiction of Maneuvering: #1 - Intra-Row; #2 - Inter-Row

Intra-Row Maneuvering

Researchers at the University of Kentucky showed that ultrasonic sensors can be effective in identifying obstacles in an agricultural setting (Dvorak, Stone, & Self, 2016). However, no research was found regarding the use of ultrasonic sensors for navigation specifically in a corn field. Due to the low cost of ultrasonic sensors and the lack of research for navigation an experimental trial was conducted. Three HC-SR04 ultrasonic sensor were purchased at a cost of

$2.50/sensor. The sensors have centimeter accuracy and a range of up to 10'. The sensors have a 30° spread pattern from the location of the sensor. With three ultrasonic sensors a fixed array was created that would allow the rover to detect and distinguish obstacles between left, center, and right. To create the array, the ultrasonic sensors were modeled in CREO Parametric, and a sketch made of the sensors degree of range given by the specification sheet in Appendix A. A mount for the three sensors was modeled with CREO Parametric and printed with PLA plastic, Figure 12.

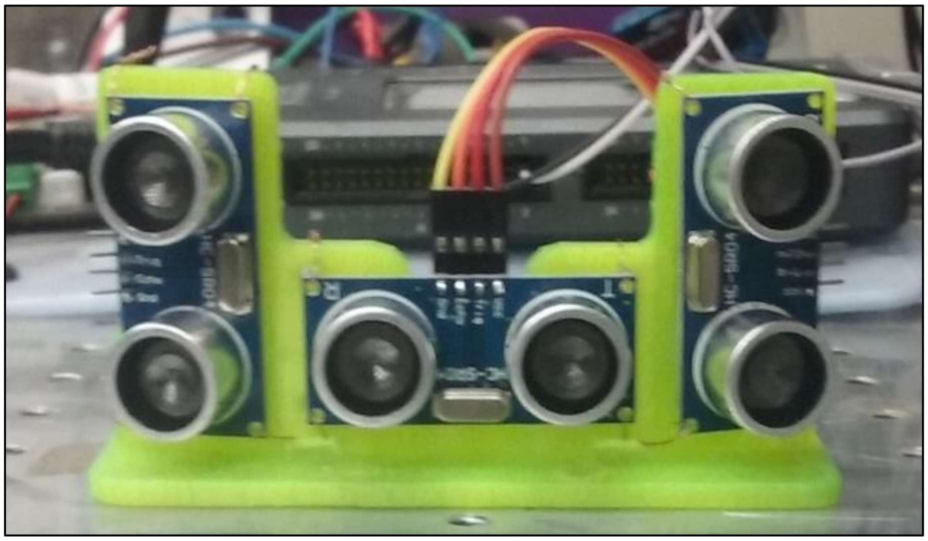


Figure 12 - Sensor Array Mounted on Prototype 1

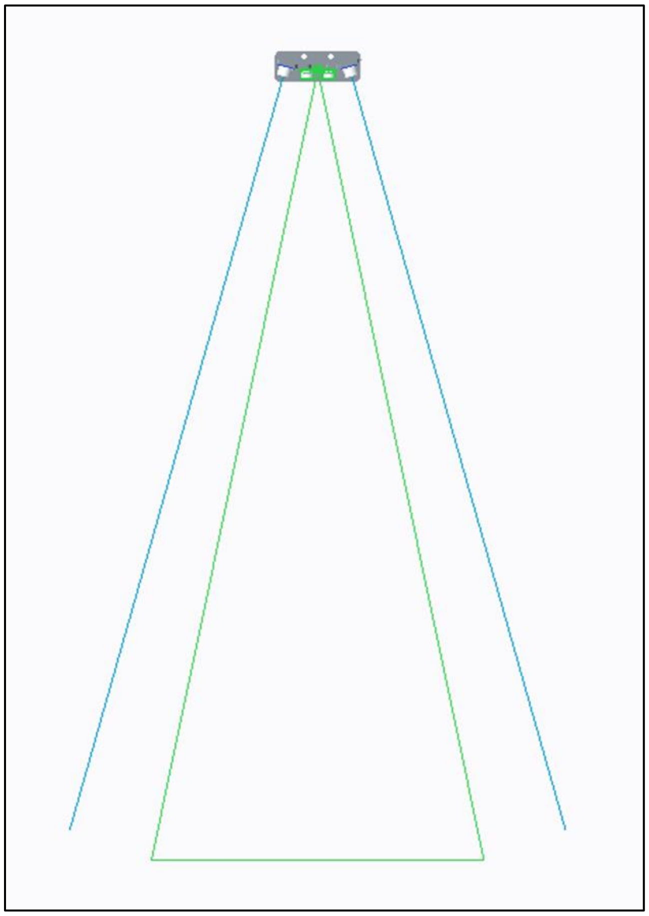


Figure 13 - Ultrasonic Sensor Array - 30° Spread Pattern of Middle Sensor (green)

As the spread pattern for the middle sensor is 30°, the support for the left and right sensors are set 15° outwards from the center plane. While facing the sensor as in Figure 12, the middle sensor is held in the horizontal position, the right sensor is rotated 90° clockwise, and the left sensor is rotated 90° counterclockwise. The ultrasonic sensors are controlled and read using the myRIO FPGA interface.   Information is passed from the myRIO FPGA to the myRIO Real Time application, where the control decisions are made. Indicator lights were programed to indicate when objects were within a specified range.   In autonomous operation control decisions are sent via PWM (Pulse Width Modulation) signals from the myRIO RT to the Sabertooth 2x12 motor controller, which provides power to the wheels in order to avoid obstacles (Appendix B -).

All three sensors are triggered at the same time from the myRIO to minimize overlap of the return signals. Upon initial testing of the sensor array, obstacle detection interference was observed between the sensors. It was believed that the sound waves from the middle ultrasonic sensor were echoing off of an obstacle and returning into the left or right ultrasonic sensors. This prevented the system from identifying the exact region of the obstacle. A better understanding of the range of each individual sensor was desired so the ranges to obstacles were mapped out. 1.5" poplar dowel rods were used to simulate cornstalks. The width and depth of each sensor was mapped out at ranges of 5", 7.5", 10", 15", 20", 25", 30", 35", and 40". The points were input to Excel to create a graph and calculate accuracy, Figure 14.

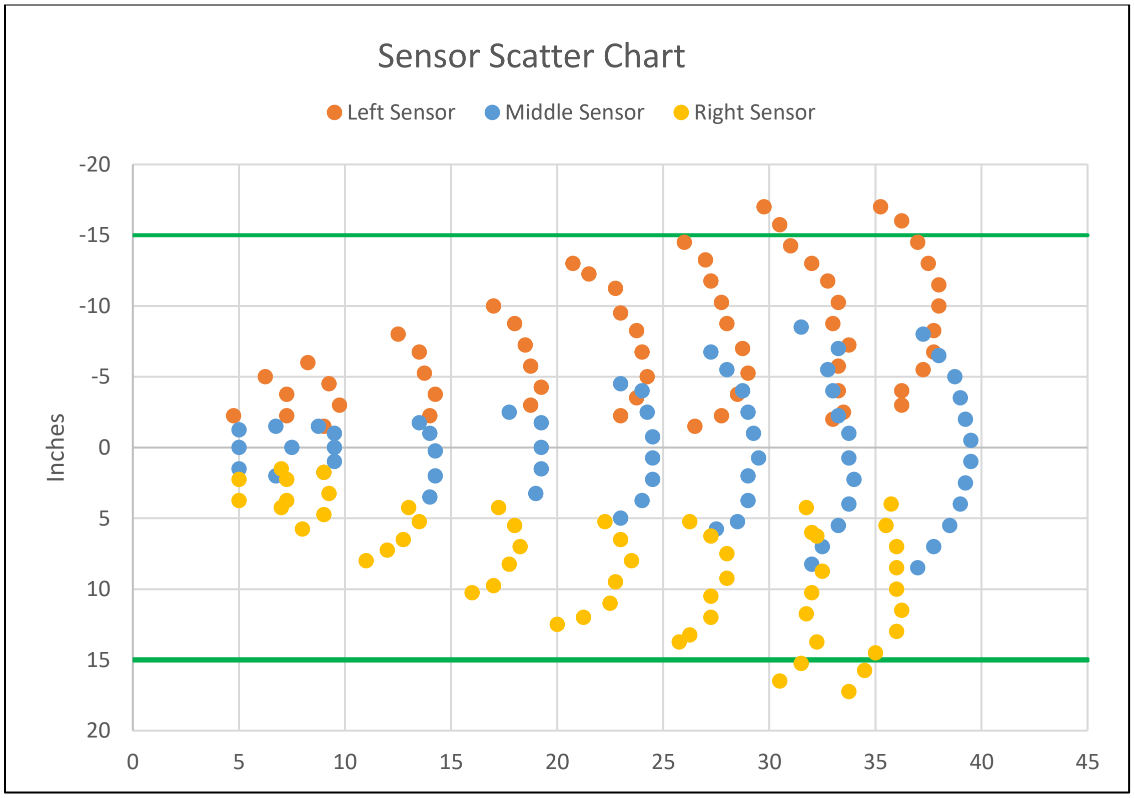


Figure 14 - Scatter Chart of Individual Sensors

Preliminary testing of the ultrasonic sensor array for autonomous control was conducted with Prototype 1. A simulated corn row lab test course was created using forty 1.5” x 12” poplar dowels placed with spacing of 30” width and 6” between corn plants. The rover was placed at a 45° heading into the row to determine if it could correct its heading and maneuver through the straight row. After successfully completing the maneuver a second scenario mimicking a curved row was created with the poplar dowels. The rover successfully completed the simulated curved row at speeds of 1mph and 2mph under complete autonomous control, Figure 15.

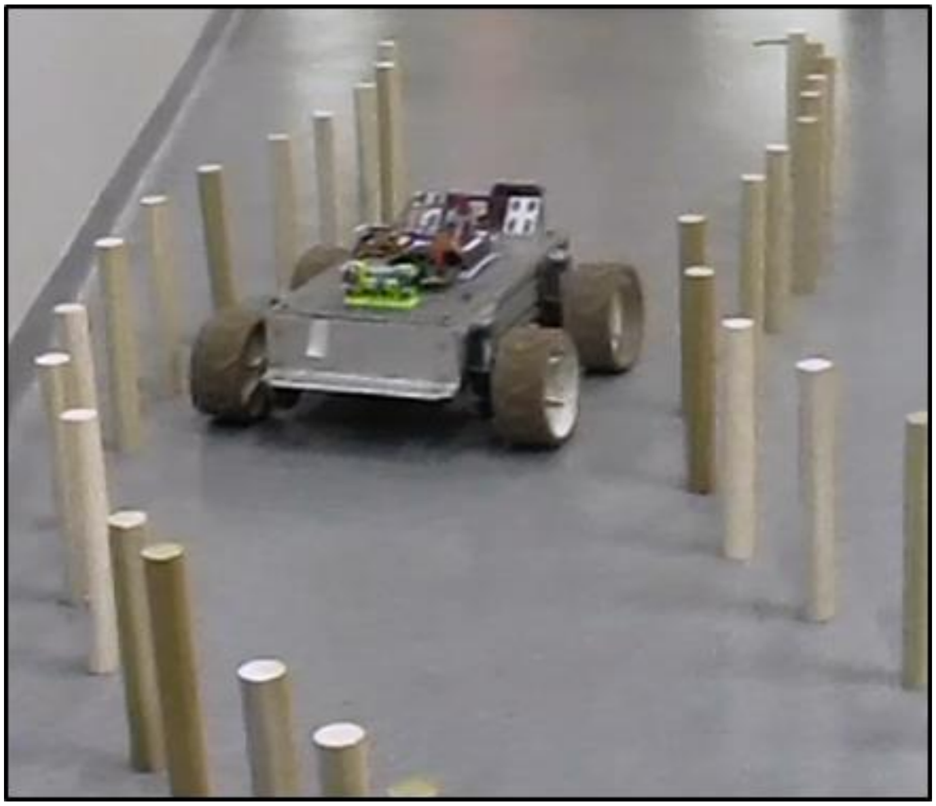


Figure 15 – Prototype 1 during Curved Row Testing – Autonomous Operation

Inter-Row Maneuvering

After addressing the challenge of intra-row maneuvering research was conducted regarding inter-row maneuvering which is more difficult. Many configurations for robotic scouting vehicles in research test plots exist, but none of the designs take into account operation in a conventionally planted corn field that has headlands. For these designs to operate in a field with headlands, valuable crops are run down when the vehicle travels from one row to the next row.

After assessing many different vehicle designs the primary goal was to choose a design that could best meet the goal of being minimally invasive. Conceptual designs were brainstormed and analyzed for their potential benefits and drawbacks. In Figure 16 a tank track configuration is proposed on the left and a crawler track configuration is shown on the right. The benefit of the tank design is that it would allow the vehicle to drive over larger obstacles with the raised front edge. The benefit of the crawler design is a larger contact area with the ground and a lower center of gravity which would contribute to improved stability. A conceptual model of the design was created to demonstrate how the autonomous scout vehicle could change its configuration to accomplish inter-row maneuvers, Figure 17.

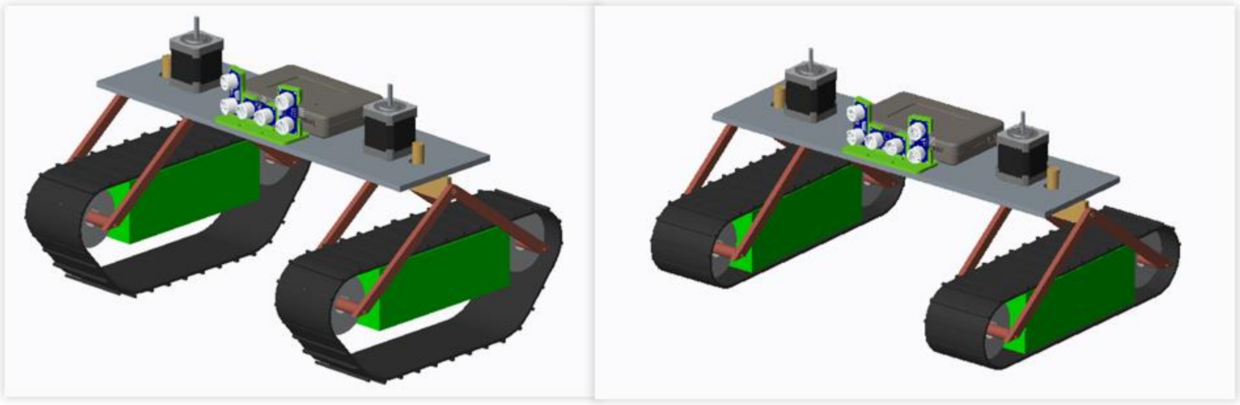


Figure 16 - Conceptual Model of Rover: Left - Tank, Right – Crawler

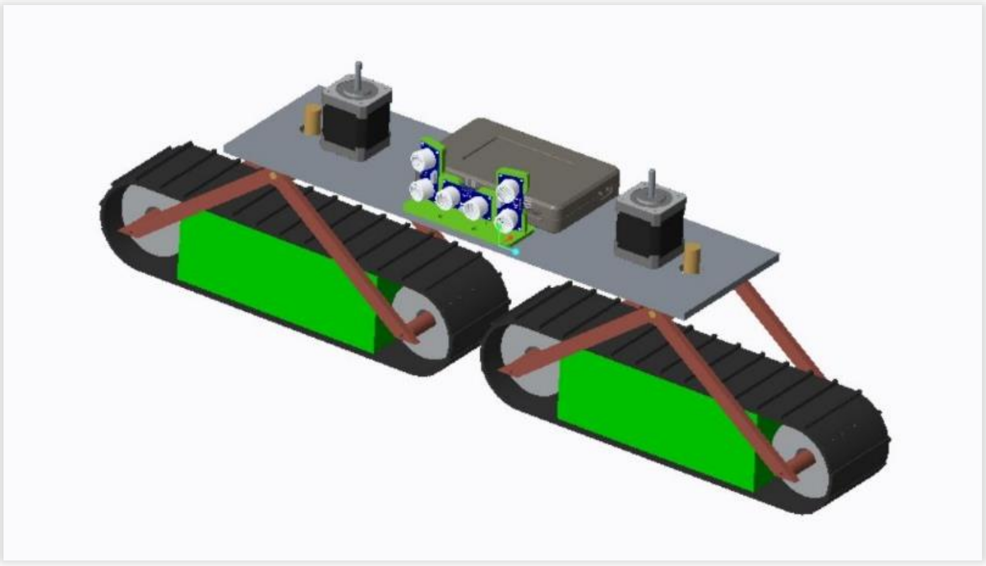


Figure 17 - Narrow Profile with the Rover in Linear Configuration

A two track design was chosen with a base vehicle width of 20” and tracks that were 14” long. The crawler design was chosen to prioritize stability and minimize compaction with a larger footprint. The vehicle would have tracks in a parallel configuration for traveling down rows (intra-row) and would change into a linear configuration to go between rows (inter-row). The design consists of two tracks and a main frame supporting the controller and scouting tools. The batteries and motors are housed inside of the track assembly. Two motors mounted on the main frame turn the track assembly during the shape changing operation. During intra-row maneuvering the vehicle will be in an ‘H’ or parallel configuration. After completing the shape changing operation the vehicle will be in a linear configuration, as both tracks will rotate to become co-linear with the main frame. In this configuration the vehicle will have a maximum width of five inches, allowing it to travel between two corn plants spaced at six inches or through any obstacle with an opening as narrow as five inches. When in the linear configuration, the vehicle will be 29” long. With the frame and the tracks of the vehicle having a width less than 6”, the vehicle should be able to easily maneuver through corn fields with high plant populations.

The linear configuration gives the robotic vehicle the ability to go over long ruts or gullies as the weight is transferred on the tracks.

Vehicle Configuration

With knowledge of the environment that the robotic vehicle would be working in and the vehicle design chosen, steps were then taken to research the necessary components required for a shape changing robot and different modes of operation for the different configurations.

Mechanical Components

For Prototype 2, modular tracks were chosen so they could be adapted to any length to accommodate design changes. The widest available tracks with a width of 3” were chosen for their ability to reduce compaction with a larger footprint. The width of the tracks also factors into the stability of the vehicle when in the linear formation, which is important to keep the vehicle upright. Additional design considerations such as keeping a low center of gravity were implemented to improve the stability of the vehicle when in the linear configuration.

A Multistar Lithium ion 22.2V battery with six cells and 16000mAh is housed within each track assembly Appendix D - . The lithium ion batteries provide a dense energy supply that when placed inside each of the track housings contributes to a low center of gravity. The output connector of the battery is a female XT 90, connected to a male-female XT90 to XT60 adapter. The 16 Ah batteries fit the space constraints for width and height to fit inside of the track and still allow for maximum track length in the linear configuration.

Each track is driven by an EC (electronically commutated) motor coupled with a planetary gearhead, as seen in Figure 18. The EC motor is a brushless DC motor with digital Hall sensors allowing for high efficiency. The 12V motor has a maximum efficiency of 76%, maximum continuous torque of 54.9 mNm, and a no-load speed of 4360 RPM. The two stage planetary gearhead has a 42 mm diameter, a reduction of 26:1, and a maximum efficiency of 81%. The motor coupled with the reduction has a maximum continuous torque of 1427 mNm, a no-load speed of 167 RPM, and an efficiency of 61.5%.

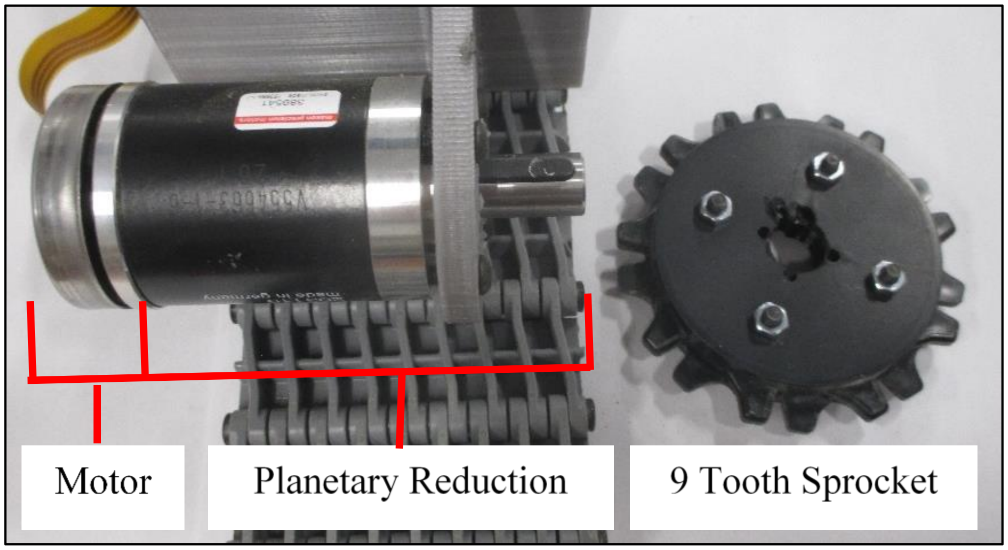


Figure 18 - Motor and Planetary Gear Reduction Mounted to Track Housing Frame

The motor is controlled by an ESCON Module 4-Q servo controller that has a continuous output current of 2A and a maximum output current of 6A. The ESCON Module is mounted onto a motherboard, shown in Figure 19, which has screw terminals for connecting to the EC motor. The motherboard has an operating voltage between 10-24V. This allows the lithium ion battery to directly plug into the motherboard via XT60 connector and reduce wire congestion. PWM control signals are sent from the myRIO to both the left and right tracks to the motherboard to enable and set the speed of the motors. The motherboard utilizes analog output pins to return speed values read by the Hall sensors back to the myRIO.

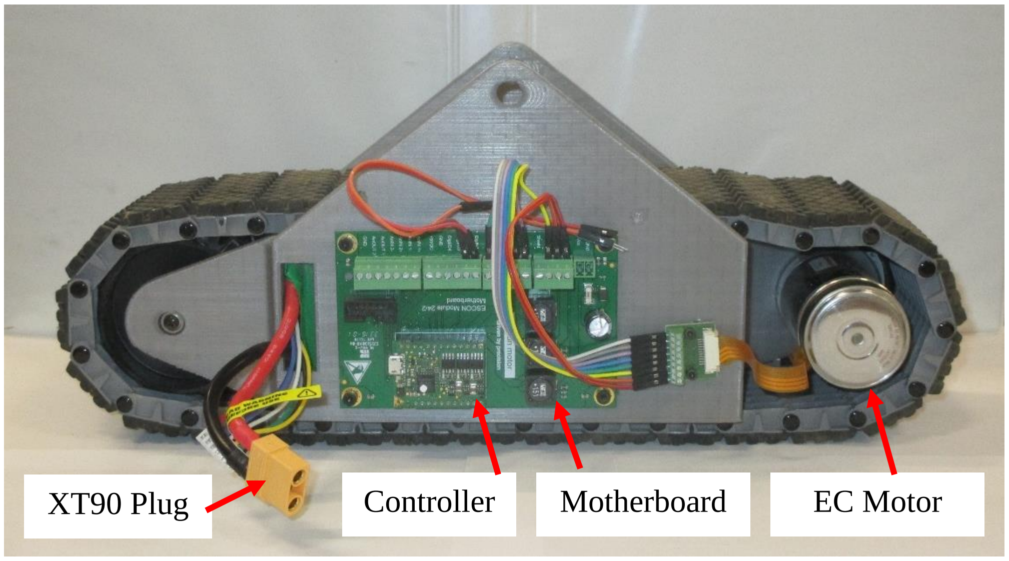


Figure 19 - Power Supply, Controller, Motherboard, and Drive Motor

Lynxmotion 9 tooth track sprockets are coupled to the output shaft of the planetary gear reduction via a 3D printed hub. The sprockets diameter is 2.5 inches and is bolted to the hub. The outer diameter of the tracks going around the sprocket is 3.75 inches. At the other end of the track unit two sprockets are coupled to a Lynxmotion passive idler hub (Figure 20). The sprockets drive Lynxmotion modular tracks that have a width of 3” and a pitch of 1.07”. The undercarriage of the tracks that engage with the sprockets is polypropylene while the part in contact with the ground is rubber. Twenty-nine tracks are connected with polypropylene axles and snap rivets to hold the axles in.

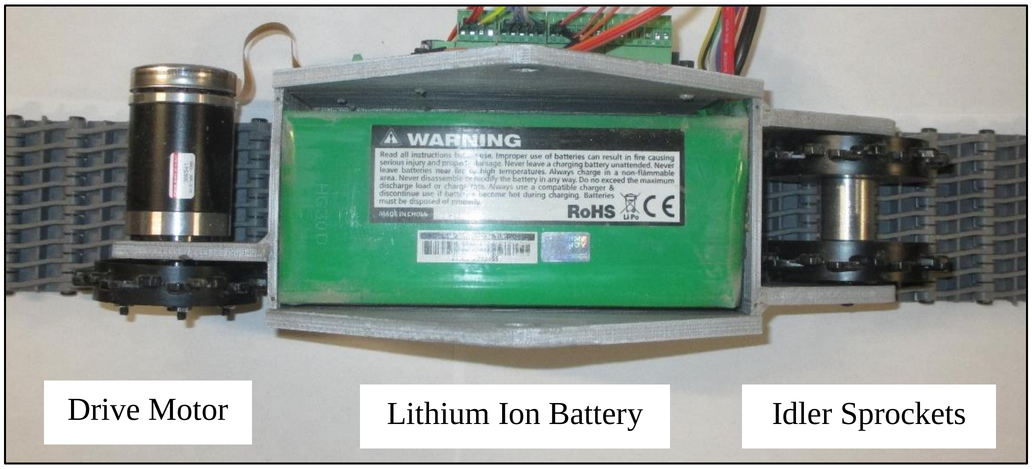


Figure 20 - Top View of Track Bogey

The gear and motor combination and the idler hub are mounted to a 3D printed PLA bogey frame. The plastic frame has space to house the lithium ion battery with tabs that the idler and gear-motor combination are mounted on. The plastic frame is connected to the main body by bolting to a hinge mount with dimensions of 4.25” x 3.25”. The hinge mount is 3D printed PLA as well and allows the bogey to have two degrees of freedom: 360° rotation around the Z axis and 15° rotation around the Y axis. The hinge mount is connected to a 3x3” ball bearing turntable which allows for the rotation in the Z axis (Figure 21). To control the rotation about the Z axis the hinge mount is also bolted to an 8mm hub that goes thru the middle of the turntable where it is fastened to the output shaft of the turret motor with a set screw.

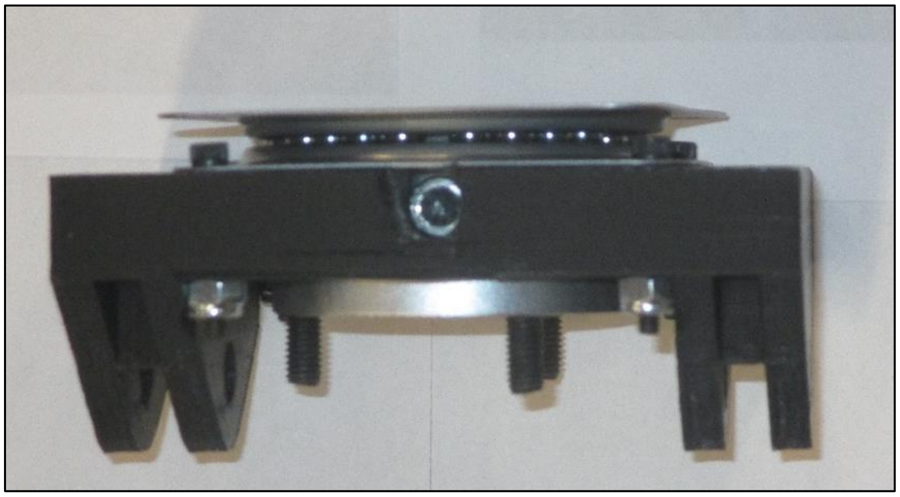


Figure 21 - Ball Bearing Turntable, Hinge Mount, and 8mm Hub

The main frame of the vehicle is ¼” aluminum with a width of 4” and 19.5”. Two holes with a diameter of 1.25” are cut 2” from each end to allow the turret motor shaft and 0.315” (8mm) hub to connect. The turret motor is mounted to the top side of the aluminum with a 3x3” steel bracket. The 3x3” ball bearing turntable is bolted to the aluminum from the bottom side of the aluminum frame.

The turret motor, shown in Figure 22, is a DC gear motor with a 264:1 reduction which allows it to output 416oz-in of torque with an output speed of 10 RPM. The turret motor has a Hall effect encoder to count revolutions of the motor. The steel bracket allows the turret motor to be fixed on top of the aluminum frame while leaving space for an 8mm hub to be connected to the track bogey on the underside of the aluminum frame. Both turret motors are controlled by a L298 Dual H-Bridge 12V DC motor controller, allowing for bi-directional rotation. The L298 motor controller receives enable and speed commands from the myRIO in the form of digital high/low and PWM signals. The turret motors can rotate the track bogeys 360° which allows the vehicle to change from a parallel configuration to a linear configuration.

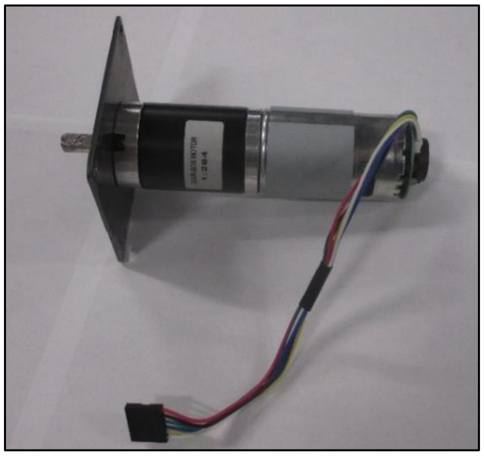


Figure 22 – Turret Motor with Mounting Plate.

Prototype 2 has dimensions of 14” x 19.5” when in the parallel configuration shown in Figure 23. After conversion to the linear configuration the dimensions change to 4” x 29”. The weight of Prototype 2 is 25 lbs. with each individual track weighing 7.5 lbs., which contributes to the vehicles low center of gravity. The tracks are 3 inches wide and are in contact with the ground for 10.75 inches for a contact area of 32.25 in^2. The pressure that the robot has upon the ground is .39 lbs./in^2. The cost of Prototype 2 is $2,303.98 and it has a top speed of 1 mph.

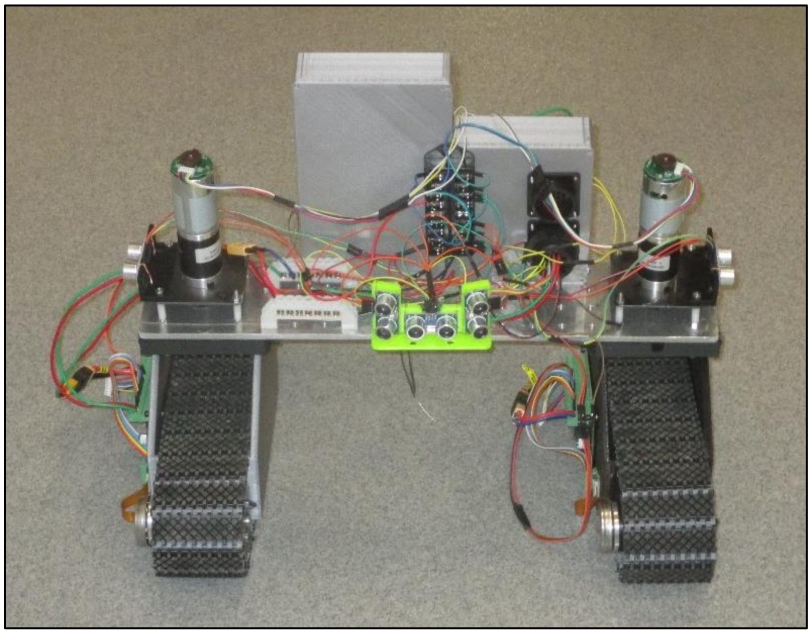


Figure 23 - Prototype 2 Front View

Program and Controls

The HiTEC Optic 5 remote control has 5 radio channels; two four-way stick controllers and a push button switch. Radio signals are sent to a Minima 6T receiver which then relays the signals to the myRIO. The right stick on the controller is self-centering upon release for both fore and aft movement as well as left and right movement, and is used for throttle and steering respectively. The left stick is self-centering upon release for both left and right movement and this channel is set up to send signals for the turret motors. The left stick fore and aft control can be set to hold any position. It is currently set up to be an on-off signal for the autonomous control, while simultaneously setting the throttle for the autonomous operation. The push button switch is a power on/off switch for all motor operations.

Table 1 - HiTEC OPTIC 5 Remote Control Commands

Control	**Fore**	**Aft**	**Left**	**Right**
Left Stick	Autonomous ON	Autonomous OFF	Turrets CCW	Turrets CW
Right Stick	Forward	Reverse	Turn Left	Turn Right
Push Button	High = On	Low = Off		

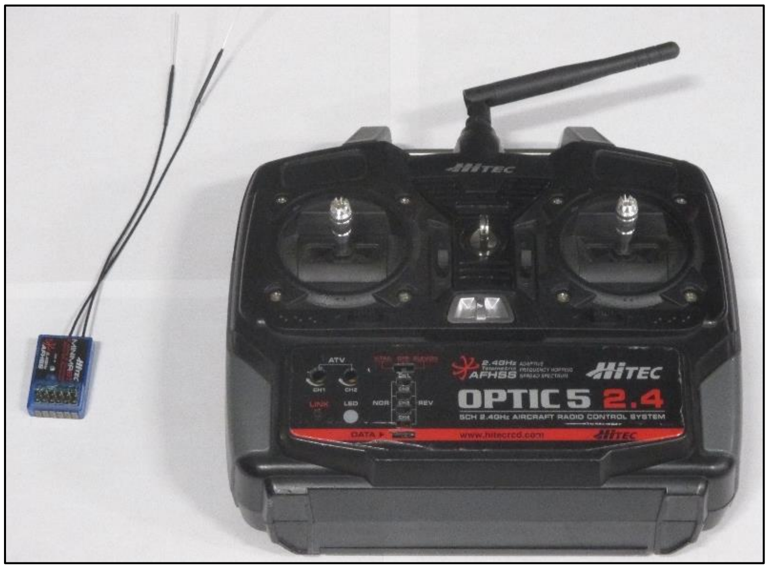


Figure 24 - Minima 6T Receiver and OPTIC 5 Remote

The signals from the Minima receiver are read by the myRIO. The radio signals are PPM (Pulse Position Modulation, similar to PWM signals). The myRIO FPGA was flashed with original code so that PPM and PWM signals could be read by a myRIO. The pulse time length and pulse time high of the PPM signals read in the FPGA interface are sent to the real-time application where the control decisions are made and output signals sent to the motor controllers.

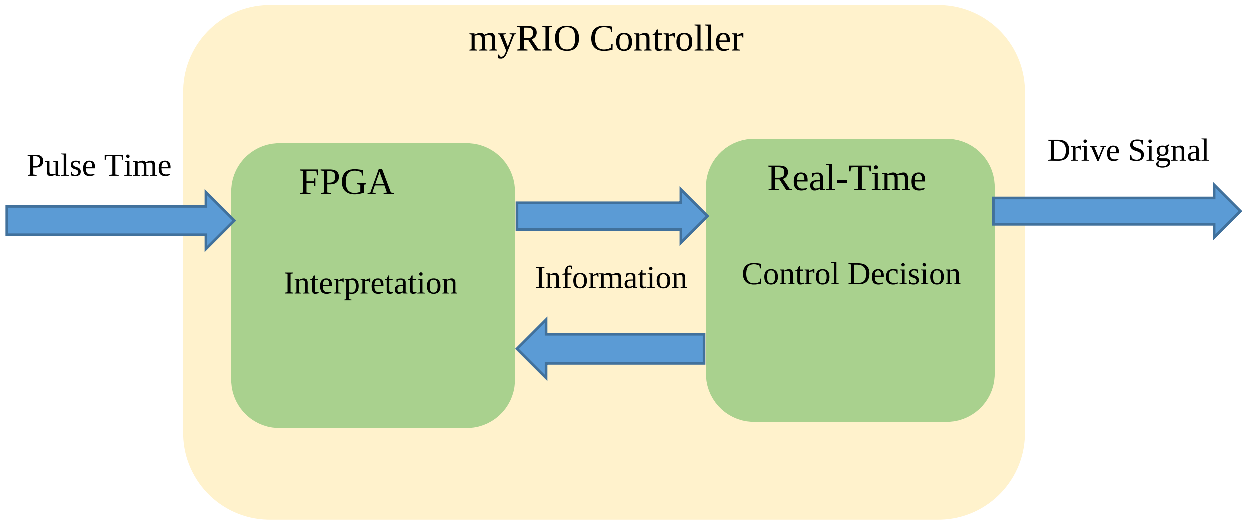


Figure 25 - myRIO Controller Signal Processing

For skid steering throttle and steering signals are merged to vary the speeds of the left and right tracks, Figure 26. When the right stick is pushed up the vehicle moves forward and both tracks have the same speed. When the right stick is pushed to the left the tracks move in the opposite direction at the same speed allowing the rover to rotate counter clockwise about the z-axis of its current position.

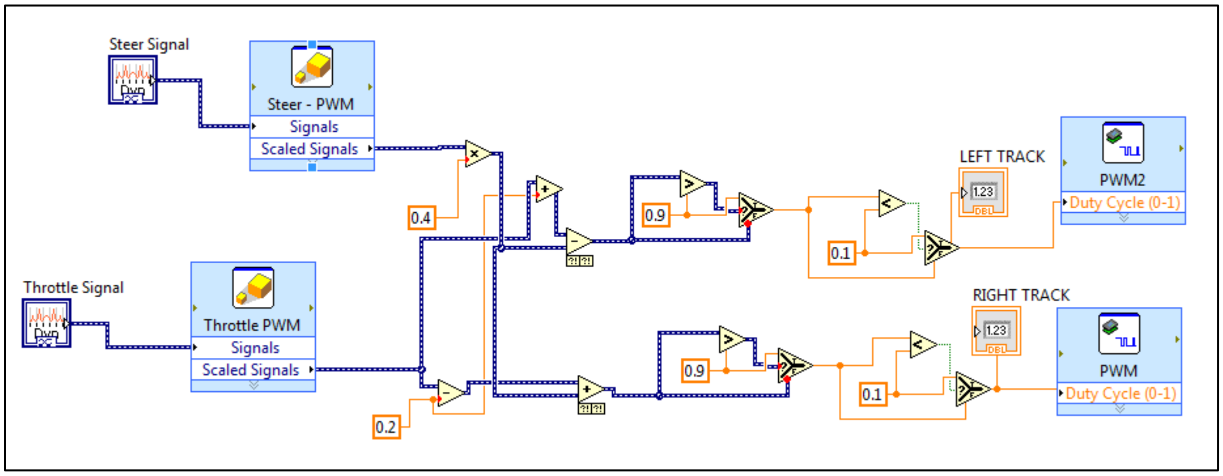
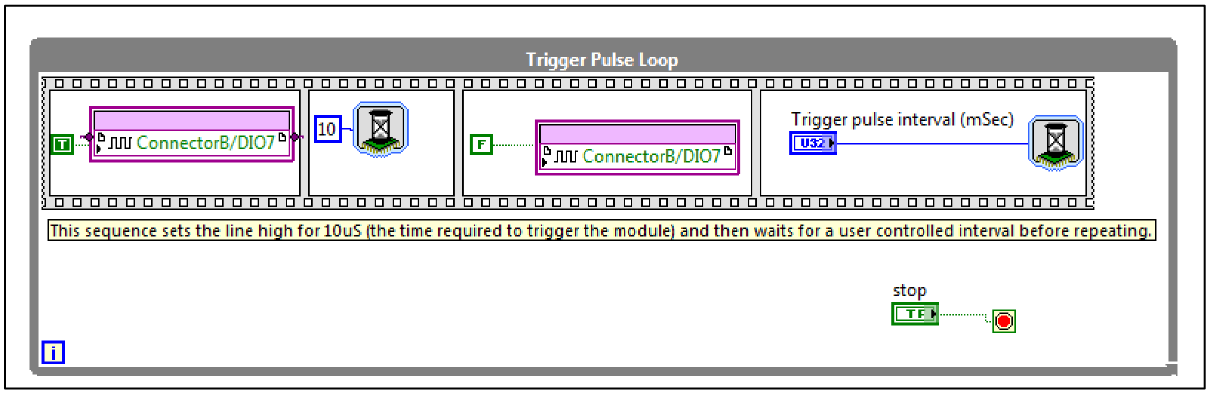


Figure 26 - Radio Input Signal Conversion to Output PWM

Within the FPGA interface code was added, compiled, and flashed to read and control the ultrasonic sensors. To trigger the HC-SR4 ultrasonic sensors the trigger pin is set to a high value (true) for 10µs and then set to low value (false). There is then a 200µs delay to allow the echo to return from nearby objects before the loop repeats, Figure 27. Upon triggering the ultrasonic sensor simultaneously sends out a sonic burst and sets the echo pin to high. When the sonic burst reflects off of an object and returns to the ultrasonic sensor the echo pin is set to low, Figure 28. The pulse width measurement loop within the FPGA returns a time value in micro-seconds which is proportional to the distance. The micro-second time value is sent to the myRIO Real Time interface where the distance to an object is calculated in inches and centimeters.



This sequence sets the line high for 10uS (the time required to trigger the module) and then waits for a user controlled interval before repeating.

Figure 27 - myRIO FPGA Trigger Loop for Ultrasonic Sensor

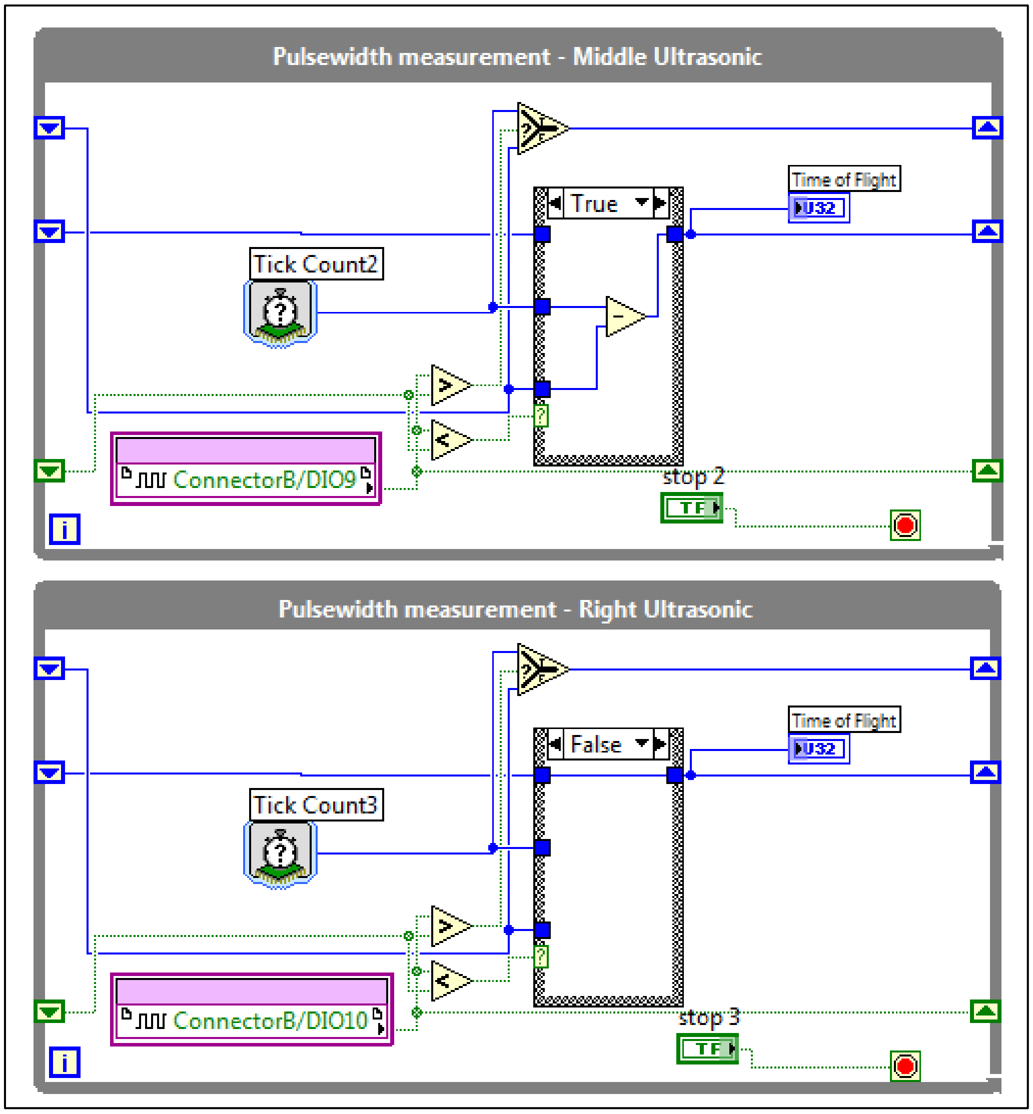
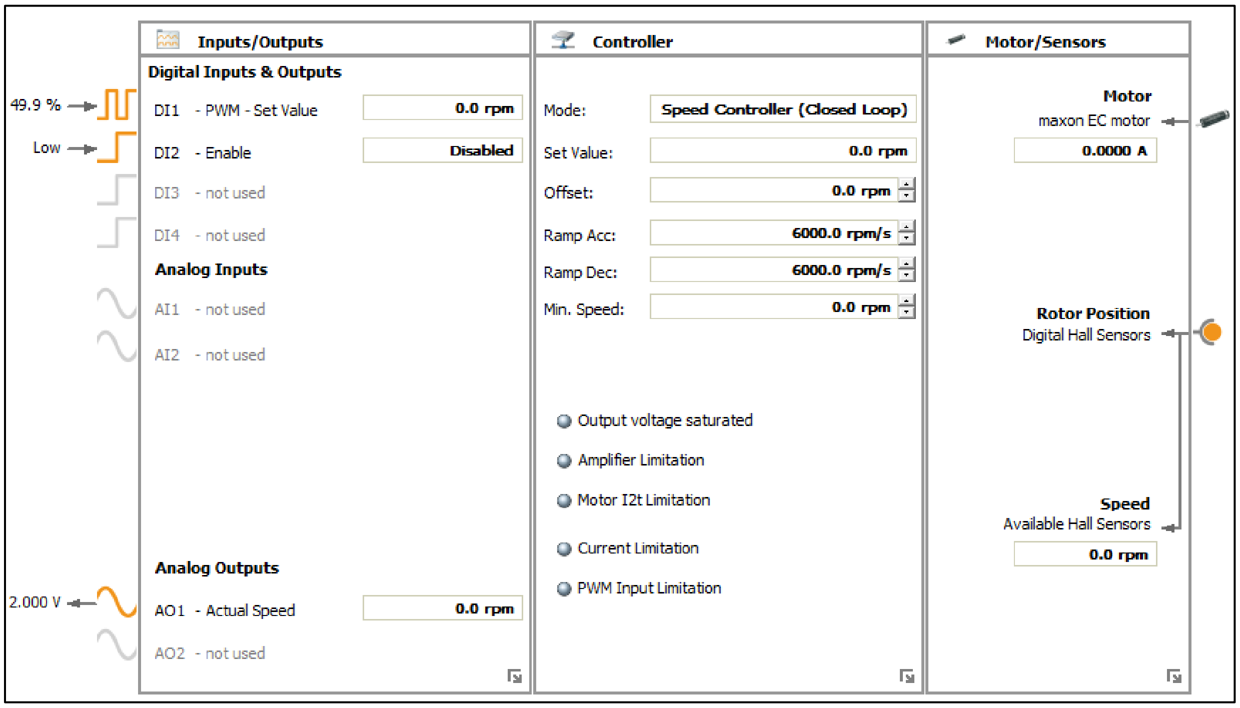


Figure 28 - myRIO FPGA Read Loop for Ultrasonic Sensor: Top –True, Bottom – False

Maxon Motor, the supplier for the drive motors, gear reductions, and motor controllers provides a servo controller program to allow end users to customize use of the motors. Upon plugging in the motor controller to a computer, a startup wizard is run allowing the user to select the type of control signals, maximum motor speed, motor acceleration, and type of encoder paired with the motor. Within the startup wizard the operator chooses from three modes of control; speed controller-closed loop, speed controller-open loop, or current controller. After all of the parameters are set for the controller the wizard takes the motor through regulation tuning to match the Hall sensors to the operating mode. The ESCON Studio Controller Dashboard allows the user to observe input and output signals, the present mode of the speed control, and motor performance, as well as understand any error that the controller reports.



Inputs/Outputs

Digital Inputs & Outputs

49.9 % DI1 - PWM - Set Value 0.0 rpm

Low DI2 - Enable Disabled

DI3 - not used

DI4 - not used

Analog Inputs

AI1 - not used

AI2 - not used

Analog Outputs

2.000 V AO1 - Actual Speed 0.0 rpm

AO2 - not used

Controller

Mode: Speed Controller (Closed Loop)

Set Value: 0.0 rpm

Offset: 0.0 rpm

Ramp Acc: 6000.0 rpm/s

Ramp Dec: 6000.0 rpm/s

Min. Speed: 0.0 rpm

Output voltage saturated

Amplifier Limitation

Motor I2t Limitation

Current Limitation

PWM Input Limitation

Motor/Sensors

Motor
maxon EC motor
0.0000 A

Rotor Position
Digital Hall Sensors

Speed
Available Hall Sensors
0.0 rpm

Figure 29 - ESCON Studio Controller Dashboard

Experiment and Analysis

Testing of the tracked robot was completed in two steps, simulations and field trials. Following the testing of inter-row and intra-row maneuvering power consumption measurements were taken. From the results of the power consumption estimates were made regarding the number of acres that could be covered by Prototype 2 at different scouting resolutions.

Simulation Trials

Simulation trials were completed using 1.5” x 12” poplar dowels poplar dowels to mimic corn plants. A simulated corn row test course was created using forty 1.5” x 12” poplar dowels placed with row spacing of 30” width and 6” between dowels. Different scenarios were created to test intra-row navigation, inter-row navigation, and combinations of both intra- and inter-row navigation. Testing showed that Prototype 2 could complete the necessary maneuvers to move between simulated rows and plants successfully (Figure 30) (Figure 31). Entering straight rows, following contoured rows, and navigating from one row to the next were completed successfully.

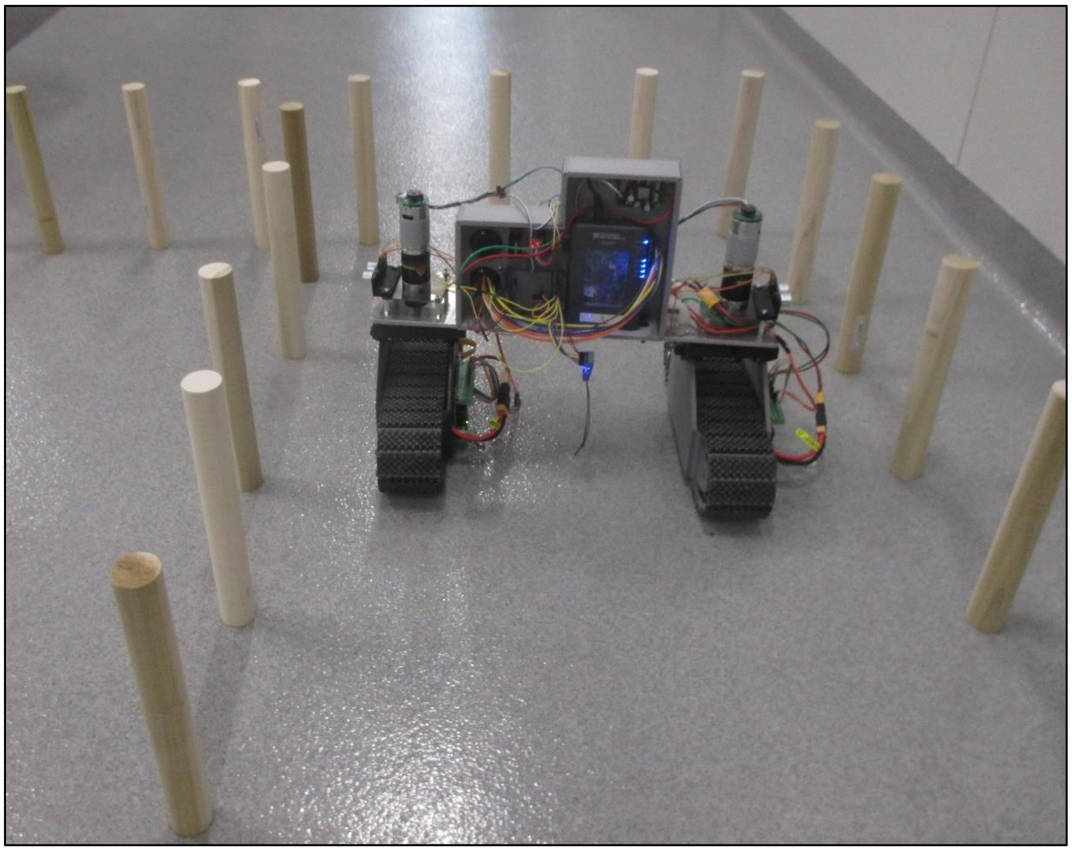


Figure 30 - Intra-row Navigation during Simulation Trail

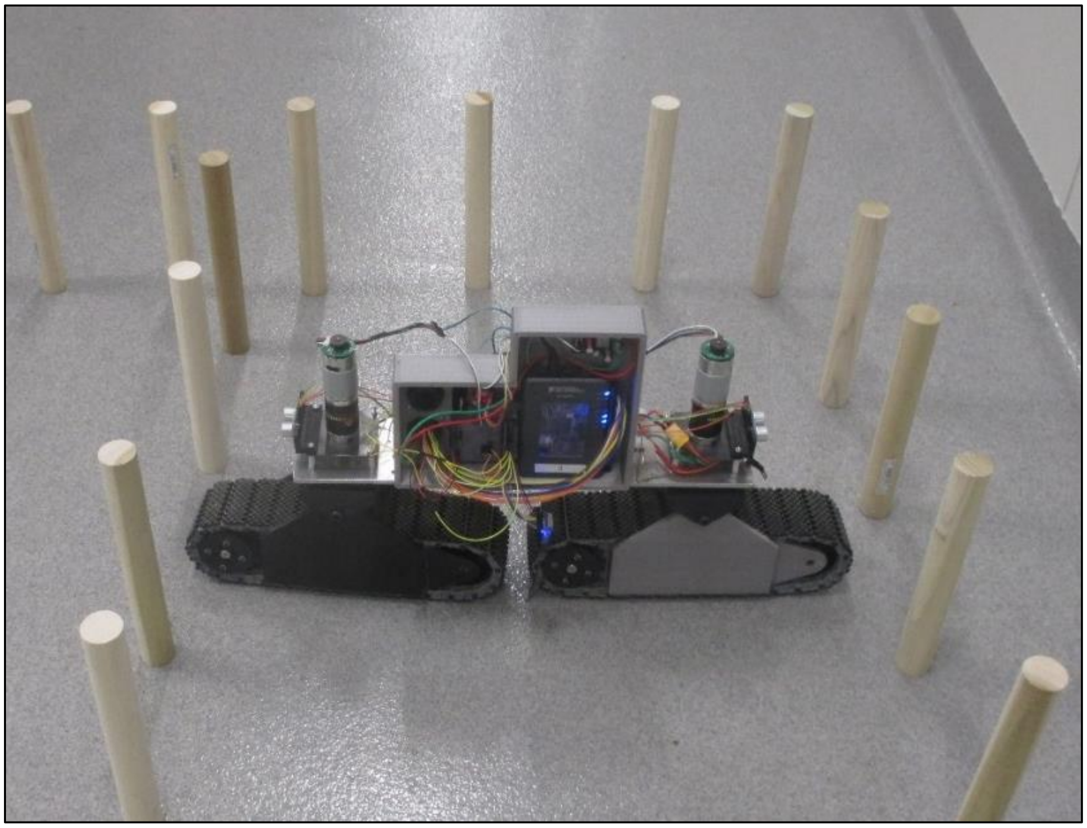


Figure 31 – Prototype 2 in Linear Configuration to Complete Inter-row Maneuver

In the linear configuration, Prototype 2 was able to navigate over a 12” gap for simulation of a pivot track rut. The simulated pivot track was created by placing two tables 12” apart and driving Prototype 2 across as shown in Figure 32. The center to center distance of the sprockets is 10.75” while the end to end length of the tracks is 14” (Figure 33). The robot can cross gaps of 12” or smaller as the front edge of the track is able to come into contact with the opposite edge before leaving the edge from where it comes. While over the gap the leading track of the robot does not have as much traction, but the rear track is able to provide power to move forward and weight to keep the robot from tipping forward over the edge. The simulated gap with the tables is different from an actual pivot track rut because it is a direct drop off the edge where a pivot track rut can be rounded at the edges.

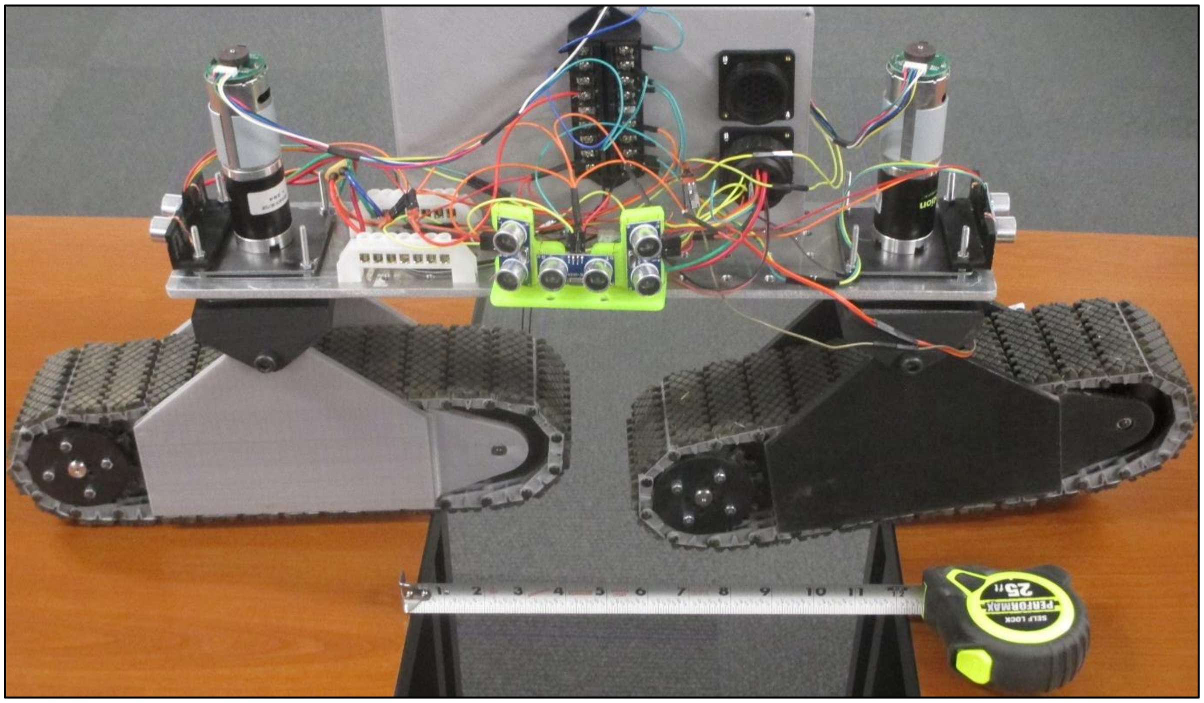


Figure 32 - Simulated Pivot Rut - 12" Gap Traversed

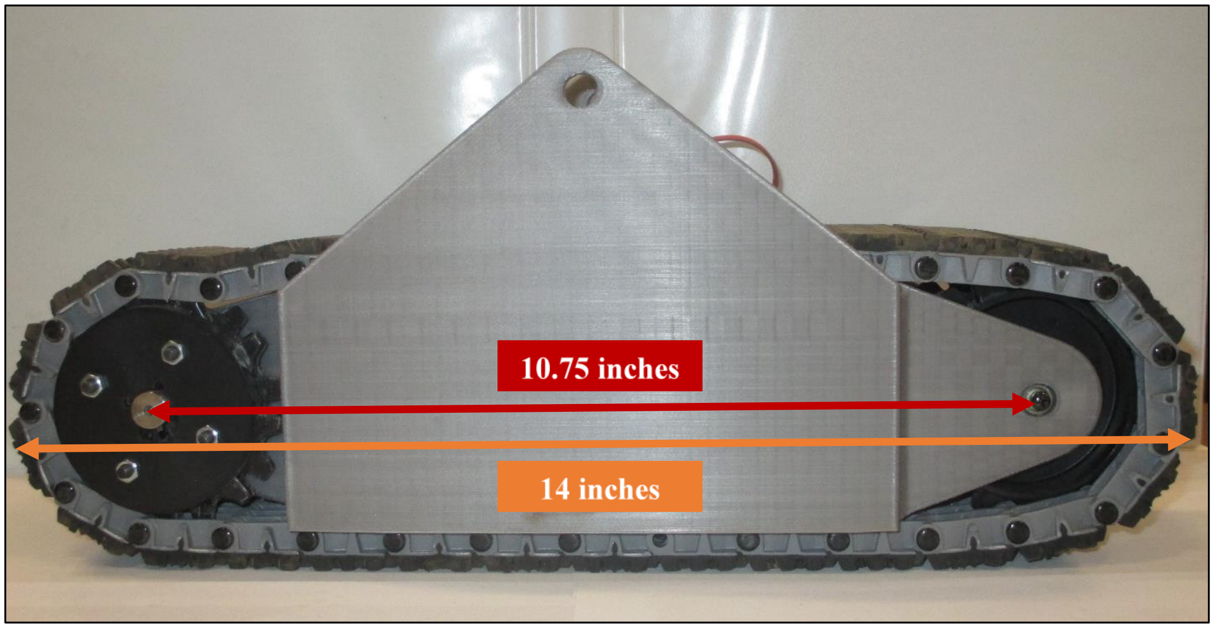


Figure 33 - Individual Track Bogey Lengths

Field Testing Trials

Field testing was conducted at the KSU Agronomy North Farm in Manhattan, KS in small plots of corn on June 9, 2017. The growth stage of the corn was V10 and after emergence the stand of corn had an approximate population of 30,000 plants per acre (Figure 34).

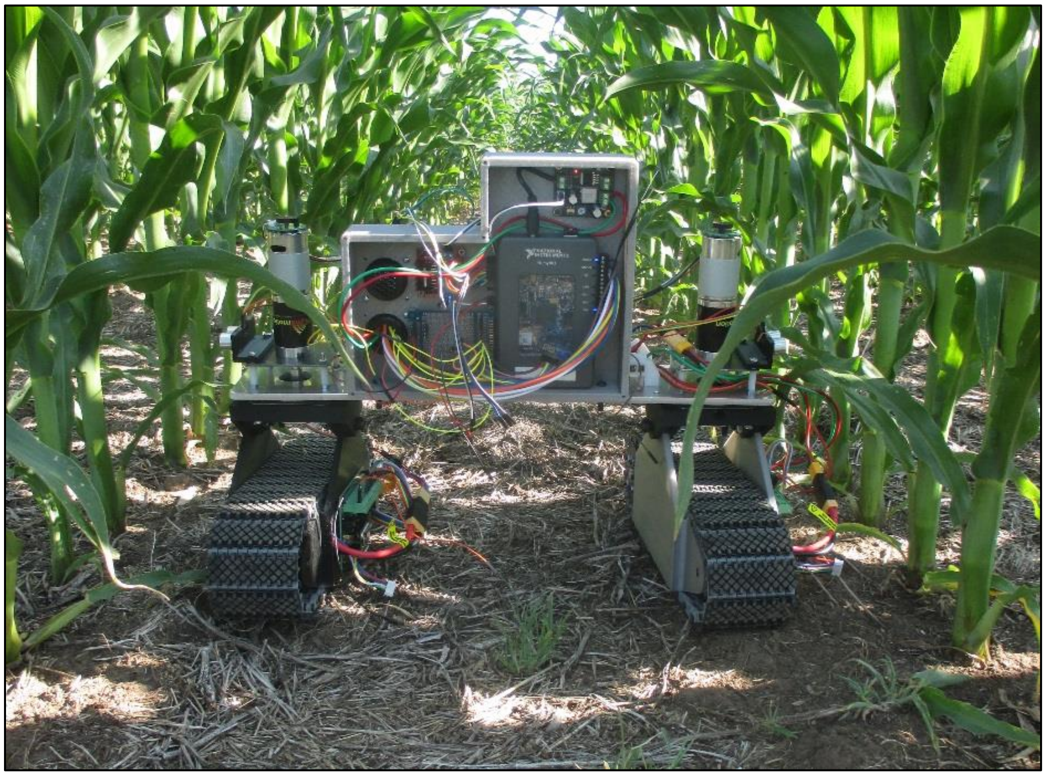


Figure 34 - Field Testing in Corn at V10.

Intra-row navigation proved to be successful as Prototype 2 had plenty of room to make corrections to turn left and right when presented with obstacles. The shaft couplers between the turret motors and the 8mm hub loosened up during testing allowing the tracks to pivot without input from the motor. This problem was solved by removing the shaft coupler and mounting the 8mm hub directly to the output shaft of the turret motor. Further testing after removing the shaft coupler showed that the tracks were able to stay fixed in their respective positions even during turning maneuvers.

The ultrasonic sensors were able to guide the scouting robot in autonomous mode down the rows of corn. Occasionally the sensors detected large overhanging leaves in the middle of the row but was able to continue moving forward past the leaves. The robot attempted to change from parallel to linear configuration but was unable to because of the shaft couplers. Testing after changing to direct mounting of the hub to the motor allowed the robot to successfully change into the linear configuration. Once in the linear configuration the robot began its inter-row maneuver as it moved between corn plants. In Figure 35 the vehicle can be seen from two different directions. The picture on the left shows the tracks two inches away from the corn stalk while the picture on the right shows the exposed motherboard caught on a stalk of corn. Due to being a prototype vehicle the motherboard was attached on the outside of the track housing for easy access. Final designs of the track housing enclose the motherboard on the inside and take into consideration shape of the housing so that it can push between plants without hanging up on stalks.

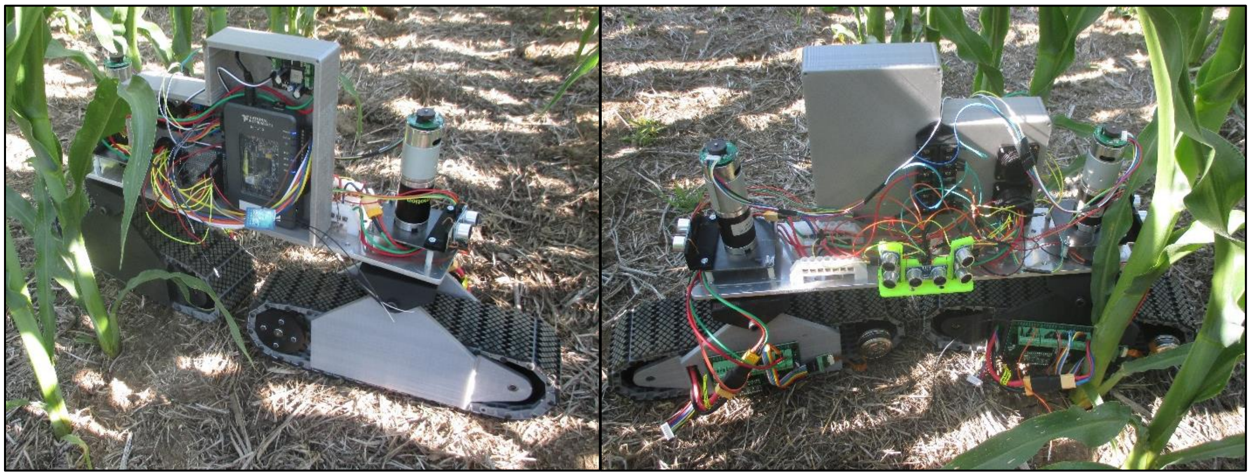


Figure 35 - Inter-Row Navigation in Linear Configuration – Test #1

The second inter-row test, shown in Figure 36, was more successful as the robot stayed closer to the stalk on the left side of the right picture allowing the motherboard and exposed wires to pass the stalk on the right side cleanly. The ultrasonic sensors facing the rows proved to be able to find gaps that were large enough for the rover to pass through, though the success of the inter-row maneuver depended upon how close the vehicle was to the middle of the gap.

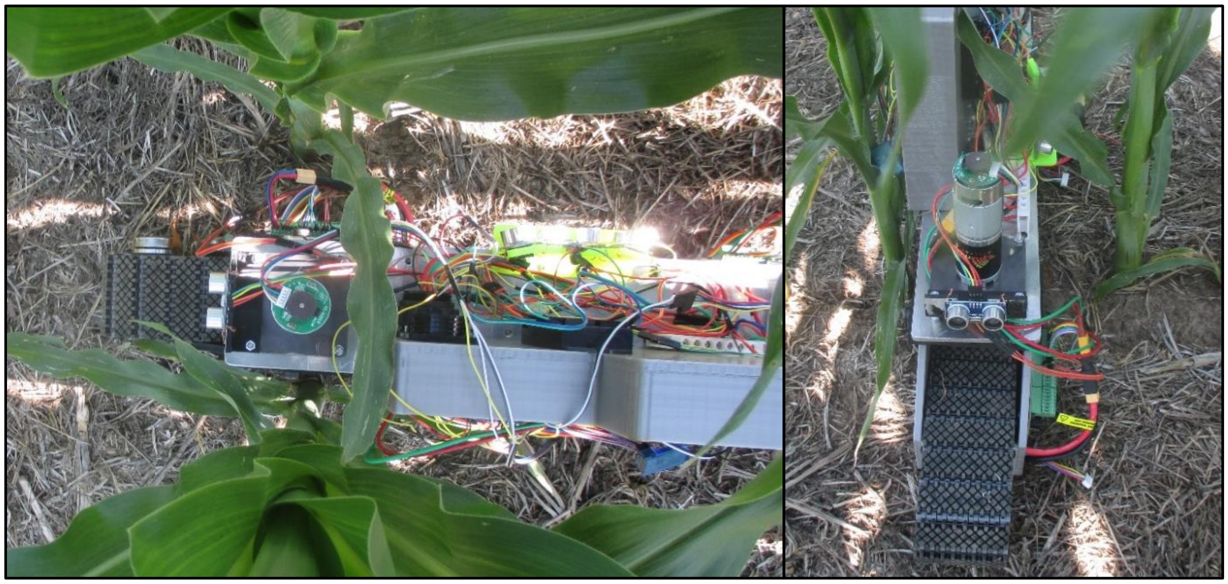


Figure 36 - Inter-Row Testing Top View – Test #2

Power Consumption Measurements

The MultiStar Lithium-Ion battery is a 22.2 Volt – 16 Amp-Hour for an ideal total power supply of 355.2 Watt-Hours. The discharge profile of lithium ion batteries is shown in Figure 37 for different discharge rates (Rushworth, 2015). The Multi-Star battery has a maximum allowable discharge rate of 10C. 1C is equivalent to the current required to discharge a lithium ion battery in one hour, which for the chosen MultiStar battery 1C = 16 Amps.

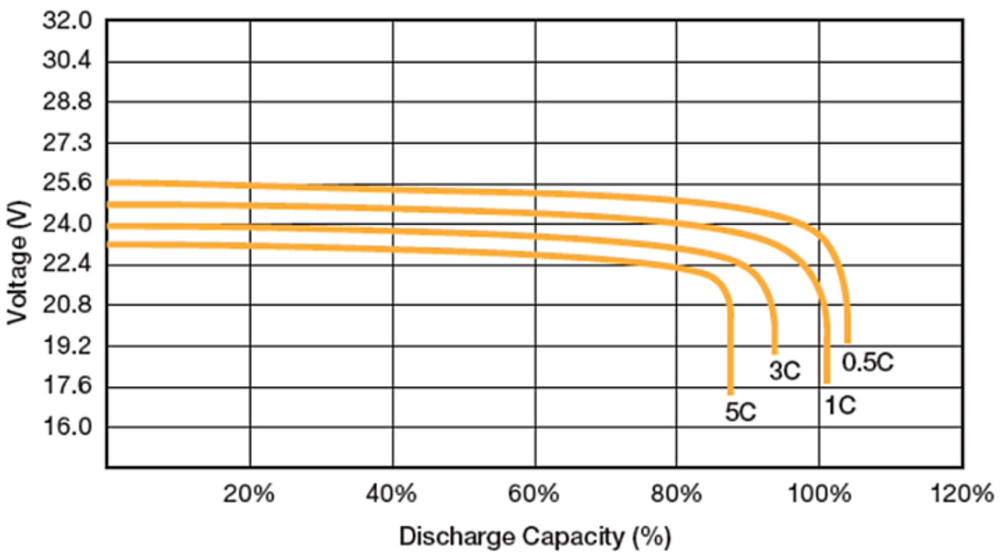


Figure 37 - Discharge Profile of a Lithium Ion Battery

The robot was tested on different terrains to calculate the power usage. Figure 38 shows the rover at rest consuming a current of .22 Amps and 22.88 Volts, resulting in power usage of 5 Watts. The current and voltage were measured at the right track where the right battery supplies power to the right motor, the myRIO, and both turret motors. The current that was being drawn at rest is due to the myRIO and operating current for the ultrasonic sensors. The robot was tested at its top speed of 1 mph for all of the terrain trials.

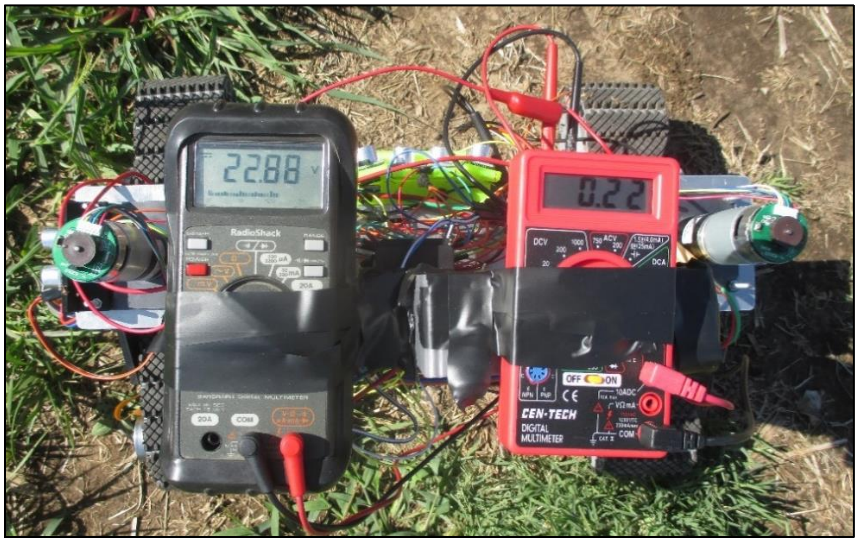


Figure 38 - Measuring Power Consumption. Left-Voltmeter, Right-Ammeter

The robot was tested on various terrains to study the power consumption rate shown in column four of Table 2. The rate of power consumption determines how far the robot will be able to go with the available power supply. Testing was first conducted in the lab on a hard and flat surface with no variability. The robot was then taken to the KSU Agronomy North Farm for trials in test plots on July 1, 2017. The first field trial was conducted traveling between two rows of corn over a 50 ft. stretch of terrain. The terrain was mostly flat but small bumps caused spikes and valleys of voltage and current during the observation. The average observed voltage and current was recorded after four passes. The second field trial was conducted by driving the robot through wheat stubble with an approximate height of 12 inches. The height of the stubble created resistance against the main frame, causing higher power consumption. The third field trial was conducted in uncut grass with an approximate height of 7 inches. The terrain during the third test had the most variability, especially when the robot drove over large clumps of grass, causing peaks and valleys in the observed voltage and current demand. Ending the discharge of the battery at the 80% discharge capacity keeps the lithium battery in a safe zone. At a maximum discharge rate of 1.6 amps the drive motor and myRIO are pulling current at .1C from the battery which would be a higher efficiency than the .5C shown in Figure 37.

Table 2 - Power Consumption for Various Terrains

Terrain	Voltage (V)	Current (A)	Power (W)	Ideal Operation Time (Hrs.)	80% Discharge Time (Hrs.)
Tiled Floor	22.30	1.00	22.30	15.9	12.72
Corn Field	22.40	1.20	26.88	13.2	10.56
Wheat Stubble	22.26	1.60	35.62	10.0	8.00
Unmowed grass	22.34	1.35	30.16	11.8	9.44

Duty Cycle – Scouting Resolution

The resolution that the farmer or research plans to scout their field at will determine the acreage that can be covered by the Prototype 2 in a day. From the power consumption information listed in Table 2, the robot can run for approximately 10.5 hours in a corn field and stopping when the battery is at 80% discharge. The information presented below depicts how the chosen resolution will affect the required run time.

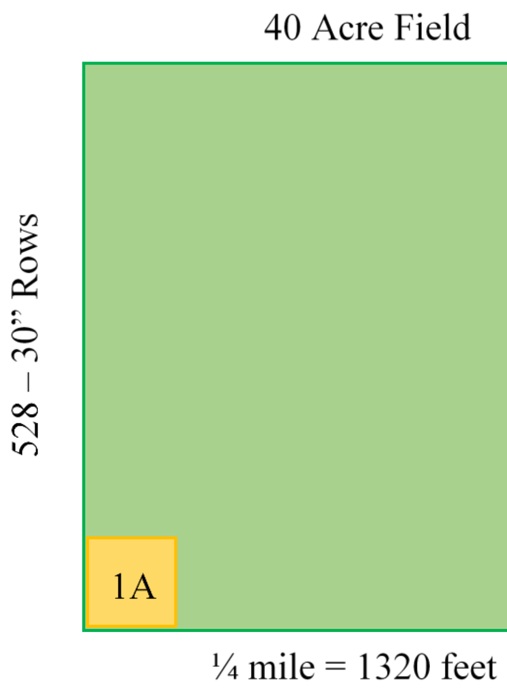


Figure 39 - 40 Acres Field - 1 Acre in Yellow

Considering the depiction of a 40 acre field shown in Figure 39 with dimensions ¼ mile by ¼ mile or a 1320 ft. square. For corn planted at 30" rows there will be 528 rows that go across the field from left to right. 528 rows that are a quarter of mile long results in 132 miles of corn rows that could potentially be scouted. Prototype 2 travels at 1mph and could traverse every row in 5.5 days if no stops were taken. By changing the scouting resolution from 1/1 (rows traveled/rows available) to 1/20 allows the scouting vehicle to complete the field in 6 hours. At a resolution of 1/20 the scouting vehicle will pass through every acre four times, as an acre is 83 rows wide for 30" rows. Table 3 shows the difference in coverage areas for Prototype 2 traveling at 1mph for 10.5 hours at different scouting resolutions through a corn field. The last column of Table 3 summarizes how many scouting vehicles with the characteristics of Prototype 2 would be required to scout a 160 acres field in a day.

Table 3 - Coverage Ability for Scouting Resolution

Resolution (Row/Rows)	Coverage (Acres/hr)	Daily (Acres/charge)	# of Robots to Scout 160 Acres Daily
1/1	.3125	3.28	49
1/5	1.625	17.06	10
1/10	3.25	32.50	5
1/20	6.50	68.25	3

Conclusion

This research presents a vehicle design that can navigate intra-row, inter-row, and around obstacles. The ultrasonic sensors proved to be effective in detecting the simulated cornstalks and maneuvering down lengths of the simulated rows. During testing in simulated rows the rover was able to correct its heading until it was headed straight down the rows and was able to follow the contour of the curves without overcorrecting. Row navigation was accomplished through a configurable vehicle design that operated in parallel configuration for intra-row maneuvering and linear configuration for inter-row maneuvering. Ultrasonic sensors provided a robust method for avoiding obstacles and guiding the robot through the corn field. The configuration change allowed the vehicle to drive between corn plants so that the robot could move from one row to the next row. Enclosing the motherboard, controller, and electrical wires within the housing will prevent them from hanging up on stalks, allowing the frame of the tracks to pass between the plants more easily

Inclusion of high capacity lithium ion batteries give the robot the ability to run continuously for extended periods of time up to 10.5 hours in corn fields. The scouting resolution that is chosen for the field will determine the acres that can be covered by the vehicle before needing to be recharged. While the robot does have two large lithium ion batteries is has minimal compaction in the field as the tracks provide a large contact area resulting in a .39 lbs./in^2 pressure. The cost of Prototype 2 is $2,303.98 produced with off the shelf parts at retail price. This put the vehicle in economic cost range for production and use in fleets.

Future research includes analysis of the vehicle concept in other row crops in order of importance; sorghum, soybeans, and sunflowers. More information regarding the power consumption and maneuvering methods in these crops will be particularly useful. Testing of the Prototype 2 capabilities in controlled settings for future operation in the field should include: operation on side slopes, power consumption when driving perpendicular to an incline, power consumption during configuration changes, and overall run time.

Other research to be conducted with the vehicle will include operation with the pest identification tools for further proof of concept. In addition to the testing of the effectiveness of the pest sensors, the effect of scouting resolution in the field will also be a large focus of research. Additional future work that will be required to bring the concept to producers will be managing and controlling fleets of scout vehicles in the same field.

Chapter 3 - Simulation of Path Planning Optimization

Introduction

The future of agriculture has the potential to use autonomous/robotic vehicles to complete tasks on farms using several smaller vehicles in a fleet. One component of coordinating multiple vehicles is to be efficient in regards to time and energy used by the robotic vehicles. The efficiency of the system can also be characterized by the shortest distance that must be traveled for the coordination to occur. The problem of calculating the shortest distance between multiple points can be characterized as a traveling salesman problem (TSP). The TSP is a study of finding the shortest path for a "salesman" to travel between "cities" in an effort to be more efficient (Potvin, 1996).

The traveling salesman problem has been studied in many other forms, one of which is the more complicated non-stationary traveling salesman problem (NTSP). The challenge is to solve for the shortest path between multiple targets that are moving in different directions and different speeds to find the optimal (shortest) path (Jiang, Sarker, & Abbass, 2005). In most circumstances coordination will be required between different autonomous vehicles. One such scenario is a fleet of autonomous scouting vehicles needing batteries swapped at different locations on the go while scouting and monitoring a field.

Many other scenarios were brainstormed to show the motivating reason for studying the Non-Stationary Traveling Salesman Problem in regards to agricultural purposes. Table 4 shows the multiple scenarios for an agricultural NTSP. In all scenarios there is the potential for multiple Primary Vehicles that need to be coordinated for a task. The Primary Vehicles represent "moving cities" in the traveling salesman problem. A path plan and coordination order would be generated for the Support Vehicle as it moves between the Primary Vehicles to refill or empty depending upon the task.

Table 4 - Applications of Non-Stationary Traveling Salesman Problem in Agriculture

Task	Primary Vehicle	Support Vehicle
Battery swap on Caretaker rover	Rover	Battery Tender
Refilling of autonomous planter	Planter	Seed Tender
Silage Chopping	Chopper	Silage Wagon
Harvesting of grain	Combine	Grain Cart

A simulation of the NTSP was created to solve for the optimal interception path of the seed tender, "traveling salesman", between the autonomous planters, "moving cities". For traditional NTSP the vector of the targets does not change over time, whereas in an agricultural field the target, the autonomous planters would change vectors when they turn at the ends of rows. The benefits of researching this different use of NTSP include optimized efficiency in other applications in agriculture such as; several combines and grain cart, silage chopper and silage wagons, and refueling of tractors.

Goals and Objectives

The objective is to create a simulation model of the Non-Stationary Traveling Salesman Problem (NTSP) in relation to agricultural vehicles performing a task requiring coordination in a field. NTSP is a more complicated form of the Traveling Salesman Problem, where the shortest path between multiple "cities" is calculated for a "salesman" to travel between said "cities". Calculating the shortest path minimizes the total time that the "salesman" spends traveling between the cities, thus increasing the efficiency of the "salesman". The traditional Non-

Stationary Traveling Salesman Problem involves “moving cities” or “targets” that are moving in different directions and potentially varying speeds that the “salesman” must reach.

The largest challenge in creating this simulation model is related to agricultural fields. While agricultural vehicles in a field move back and forth with constant speed, the velocity vector changes every time a vehicle turns around to make its next pass. For the purpose of the simulation model the speed of all “cities” will be held constant throughout to counter the complexity added to the system due to the changing velocity vectors.

The goal of this simulation is to provide a routing solution to coordinate vehicles within an agricultural field. The scenario that this simulation has been developed around is the refilling of autonomous planters in a corn field by an autonomous tender. The solutions will be scored based upon the time taken for the tender to refill all of the planters as well as the area of the field that is able to be planted as a result of the refilling operation. Future uses for the simulation include the optimization of the number of planters and tenders working together in the same field.

Materials and Methods

A simulation model was created using Anaconda which is an open source distribution of the programming language Python 2.7. Other resources used included the Theoretical Crop Modeling Laboratory, Throckmorton 2206 at Kansas State University.

Process of Simulation

The process of generating a routing solution for the autonomous tender is detailed in the following order.

Field Initialization

The parameters of area, length, and width are determined and input into the simulation. For the current analysis a 40 acre field was chosen. This allowed the field to be large enough to ensure that the planters would need at least one refill to complete work in their zones, while being of a size that would be easy to scale and compare to field with the size of 80 or 160 acres. For this simulation the length and width were set to be equal to keep the simulation as simple as possible. The parameters for the field are summarized in Table 5.

Table 5 - Parameters and Variables for Field

Parameters		**Units**
Area	16.19 (40)	Hectares (Acres)
Length	402 (1320)	Meters (feet)
Width	402 (1320)	Meters (feet)
Row Width	.762 (2.5) (30)	Meters (feet) (inches)
Home Base Coordinates	(0,0)	(X,Y)
Variables		
Number of Planters	N	Input number
Number of Zones	# of planters	N (unit less)
Zone Width	Width/Number of Zones	Meters (feet)

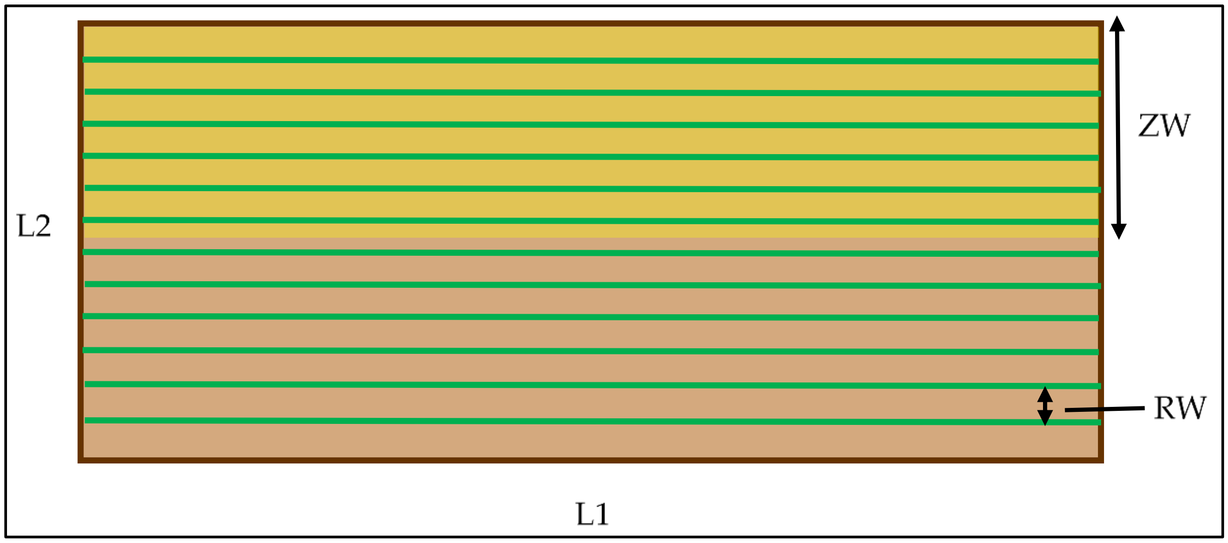
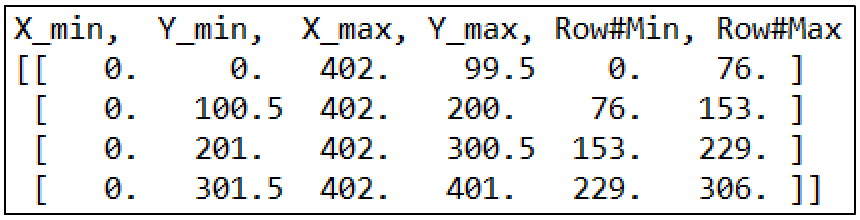


Figure 40 - Depiction of a Potential Field with Two Zones for Two Vehicles

The second step of the simulation is to input the number of planters that are desired for the simulation which will determines the number of zones within that field. The width of the field is divided by the number of zones and the coordinates for the zones are calculated for each planter shown in Figure 41.



```
X_min,  Y_min,  X_max, Y_max, Row#Min, Row#Max
[[   0.     0.   402.    99.5    0.     76. ]
 [   0.   100.5  402.   200.     76.   153. ]
 [   0.   201.   402.   300.5  153.   229. ]
 [   0.   301.5  402.   401.   229.   306. ]]
```

Figure 41 - Coordinates for Zones of Planters

Vehicle Initialization

A robot object class is created within the simulation so that both tenders and planters can use the same basic programming with minor variations. The major differences between the planter and the tender will be their respective velocities, hopper size, and unloading rates which are summarized in Table 6. While not mandatory it is beneficial for the tender, as the "salesman", to have a higher max velocity than the planter for it to be able to target and intercept for the refilling operation.

Table 6 - Parameters of Planter and Tender

Parameter	Autonomous Planter	Units	Autonomous Tender
Max Velocity	1.0	m/s	4.0
Max Hopper Volume	.07 (2.488)	m^3 (ft^3)	.21 (7.464)
Unloading Rate	(7.6971x10$^{\wedge -5}$)	(ft^3/sec)	(.04113)

Another parameter for the simulation is the planting population for the field. A higher planting population will affect the rate at which the hopper on the planter empties. Farmers who are trying to reach specific plant populations upon emergence must also take into consideration that some seeds may not germinate or emerge. Therefore an emergence factor can be used to get closer to the desired planting population. Other related factors include the bulk density of the seed that is being planted as well as the approximate number of seeds per pound. With these parameters known the distance that a planter can go before running out can be calculated. The parameters for planting corn in this simulation are listed in Table 7.

Table 7 - Parameters for Planting

Seeding Parameters	Corn	Units
Goal Population	28000	Plants/acre
Emergence Percentage	95	%
Seeding Population	29400	Seeds/acre
Seed Weight	1600	Seeds/lb.
Seed Bulk Density	56	lb. seed/bu.

Two other functions built into the robot class include *Locate* and *Hopper_Vol* sub-routines. A time point can be sent to these functions and the position or amount of seed remaining within the hopper will be returned (Figure 42).

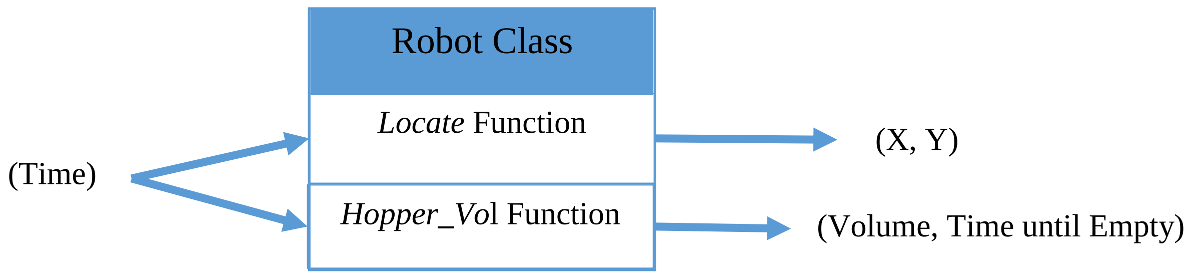


Figure 42 - Functions Built into Robot Class

A fleet of planters is created within the simulation by calling the robot class for N number of times and assembling them within an array, FLEET. The tender and its properties are also created from the robot class. With the vehicles created the next step is to determine interception routines.

Targeting for Interception

The targeting routine is titled *Row_Locate* and it is given the following information: begin interception time, a numerated planter from the fleet, a row number within that planter's work zone, and the tender. The targeting routine then calculates the time that it will take the tender to reach each end of the row number passed to the routine. The time values that it takes to reach the ends of the row are independently added to the begin interception time, both of which are then sent to the *Locate* function within the planter class to determine if the planter is in this row between the two calculated times. If the planter is not within the row that was input to the function, nothing happens and a score is returned at the end of the function.

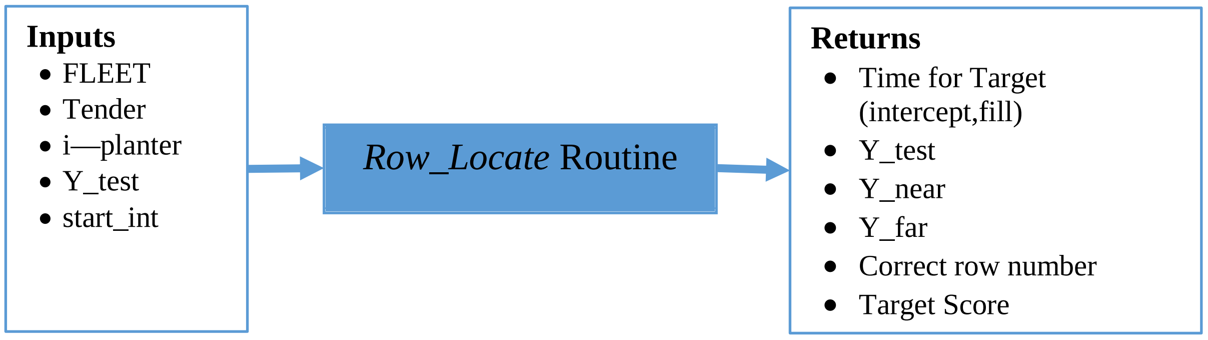


Figure 43 - Inputs and Returns of *Row_Locate* Routine

When the planter is in the row that was input to the function the interception point is calculated. The location of both the planter and tender is known at the beginning of the interception time that is input to the function, shown by the green and purple circles in Figure 44. Because both the planter and the tender are moving, the point of interception must be a point in the future of the location of the planter (basically the tender begins heading to a point that the planter will be with the shortest amount of time for interception).

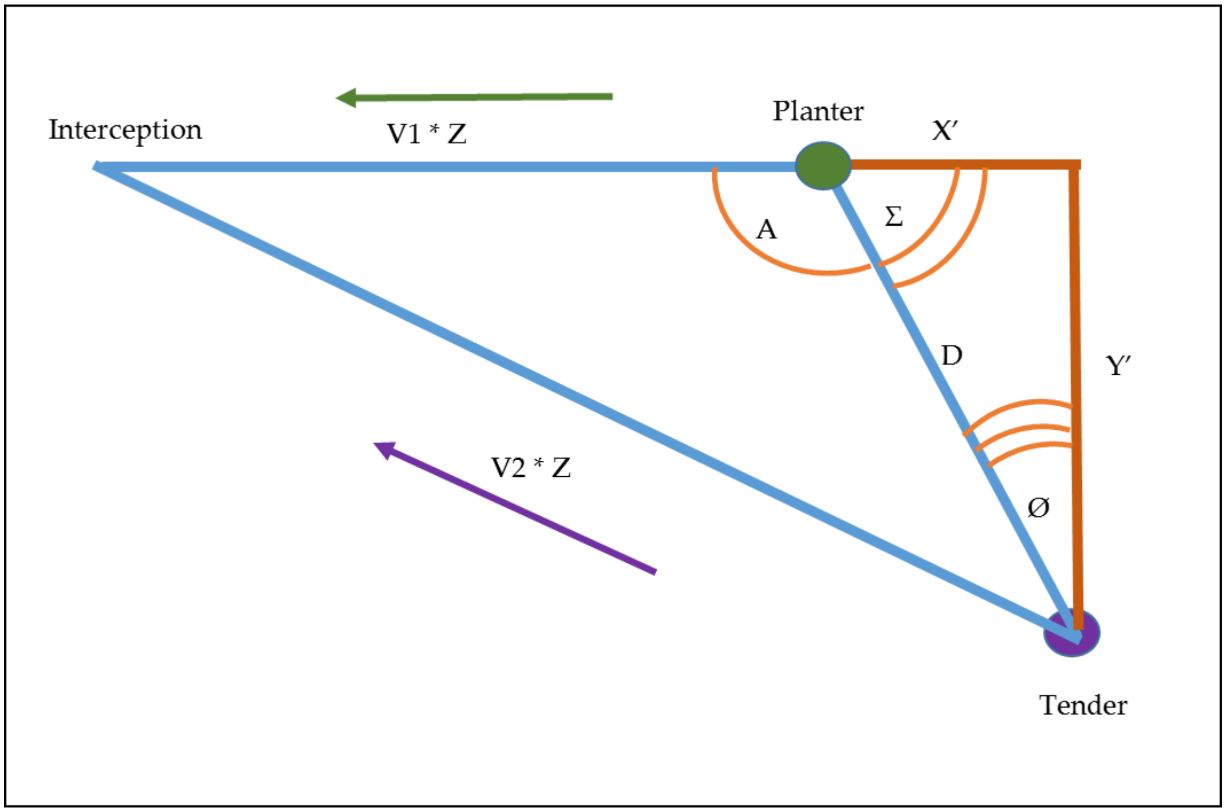


Figure 44 - Targeting Planter for Future Interception

The equation below is the derivation of trigonometric equations to calculate the time, **Z**, at which the interception will happen between the planter and the tender.

$$Z = \frac{V1 * D * \cos(A) \pm D\sqrt{V1^2 * \cos(A)^2 - V1^2 + V2^2}}{V1^2 - V2^2}$$

After intercepting the planter the tender will unload seed from its hopper into the hopper of the planter. For this simulation the transfer of seed occurs on the move, in a similar manner to how grain is often offloaded from a combine to a grain cart while still continuing to harvest grain. This was chosen to allow the planter to plant as efficiently as possibly with minimal downtime for the autonomous system. The volume of grain that has been planted will be replaced by the tender. The rate at which the tender can unload seed and the amount of seed that has been planted will determine how long it takes to refill the hopper on the planter. The total time to intercept and refill the planter is returned at the completion of the function.

Simulated Planting

The planting routine, titled *PLANT*, sends all of the planters into action. The *PLANT* function is given an array of test rows, POPULATION, for interception of all planters, in addition to information about the FLEET and the tender. Upon successful interception of all of the planters the total time required by the tender to intercept and fill the planters is returned to be compared to other randomly generated solutions. The rows with a zone that a planter is unable to plant due to not being filled are calculated and returned at the end of the function as well. These unplanted rows and the total time of tender operation are combined together into a score to evaluate different path solutions.

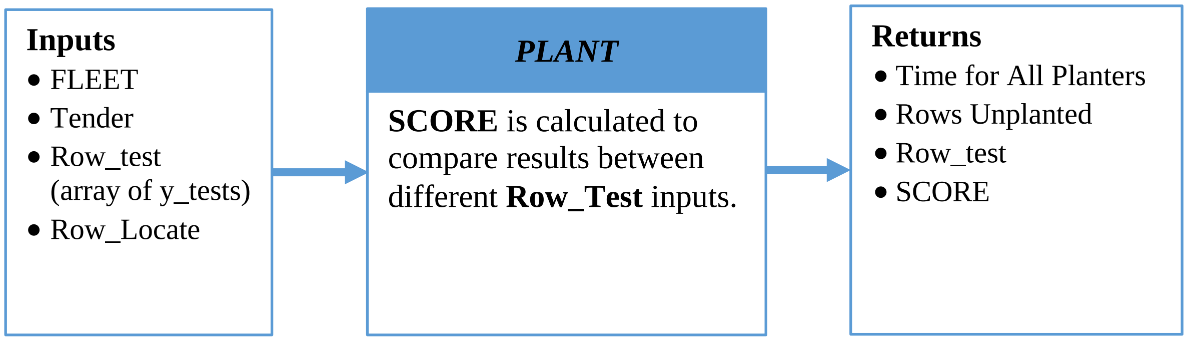

Inputs
• FLEET
• Tender
• Row_test (array of y_tests)
• Row_Locate
PLANT
SCORE is calculated to compare results between different Row_Test inputs.
Returns
• Time for All Planters
• Rows Unplanted
• Row_test
• SCORE

Figure 45 - PLANT Function Inputs and Returns for FLEET

Genetic Algorithm

A genetic algorithm is used to evaluate the initial population and then generate new guesses to find the optimal solution. The genetic algorithm (GA) function built into the simulation program inputs the PLANT function and a population with the number of values that can be set by the simulation operator (Figure 46). One value of the population has an array of *row_tests* that correspond to the number of planter within the simulation, N. The GA function is also given a number of iterations that are desired to be run to find the optimal solution. The outputs from the GA function include the best score from every iteration, the row guesses for the best score, and the population of the final iteration which includes the row guesses of the best score.

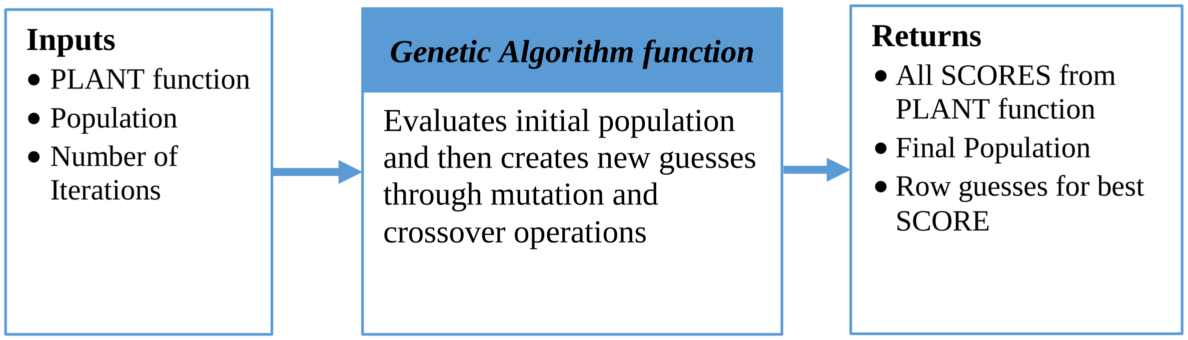

Inputs
PLANT function
Population
Number of Iterations
Genetic Algorithm function
Evaluates initial population and then creates new guesses through mutation and crossover operations
Returns
All SCORES from PLANT function
Final Population
Row guesses for best SCORE

Figure 46 - Input and Returns for the Genetic Algorithm Function

The inner workings of the GA function are illustrated in Figure 47. The initial population of guesses is evaluated and scored. Upon scoring the answers are reordered so that the best score (lowest value) is moved to the top of the list. The best score is then moved to a new population array. The remaining answers within the initial population are then put through a tournament style selection process. Two row number are generated at random and the population guess that corresponds with that row number is moved to the new population array (Figure 48). This process is repeated until the new population array is filled with the same number of guesses as the initial population array. After the tournament selection the population is exposed to a mutation operator which can randomly change parts of the answer, analogous to genes swapping or changing place. Following the mutation the GA then performs the crossover operation where two parents are crossed at some point along their gene (Figure 49). The process is then repeated for the number of iterations that are desired by the simulation operator.

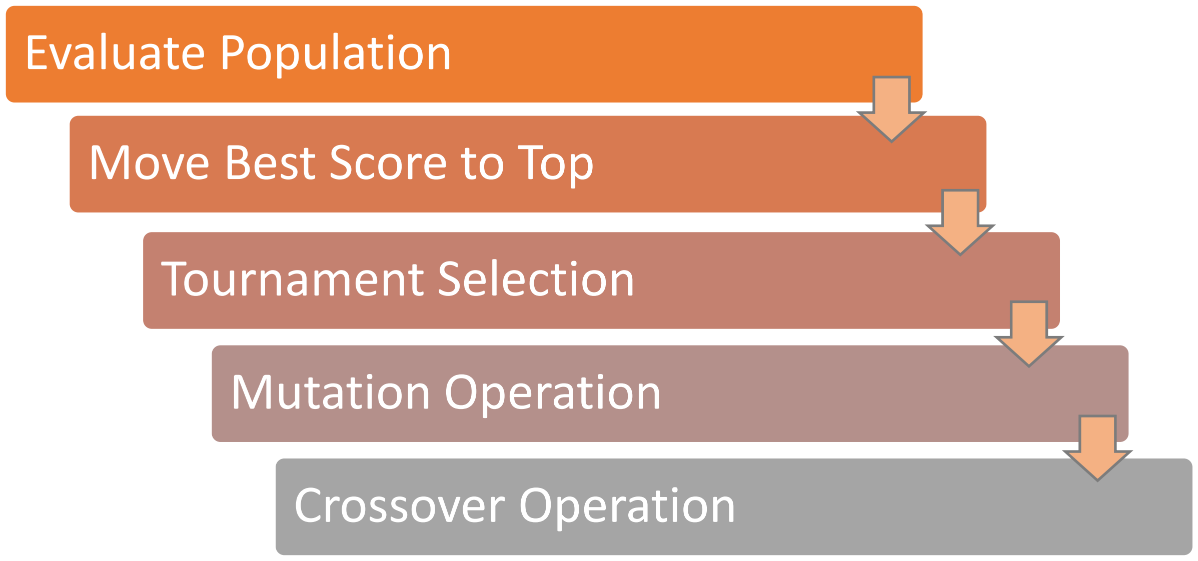
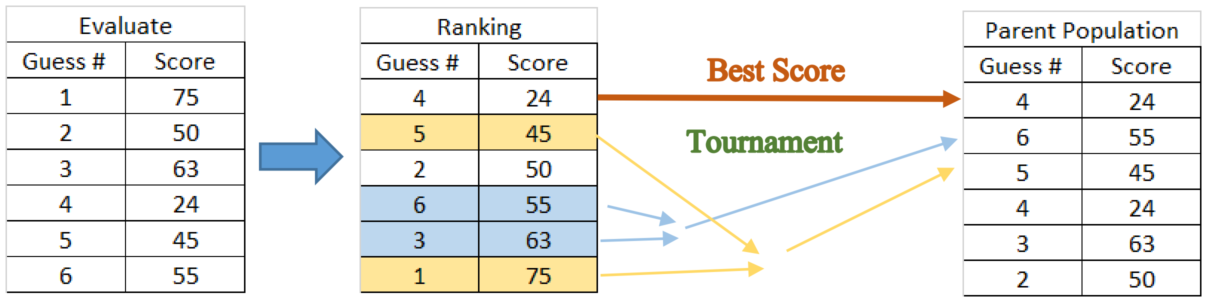


Figure 47 - Process for Genetic Algorithm

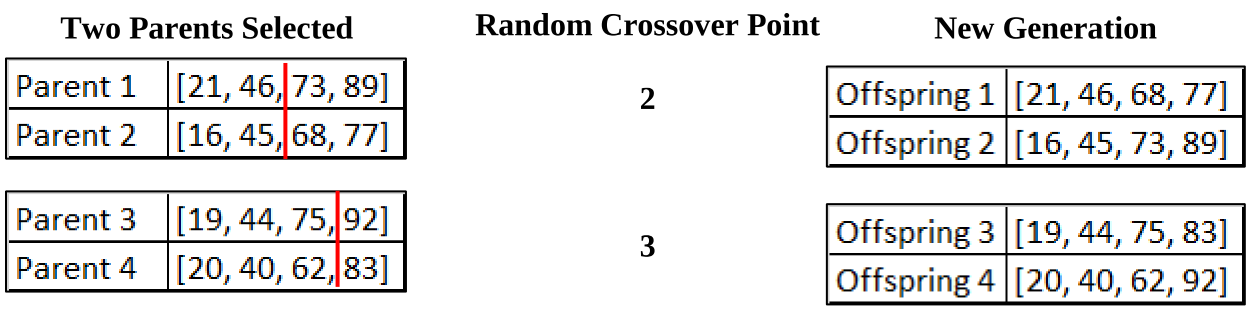


Evaluate	
Guess #	Score
1	75
2	50
3	63
4	24
5	45
6	55

Ranking	
Guess #	Score
4	24
5	45
2	50
6	55
3	63
1	75

Parent Population	
Guess #	Score
4	24
6	55
5	45
4	24
3	63
2	50

Figure 48 –Evaluation, Ranking, and Tournament Selection of Parent Population



Parent 1	[21, 46, 73, 89]
Parent 2	[16, 45, 68, 77]

Parent 3	[19, 44, 75, 92]
Parent 4	[20, 40, 62, 83]

Offspring 1	[21, 46, 68, 77]
Offspring 2	[16, 45, 73, 89]

Offspring 3	[19, 44, 75, 83]
Offspring 4	[20, 40, 62, 92]

Figure 49 - Example of Crossover Operation with Random Crossover Point

Performance Analysis of Simulations

A 40 acre field with four robotic planters was chosen for the simulation. The meta-parameters chosen for the first simulation include: a population size of 500 path combinations, a mutation rate of 3%, and a crossover rate of 60%. An initial simulation ran for 5,000 iterations produced a best score of 1081 after completing 4,000 generations. A second simulation was run for 50,000 iterations and reach a best score of 347 after 33,000 generations. Due to the length that it took the simulation to find an optimal solution, a tournament style selection was added to the genetic algorithm to aid in finding a heuristic solution in a shorter amount of time.

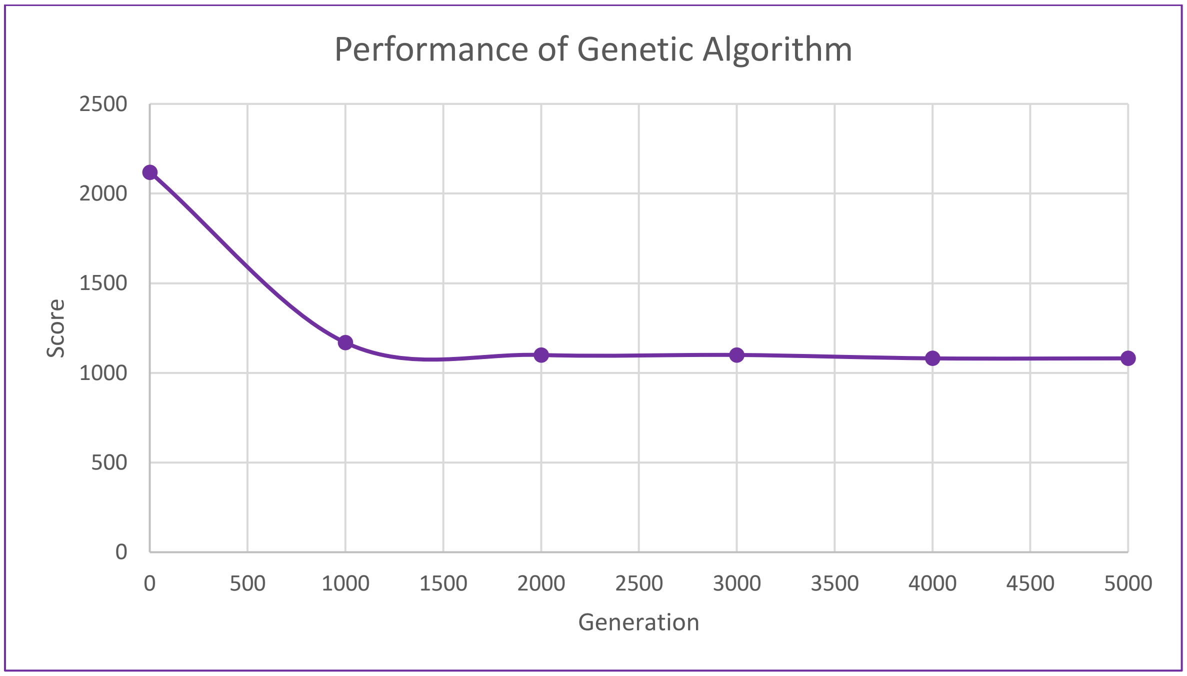


Figure 50 - Performance of Genetic Algorithm - 5000 Generations

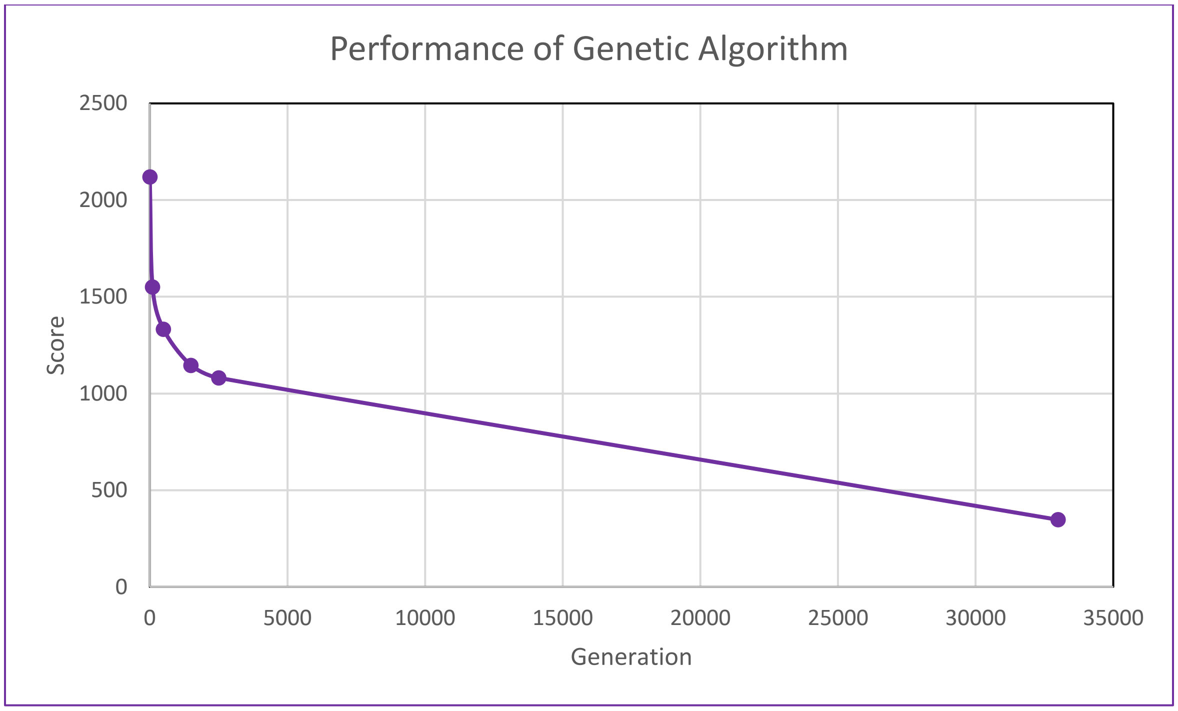


Figure 51 - Performance of Genetic Algorithm - 50,000 Generations

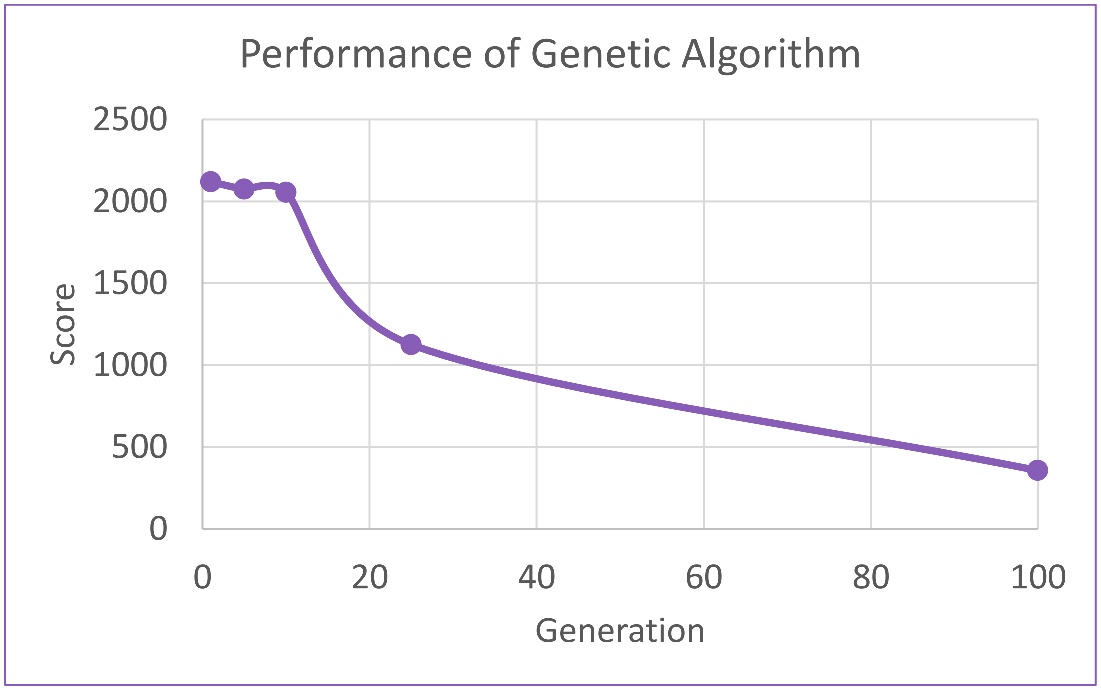


Figure 52 - Performance of Genetic Algorithm with Tournament Selection

Conclusion and Future Research

The simulation of the non-stationary traveling salesman problem proved to be effective in finding a routing solution for a tender traveling between four autonomous planters that required refilling. Addition of the tournament selection for parents reduced the number of iterations required to find the optimal solution. The simulation can be adapted for future research regarding the comparison of number of planters, number of tenders, and number of refill operations allowed. Direct comparisons between a field planted by autonomous planters returning to a home base for refill and a field planted by autonomous planters that are filled on the go will provide specific information as to how coordination of autonomous vehicles will be important for agriculture.

The simulation can also be modified to fit other agricultural applications such as a grain cart that is working with several combines within a field. The simulated coordination will be particularly useful for autonomous ground scouting vehicles in agriculture that are designed to have batteries exchanged by UAVs instead of returning to the home base.

Future research will focus on optimizing the coordination system and how the coordination operation can improve autonomous field operations in agriculture. The simulation tool will allow researchers to simulate field operations before robotic vehicles are ever sent out to the field, allowing them to determine the optimal equipment required for the field.

References

Bawden, O., Gall, D., Kulk, J., Perez, T., & Russell, R. (n.d.). *A Lightweight, Modular Robotic Vehicle for the Sustainable Intensification of Agriculture.*

Cavendar-Bares, K., Bares, J., & Bares, C. (2014, September 8). *ROWBOT in the Field*. Retrieved from www.rowbot.com: http://rowbot.com/blog-posts/2014/9/8/rowbot-in-the-field

Conesa-Munoz, J., Bengochea-Guevara, J. M., Andujar, D., & Ribeiro, A. (2016). Route Planning for Agricultural Tasks: A General Approach for Fleets of Autonomous Vehicles in Site-Specific Herbicide Applications. *Computers and Electronics in Agriculture 127*, 204-220.

Conesa-Munoz, J., Valente, J., del Cerro, J., Barrientos, A., & Riberio, A. (2016). A Multi-Robot Sense-Act Approach to Lead a Proper Acting in Environmental Incidents. *Sensors*.

Dvorak, J. S., Stone, M. L., & Self, K. P. (2016). Objct Detection for Agricultural and Construction Environments Using an Ultrasonic Sensor. *Journal of Agricultural Safety and Health*, 107-119.

Jiang, Q., Sarker, S., & Abbass, H. (2005). Tracking Moving Targets and the Non-Stationary Traveling Salesman Problem. *Complexity International Volume 11*, 171-179.

Kansas State University. (2007). *Corn Production Handbook.* Manhattan, KS.

Kinze Manufacturing Inc. (2014, December 14). *KINZE*. Retrieved from www.kinze.com: http://www.kinze.com/article.aspx?id=341

Kohanbash, D., Bergerman, M., Lewis, K. M., & Moorehead, S. J. (2012). A Safety Architecture for Autonomous Agricultural Vehicles. *American Society of Agricultural and Biological Engineers Annual International Meeting.* Dallas, TX: ASABE.

Larson, J., Okorn, B., Pastore, T., Hooper, D., & Edwards, J. (2014). Conter Tunnel Exploration, Mapping, and Localization with an Unmanned Ground Vehicle. *Unmanned Systems Technology XVI.* Baltimore, MD: SPEI Proceedings.

Li, N., Ma, S., Li, B., Wang, M., & Wang, Y. (2010). A Dynamic Shape-Shifting Method for a Transformable Tracked Robot. *Proceedings of the 2010 IEEE International Conference on Robotics and Biomimetics* (pp. 466-471). Tianjin, China: IEEE International.

Lui, L., Mei, T., Niu, R., Lui, Y., & Chu, S. (2016). RBF-Based Monocular Vision Navigation for Small Vehicles in Narrow Space Below Maize Canopy. *Applied Sciences*.

Mathew, N., Smith, S. L., & Waslander, S. L. (2015). Multirobot Rendezvous Planning for Recharging in Persistent Tasks. *IEEE Transactions on Robotics*, 128-142.

Pitla, S. K., Luck, S. D., & Shearer, S. A. (2010). Low-Cost Obstacle Detection Sensor Array for Unmanned Agricultural Vehicles. *ASABE Annual International Meeting.* Pittsburgh, Pennsylvania.

Potvin, J.-Y. (1996). Genetic Algorithms for the Traveling Salesman Problem. *Annals of Operations Research 63*, 339-370.

Rushworth, J. (2015, March 3). *Victron Energy*. Retrieved from https://www.victronenergy.com/blog/2015/03/30/batteries-lithium-ion-vs-agm/

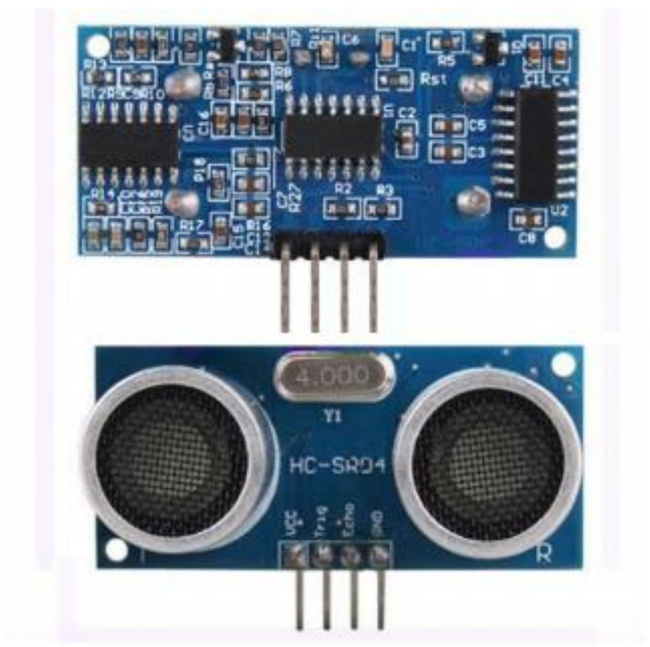
Santosh A. Hiremath, G. W. (2013). Laser Range FInder Model for Autonomous Navigation of a Robot in a Maize Field using a Particle Filter. *Computers and Electronics in Agriculture*.

Strickland, E. (2017, May 4). *Robotics*. Retrieved from IEEE Spectrum: http://spectrum.ieee.org/automaton/robotics/industrial-robots/demo-sarcos-snake-robot

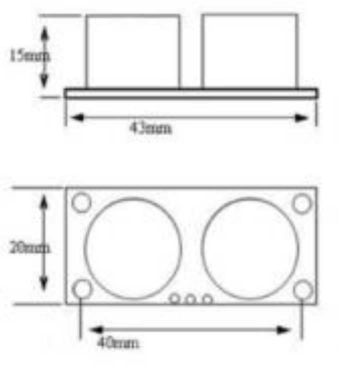
USDA NASS. (2012). *2012 Census of Agriculture*. USDA.

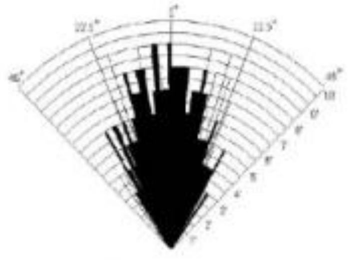
Appendix A - Ultrasonic Specification Sheet

3.0 PRODUCT LAYOUT



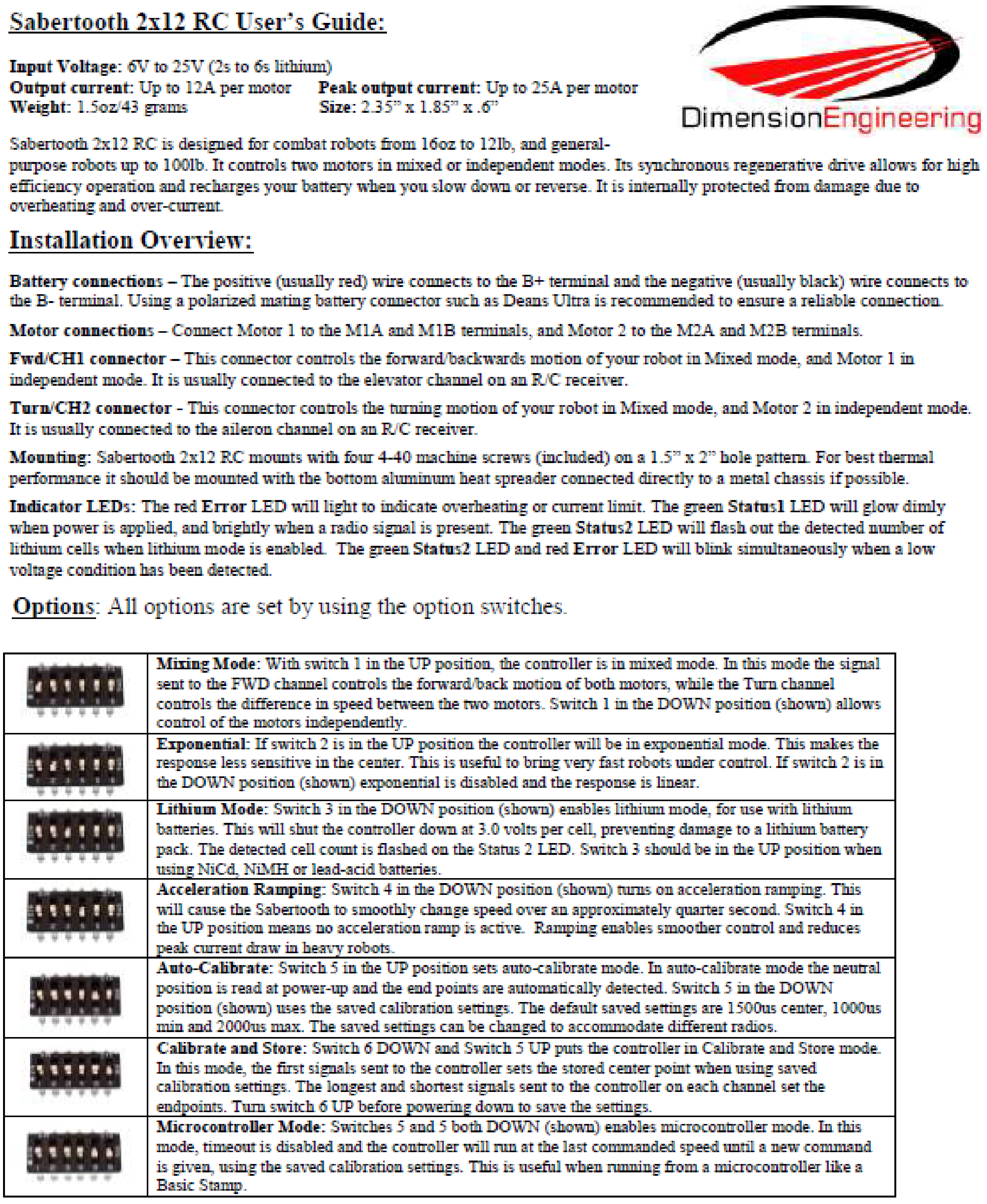
VCC = +5VDC
Trig = Trigger input of Sensor
Echo = Echo output of Sensor
GND = GND





Practical test of performance,
Best in 30 degree angle

Appendix B - Sabertooth Motor Controller



Sabertooth 2x12 RC User's Guide:

Input Voltage: 6V to 25V (2s to 6s lithium)
Output current: Up to 12A per motor **Peak output current:** Up to 25A per motor
Weight: 1.5oz/43 grams **Size:** 2.35" x 1.85" x .6"

DimensionEngineering

Sabertooth 2x12 RC is designed for combat robots from 16oz to 12lb, and general-purpose robots up to 100lb. It controls two motors in mixed or independent modes. Its synchronous regenerative drive allows for high efficiency operation and recharges your battery when you slow down or reverse. It is internally protected from damage due to overheating and over-current.

Installation Overview:

Battery connections – The positive (usually red) wire connects to the B+ terminal and the negative (usually black) wire connects to the B- terminal. Using a polarized mating battery connector such as Deans Ultra is recommended to ensure a reliable connection.

Motor connections – Connect Motor 1 to the M1A and M1B terminals, and Motor 2 to the M2A and M2B terminals.

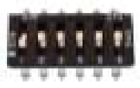
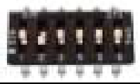
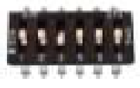
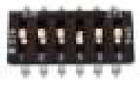
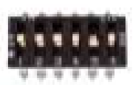
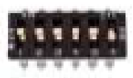
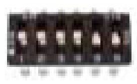
Fwd/CH1 connector – This connector controls the forward/backwards motion of your robot in Mixed mode, and Motor 1 in independent mode. It is usually connected to the elevator channel on an R/C receiver.

Turn/CH2 connector - This connector controls the turning motion of your robot in Mixed mode, and Motor 2 in independent mode. It is usually connected to the aileron channel on an R/C receiver.

Mounting: Sabertooth 2x12 RC mounts with four 4-40 machine screws (included) on a 1.5" x 2" hole pattern. For best thermal performance it should be mounted with the bottom aluminum heat spreader connected directly to a metal chassis if possible.

Indicator LEDs: The red **Error** LED will light to indicate overheating or current limit. The green **Status1** LED will glow dimly when power is applied, and brightly when a radio signal is present. The green **Status2** LED will flash out the detected number of lithium cells when lithium mode is enabled. The green **Status2** LED and red **Error** LED will blink simultaneously when a low voltage condition has been detected.

Options: All options are set by using the option switches.

	**Mixing Mode:** With switch 1 in the UP position, the controller is in mixed mode. In this mode the signal sent to the FWD channel controls the forward/back motion of both motors, while the Turn channel controls the difference in speed between the two motors. Switch 1 in the DOWN position (shown) allows control of the motors independently.
	**Exponential:** If switch 2 is in the UP position the controller will be in exponential mode. This makes the response less sensitive in the center. This is useful to bring very fast robots under control. If switch 2 is in the DOWN position (shown) exponential is disabled and the response is linear.
	**Lithium Mode:** Switch 3 in the DOWN position (shown) enables lithium mode, for use with lithium batteries. This will shut the controller down at 3.0 volts per cell, preventing damage to a lithium battery pack. The detected cell count is flashed on the Status 2 LED. Switch 3 should be in the UP position when using NiCd, NiMH or lead-acid batteries.
	**Acceleration Ramping:** Switch 4 in the DOWN position (shown) turns on acceleration ramping. This will cause the Sabertooth to smoothly change speed over an approximately quarter second. Switch 4 in the UP position means no acceleration ramp is active. Ramping enables smoother control and reduces peak current draw in heavy robots.
	**Auto-Calibrate:** Switch 5 in the UP position sets auto-calibrate mode. In auto-calibrate mode the neutral position is read at power-up and the end points are automatically detected. Switch 5 in the DOWN position (shown) uses the saved calibration settings. The default saved settings are 1500us center, 1000us min and 2000us max. The saved settings can be changed to accommodate different radios.
	**Calibrate and Store:** Switch 6 DOWN and Switch 5 UP puts the controller in Calibrate and Store mode. In this mode, the first signals sent to the controller sets the stored center point when using saved calibration settings. The longest and shortest signals sent to the controller on each channel set the endpoints. Turn switch 6 UP before powering down to save the settings.
	**Microcontroller Mode:** Switches 5 and 5 both DOWN (shown) enables microcontroller mode. In this mode, timeout is disabled and the controller will run at the last commanded speed until a new command is given, using the saved calibration settings. This is useful when running from a microcontroller like a Basic Stamp.

Appendix C - Motor Combination Specifications

Planetary Gearhead GP 42 C Ø42 mm, 3 - 15 Nm, Ceramic Version

Part number 203119

General information

Gearhead type	GP
Outer diameter	42 mm
Version	Ceramic version

Gearhead Data

Reduction	26 : 1
Absolute reduction	26/1
Max. motor shaft diameter	8 mm
Number of stages	2
Max. continuous torque	7.5 Nm
Max. intermittent torque	11.3 Nm
Direction of rotation, drive to output	=
Max. efficiency	81 %
Average backlash no load	0.8 °
Mass inertia	9.1 gcm²
Gearhead length (L1)	55.5 mm
Max. transmittable power (continuous)	240 W
Max. transmittable power (intermittent)	360 W

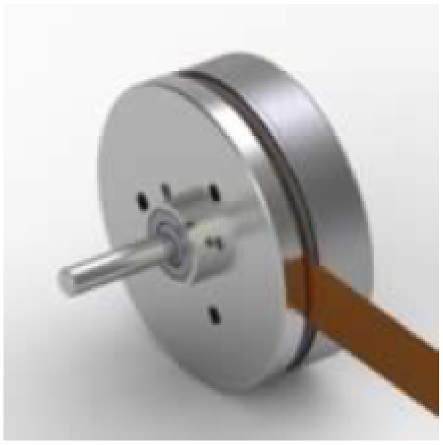
Technical Data

Radial play	max. 0.06 mm, 12 mm from flange
Axial play	max. 0.3 mm
Max. radial load	240 N, 12 mm from flange
Max. axial load (dynamic)	150 N
Max. force for press fits	300 N
Max. continuous input speed	8000 rpm
Max. intermittent input speed	8000 rpm
Recommended temperature range	-40...+100 °C
Number of autoclave cycles	0

Product

Weight	360 g

EC 45 flat Ø42.9 mm, brushless, 30 Watt, with Hall sensors
Part number 200142



Values at nominal voltage

Nominal voltage	12 V
No load speed	4360 rpm
No load current	163 mA
Nominal speed	2910 rpm
Nominal torque (max. continuous torque)	54.9 mNm
Nominal current (max. continuous current)	2.02 A
Stall torque	247 mNm
Stall current	9.69 A
Max. efficiency	76 %

Characteristics

Terminal resistance	1.24 Ω
Terminal inductance	0.56 mH
Torque constant	25.5 mNm/A
Speed constant	374 rpm/V
Speed / torque gradient	18.2 rpm/mNm
Mechanical time constant	17.6 ms
Rotor inertia	92.5 gcm²

Thermal data

Thermal resistance housing-ambient	6.73 K/W
Thermal resistance winding-housing	3.92 K/W
Thermal time constant winding	11.4 s
Thermal time constant motor	296 s
Ambient temperature	-40...+100 °C
Max. winding temperature	+125 °C

Mechanical data

Bearing type	ball bearings
Max. speed	10000 rpm
Axial play	0 - 0.14 mm
Max. axial load (dynamic)	4.8 N
Max. force for press fits (static)	53 N
(static, shaft supported)	1000 N
Max. radial load	18 N, 5 mm from flange

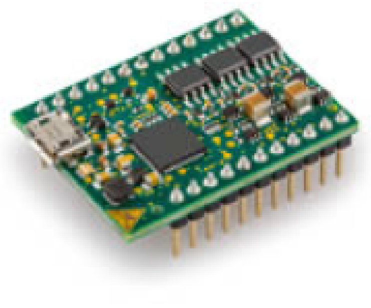
Other specifications

Number of pole pairs	8
Number of phases	3
Number of autoclave cycles	0

Product

Weight	75 g

ESCON Module 24/2, 4-Q servo controller for DC/EC motors, 2/6 A, 10-24 VDC
Part number 466023



Product	
Weight	7 g
Motor	
DC motors up to	48 W
EC motors up to	48 W
Sensor	
Without sensor (DC motors)	Yes
DC tacho	Yes
Digital incremental encoder (2-channel, single-ended)	Yes
Digital incremental encoder (2-channel, differential)	Yes
Digital incremental encoder (3-channel, differential)	Yes
Digital Hall sensors (EC Motors)	Yes
Operating modes	
Current controller	Yes
Speed controller (open loop)	Yes
Speed controller (closed loop)	Yes
Electrical data	
Operating voltage Vcc (min.)	10 V
Operating voltage Vcc (max.)	24 V
Max. output voltage (factor * Vcc)	0.98
Max. output current Imax	6 A
Max. time of peak output current Imax	4 s
Continuous output current Icont	2 A
PWM clock frequency of power stage	53.6 kHz
Sampling rate PI current controller	53.6 kHz
Sampling rate PI speed controller	5.36 kHz
Max. efficiency	92 %
Max. speed (DC)	150000 rpm
Max. speed (EC; 1 pole pair) block commutation	150000 rpm
Inputs	
Hall sensor signals	H1, H2, H3
Encoder signals	A, A\, B, B\
Max. encoder input frequency	1 MHz
Digital inputs	2
Functionality of digital inputs	Enable, enable CW, enable CCW, enable CW+CCW, enable + direction of rotation, stop, PWM set value, RC Servo set value, fixed set value
Analog inputs	2
Resolution, range, circuit	12-bit, -10…+10V, differential
Functionality of analog inputs	Set value, current limit, offset, speed ramp
Outputs	
Digital outputs	2
Functionality of digital outputs	ready, speed comparator, current comparator, commutation frequency
Analog outputs	2

	Resolution, range	12-bit, -4…+4V
	Functionality of analog outputs	current monitor,speed monitor, temperature, fixed value
Voltage outputs		
	Hall sensor supply voltage	+5 VDC, max. 30 mA
	Encoder supply voltage	+5 VDC, max. 70 mA
	Auxiliary output voltage	+5 VDC, max. 10 mA
Interface		
	USB 2.0 (full speed)	Yes
Display		
	Status indicator "Ready"	green LED
	Status indicator "Error"	red LED
Protective functions		
	Protective functions	current limit, overcurrent, excess temperature, undervoltage, overvoltage, voltage transients, short-circuits in the motor cable
Ambient conditions		
	Temperature – Operation (min.)	-30 °C
	Temperature – Operation (max.)	60 °C
	Temperature – Extended Range	+60…+80 °C, Derating: -0.1 A/°C
	Temperature – Storage (min.)	-40 °C
	Temperature – Storage (max.)	85 °C
	Humidity (non-condensing) (min.)	5 %
	Humidity (non-condensing) (max.)	90 %
Mechanical data		
	Weight	7 g
	Dimension (length)	35.6 mm
	Dimension (width)	26.7 mm
	Dimension (height)	12.7 mm
	Mounting	mountable on socket terminal strips pitch 2.54 mm
Software		
	Installation program	ESCON Setup
	Graphical User Interface	ESCON Studio
	Operating system	Windows 10, 8, 7, Windows XP SP3

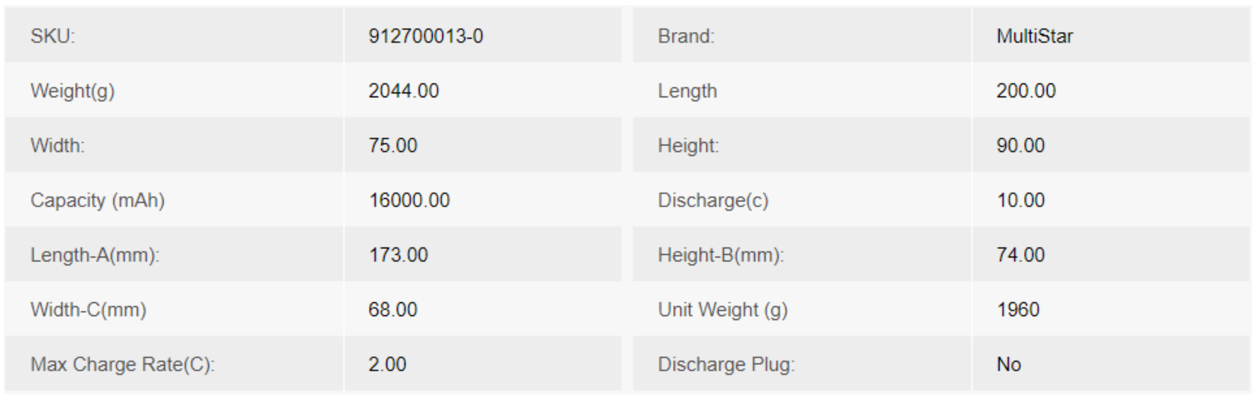
ESCON Module 24/2 Motherboard
Part number 486400

Product	
Weight	82 g

Adapter 8-pole flexprint connector to 8-pole screw terminal
Part number 425931

Product	
Weight	1 g

Appendix D - Lithium Ion Battery Specifications



SKU:	912700013-0	Brand:	MultiStar
Weight(g)	2044.00	Length	200.00
Width:	75.00	Height:	90.00
Capacity (mAh)	16000.00	Discharge(c)	10.00
Length-A(mm):	173.00	Height-B(mm):	74.00
Width-C(mm)	68.00	Unit Weight (g)	1960
Max Charge Rate(C):	2.00	Discharge Plug:	No

REF<https://hobbyking.com/en_us/multistar-high-capacity-6s-16000mah-multi-rotor-lipo-pack.html>

Appendix E - Component List

Component	**Quantity**	**Individual Cost**	**Total Cost**
Multistar 6S 22.2V 16000mAh Lithium Ion Battery	2	$120.00	$240.00
EC 45 Flat, Ø42.9mm, brushless, 30 Watt, with Hall Sensors	2	$73.00	$146.00
Planetary Gearhead GP 42 C Ø42mm, 3-15 Nm, Ceramic Version	2	$235.24	$470.48
ESCON Module 24/2, 4-Q servo controller for DC/EC motors, 2/6A, 10-24 VDC	2	$81.15	$162.30
ESCON Module 24/2 Motherboard	2	$78.80	$157.60
Adapter 8-pole Flexprint connector to 8-pole screw terminal	2	$20.60	$41.20
HC-SR04 Ultrasonic Sensor	5	$2.50	$12.50
myRIO 1900	1	$535.00	$535.00
L298 Dual H-Bridge DC Motor Controller	1	$6.89	$6.89
Lynxmotion 12V 10 RPM 416 oz-in 1/264 Brushed DC Gear Motor w/Encoder	2	$62.83	$125.66
HiTEC Minima 6T Radio Receiver	1	$25.99	$25.99
HiTEC Optic 5 2.4GHz Radio Controller	1	$80.90	$80.90
4x15” Aluminum Plate	1	$5.00	$5.00
3”x3” ball bearing turntable	2	$4.95	$9.90
8mm set screw hub	2	$9.40	$18.80
Lynxmotion Track Sprocket - 9 Tooth (Pair)	3	$9.95	$29.85
Lynxmotion Track - 3" Wide x 21 links ~23"	3	$40.32	$120.96
Lynxmotion HUB-13 Passive Idler Hub (Pair)	1	$14.95	$14.95
3D printed parts – Ultrasonic array, track rocker bracket, track housing, hubs	2	$50.00	$100.00
GRAND TOTAL			**$2,303.98**

Appendix F - Simulation Code (Python 2.7)

```
"""
Created on Mon May 08 20:10:34 2017

@author: aaschmitz
"""

import numpy as np              # Import array routines random
from numpy.random import rand,seed,shuffle      # Import pseudorandom nunber generator and
seed
import matplotlib.pyplot as plt # Import plotting library
import winsound
import math

                                # 40 Acres field (402m x 402m)
L = 402.0                       # length of field the long ways (meter)
L2 = 402.0                      # Length of field short ways (meter)
dt = 1.0                        # timestep of recorded information for graphs

Time=np.arange(0.,33000)
N=4                             # number of planters

def Coordinates(N):             #Function Creates Workzones

    coor    =[]                 # aranges the coordinates of each zone
    space   =[]                 # gives the spacing difference between zones
    IniPos     =[]              # gives the initial positions of each rover
    for i in xrange(N):
        coor += [[0.0, L2*(i)/N, L,
L2*(i+1)/N,math.floor((L2*(i)/N)/.762),math.floor((L2*(i+1)/N)/.762)]]  #
[x_start,y_start,x_end,y_end,row_num]
        space+= [[0,0,0,-1,0,0]]
    FieldCoord=np.array(space)+np.array(coor)       # coordinates of each zone for each planter
    IniPos=[[0.0,0.0]]                              # position of Home base - start position
    print "X_min,  Y_min,  X_max, Y_max, Row#Min, Row#Max"
    print FieldCoord
    print IniPos
    return IniPos,FieldCoord

Field = Coordinates(N)                              # combined information about field

class Robot(object):
    """ Define some basic properties and functions of rover vehicles"""
    def __init__(self,i, Hvol=2.488,V=1.0,W=2.0,Start=Field[0],Coord=Field[1],
Fill=False,Planter=True):

        # Constuctor for autonomous planter class

        self.start_x    = Start[0][0]          # home base x coordinate
        self.start_y    = Start[0][1]          # home base y coordinate
        self.x_min     =Coord[i][0]           # x lower boundary for workzone
        self.y_min     =Coord[i][1]           # y lower boundary for workzone
        self.x_max     =Coord[i][2]           # x upper boundary for workzone
        self.y_max     =Coord[i][3]           # y upper boundary for workzone

        self.RTST = 0                         # row traverse start time - when robot starts working in zone

        self.x   = self.start_x               # X position of Rover
        self.y   = self.start_y               # Y position of Rover
        self.row_width = .762                  # gives row width in meters (30 inches)
        self.rownum  = int(self.y/self.row_width)    # Row that the rover is in (y position/row width)

        self.row_initial = int(self.y_min/self.row_width) # first row of field
        self.Direction = 1                    # Gives degree heading of the vehicle (NESW)
        self.Velocity   =V                    # Initialize the Rover Velocity (m/s)
        self.T_Radius   =math.pi*self.row_width # Initialize turn cirucmfrence (m)
        self.T_Delay    =self.T_Radius*(1/self.Velocity) # Time to complete a turn

        self.TOLF        = 0.0                # time when hopper last filled
        self.HopMax       = 2.488              # ft^3 - volume of hopper
        self.Hopper      =Hvol                # Initialize the Hopper Volume (ft^3)
        self.seeding_rate = .000076971         # ft^3/s - seeding rate of planter
        self.T_Unload     = .041333            # ft^3/s - unload rate of the hopper

    def Locate(self,t,L):                     # locate the position based on time input
        tp = t-self.RTST                      # how long the rover has been running
        onerow = L + self.T_Radius            # length of one row
        rowtime = onerow/(self.Velocity)      # time that it take to complete one row
        self.rownum = tp/(rowtime) + self.row_initial     # how many passes completed
        rowfrac = self.rownum - int(self.rownum)         # gives the fraction of row completed
        self.rownum = int(self.rownum)                  # gives integer, for row number
        self.y = self.rownum*self.row_width        # gives Y position of planter
        if self.rownum % 2 ==0:
            self.Direction = 1                # gives the heading
            self.x = rowfrac*L           # gives the x postion
        else:
            self.Direction = -1               # gives the heading
            self.x = (1-rowfrac)*L       # gives the x position
        return self.x, self.rownum*self.row_width
    def Hopper_Vol(self,t,L):                  # determine hopper volume at time input
        tr = t-self.TOLF                       # time running since last refill
        time_to_empty = (self.HopMax)/(self.seeding_rate)      # time it takes to empty full hopper

        if self.Hopper <= 0.0:
            self.seeding_rate = 1.0
            empty_in = 0.0
            return [self.Hopper,empty_in]

        else: #self.x > 0 and self.x < L:
            #seeds_pound  = 1600.0                        # seeds per pound
            #seeds_acre   = 29400.0                       # plants planted per acre
            #pounds_acre  = seeds_acre/seeds_pound        # lbs of seed per acre
            #row_dist     = 5305.0                        # (m) length of row/acre
            #seed_metered = pounds_acre/row_dist          # lbs of seed per meter traveled
            #seeding_lb = self.Velocity*seed_metered      # lbs of seed per second
            #seeding = seeding_lb/45.0                    # volume of seed metered (45lbs/ft^3)
            self.Hopper  = self.HopMax*(1.0-(tr/time_to_empty))  # rate of changing hopper volume
            empty_in     = time_to_empty - tr
            #print self.Hopper
            return [self.Hopper,empty_in]
#       else:
#           self.seeding_rate = .000076971                # resets the seeding rate
#           return [self.Hopper,empty_in]

FLEET = []                         # initialize the array for a fleet
x_pos = []                         # initialize the fleet array to store x positions
y_pos = []                         # initialize the fleet array to store y positions
hop_vol = []                       # initialize the fleet array to store hopper volume

for i in xrange(N):                # loop to create N planters
    Planter=Robot(i)               # planters created from robot class
    FLEET+=[Planter]               # Planter added to the fleet

#    FX = []                       # planter x array
#    FY = []                       # planter y array
#    FH = []                       # planter Hopper array
#
#    for t in Time:                # time to run planting of field
#       FLEET[i].Locate(t,L)       # locate function gives position of planter at timepoints
#       FLEET[i].Hopper_Vol(t,L)   # function to give hopper volume at a particular time
#
#       xF = FLEET[i].x            # x position of planter at t
#       yF = FLEET[i].y            # y position of planter at t
#       hF=[FLEET[i].Hopper]       # hopper volume at t
```

```
#
#      FX += [xF]                  #data point added to array
#      FY += [yF]                  #data point added to array
#      FH += [hF]                  #data point added to array
#
#    x_pos +=[FX]                   # all of planter I position added to fleet array
#    y_pos +=[FY]                   # all of planter I position added to fleet array
#    hop_vol +=[FH]                  # all of planter I position added to fleet array

Tender=Robot(0, Hvol=7.464, V=4.0,) # create the tender, 6 Bu capacity

def Row_Locate(FLEET,i,Tender,L,y_test,start_int, VERB = False):

   '''find clock time at which it take planter to reach each end of row'''
   t1 = start_int + (math.sqrt((0.0-Tender.x)**2 + (y_test-Tender.y)**2)/Tender.Velocity)  # time
to get to close end of guess row
   t2 = start_int + (math.sqrt((L-Tender.x)**2 + (y_test-Tender.y)**2)/Tender.Velocity)    #time
to get to far end of guess row

   y_near = FLEET[i].Locate(t1,L)[1]          # identifies where planter will be at t1
   y_far = FLEET[i].Locate(t2,L)[1]           # identifies where planter will be at t2

   DF = start_int
   time_for_target = 0.0
   T_R = 0.0
   row_num = 0.0

   DNP = 100.0                                # Did Not Plant Factor for Score

   if y_near > y_test and y_far > y_test and y_near==y_far :      # if planter is above the guess
row then bad
       #print "BAD"
       row_num = y_near/.762
       DF = start_int
       time_for_target = 0.0
       T_R = 0.0
       target_score = DNP
       if VERB:
           print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
           print  [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]

   elif y_near > y_test and y_far > y_test and y_near != y_far:
       #print "BAD1 - planter changes rows"                          # if planter is above the guess row
then bad
       row_num = y_near/.762                   # case for y_near and y_far not equaling
       DF = start_int
```

```
        time_for_target = 0.0
        T_R = 0.0
        target_score = DNP
        if VERB:
            print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
            print [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]

    elif y_near == y_test and y_far > y_test and y_near != y_far:
        ##print "BAD2 - planter changes rows"                          # if planter is above the guess row
then bad
        row_num = y_near/.762                   # case for y_near and y_far not equaling
        DF = start_int
        time_for_target = 0.0
        T_R = 0.0
        target_score = DNP
        #print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
        #print [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]

    elif y_far == y_test and y_far > y_test and y_near != y_far:
        #print "BAD3 - planter changes rows"                          # if planter is above the guess row
then bad
        row_num = y_near/.762                   # case for y_near and y_far not equaling
        DF = start_int
        time_for_target = 0.0
        T_R = 0.0
        if VERB:
            print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
            print [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]
    elif y_near > y_test and y_far < y_test:      # if planter is above the guess row then bad
        #print "BAD3"
        row_num = y_far/.762
        DF = start_int
        time_for_target = 0.0
        T_R = 0.0
        target_score = DNP
        if VERB:
            print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
            print [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]

    elif y_near < y_test and y_far > y_test:      # if planter is above the guess row then bad
        #print "BAD4"
        row_num = y_near/.762
        DF = start_int
        time_for_target = 0.0
        T_R = 0.0
        target_score = DNP
        if VERB:
            print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
            print [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]

    elif y_near < y_test and y_far < y_test:    # if planter below the guess row, the tender can wait..
        #print "BETTER"
        row_num = y_near/.762
        DF = start_int
        time_for_target = 0.0
        T_R = 0.0
        target_score = DNP/2
        if VERB:
            print "[DoneFill, Time for Target, Time to Refill, y_test,y_near,y_far,row_num]"
            print [DF, time_for_target, T_R, y_test,y_near,y_far,row_num]

    elif y_near == y_test and y_far == y_test:  # if planter is in same row then good!!
        #print "GOOOO!!"
        target_score = 0.0
        row_num = y_near/.762
        H=FLEET[i].Direction                    # heading/directions of planter
        V1 = FLEET[i].Velocity                  # velocity of planter
        P_Vector = V1*H                         # vector of planter
        VT = Tender.Velocity                    # Tender velocity
        Px = FLEET[i].Locate(start_int,L)[0]    # x position of planter at beginning of intercept
        Py = FLEET[i].Locate(start_int,L)[1]    # y position of planter at beginning of intercept
        Tx = Tender.x                           # tenders current x position
        Ty = Tender.y                           # tenders current y position
        X_Diff = Px - Tx                        # x difference
        Y_Diff = Py - Ty                        # y difference
        D = math.sqrt(X_Diff**2 +Y_Diff**2)     # hypotenuse distance
        Theta = math.atan2(X_Diff,Y_Diff)       # angle of heading (radians)
        A = 90*(math.pi/180) + Theta            # angle to degrees

        Z1 = (P_Vector*D*math.cos(A) + D*math.sqrt((P_Vector**2)*((math.cos(A))**2)-
P_Vector**2+VT**2))/(P_Vector**2 - VT**2)
        Z2 = (P_Vector*D*math.cos(A) - D*math.sqrt((P_Vector**2)*(math.cos(A)**2)-
P_Vector**2+VT**2))/(P_Vector**2 - VT**2)

                #calculates the time to intercept the planter at earliest point
                #Z1 and Z2 are two answers of quadratic equation
        if VERB:
            print "distance apart   ", D, "m"
            print "angle           ", Theta, "radians"
            print "big angle       ", A, "radians"

            print "Tx > Px time to intercept   ", Z1, "seconds"     # tender above planter
            print "Tx < Px time to intercept    ", Z2, "seconds"    # tender below planter

        if Z1 >= 0:
            intercept_time = Z1
        if Z2 > 0:
            intercept_time = Z2

        time_of_int = start_int + intercept_time
                                            # function to refill hoppers

        Position = FLEET[i].Locate(time_of_int,L)    # gives position of planter at time of
interception

        Tender.x = Position[0]                    # sets tender position to intercept position   x
        Tender.y = Position[1]                    # sets tender position to intercept position   y
        Pmax = FLEET[i].HopMax                         # max hopper volume on the planter
        Pcur = FLEET[i].Hopper_Vol(time_of_int,L)[0]       # current hopper volume on the planter
        Psr = FLEET[i].seeding_rate                    # rate of seed leaving the hopper
        Tur = Tender.T_Unload                          # unload rate from the tender to the hopper 2
bushel/min
        T_R = ((Pmax - Pcur)/Tur) + (1.0/(1.0+Psr))     #  time it takes to refill the hopper

        if VERB:
            print "time to refill", T_R

        time_for_target = intercept_time + T_R

        #print "time for target", time_for_target

        DF = start_int + time_for_target
        FLEET[i].TOLF = DF                             # Time of last fill = time at done fill

        #print "clock time when done filling", DF
        #print " "

        Pos_df = FLEET[i].Locate(time_of_int,L)    # gives position of planter at time of done
filling
        Tender.x = Pos_df[0]                    # sets tender position to done filling position   x
        Tender.y = Pos_df[1]                    # sets tender position to done filling position  y

        return [DF,time_for_target,T_R,y_test,y_near,y_far,row_num,target_score]     # returns the
time that it took to fill up
                                                        # will need to include tender position when done as
well
    return [DF,time_for_target,T_R, y_test,y_near,y_far,row_num,target_score]
```

```
print "   "
row_test = [20,151,283,415]            # this works - score 2119
#row_test = [30,161,293,425]           # this works - score 1527
#row_test = [40,171,303,435]           # this works - score 947
#row_test = [50,181,313,445]            # this works - score 372
#row_test = [53,184,316,448]            # this works - score 356

def PLANT(row_test,FLEET,Tender,L,Row_Locate,Verbose = False):

    TenderTrack = []
    row_target_score = []
    first_row = row_test[0]
    start_int = first_row*((L+math.pi*.762)/(FLEET[0].Velocity))
    end_fill = [start_int]
    if Verbose:
        print "Calculated Start Tracking Time   " ,start_int
        print " "

    for i in xrange(N):
        y_test = row_test[i]*.762

        Test_Planter = i
        Trial = Row_Locate(FLEET,Test_Planter,Tender,L,y_test,start_int,VERB = Verbose)

        start_int = math.ceil((Trial[0]))                  #gives new start time, rounded up
        if Verbose:
            print "clock time done   ",Trial[0]
            print i,  "--", start_int
            print "new Start int     ",start_int
            print " "
        end_fill+=[start_int]
        TenderTrack += [Trial]
        row_target_score +=[Trial[-1]]

    time_for_all = end_fill[-1]-end_fill[0]                # time at end of last fill, minus start time
    if Verbose:
        print " "
        print "row target scores   ", row_target_score
        print " "
        print "Time Running in Field    ", time_for_all         # time spent from begining to end
        print " "

    #print TenderTrack
    """Funtion to find when/where planters run out"""
    """What percentage of their job are the able to do?"""
    status = []
    empty_position = []
    for i in xrange(N):
        time = 30000
        HopperStatus = FLEET[i].Hopper_Vol(time,L)
        position = FLEET[i].Locate((time+HopperStatus[-1]),L)
        status += [HopperStatus]
        empty_position += [position]

    field_remaining =[]
    scaling = []
    for i in xrange(N):
        PG = Field[-1][i][3]            #planters goal rows
        PE = empty_position[i][1]       #planters end rows
        rows_remaining = PG-PE          # amount of rows remaining per each planter
        field_remaining += [rows_remaining]
        scaling += [N-i]
    scale = np.array(scaling)
    field = np.array(field_remaining)     # total amount of rows remaining in the field
    field[np.where(field<0)] = 0
    field = np.sum(field)
    total_row_score = np.sum((row_target_score)*(scale)**2)
    row_factor = 20.0
    SCORE = time_for_all + field*row_factor + total_row_score    # time to plant all + factor of
unplanted field
    if Verbose:
        print "array of rows remaining", field_remaining
        print " "
        print "Unplanted Rows in Field Remaining    ", field*row_factor
        print "Time for All                        ", time_for_all
        print "Planters Not Reached                ", total_row_score
        print "Scaling for Rows                    ", scale
        print "Score                               ", SCORE
        print " "
    return [time_for_all, field, row_test, SCORE]     # time from begining of interception to last
fill,  rows left
#Results  = PLANT(row_test,FLEET,Tender,L,Row_Locate,Verbose = True)
row_test = []
for i in xrange(N):
    row_test += [Field[-1][i][4]+1]  #print the row number of the lower bound of the zone
print "Row Test",  row_test
print " "
seed(seed=0)
sets = 500          # sets the number of intial conditions
                    # Create 100 sets of initial conditions.
                    # The row guesses will be random numbers between upper and lower bounds
```

```
POPULATION=np.array(row_test)+np.round(np.random.rand(sets,N)*np.array([(row_test[1]-
row_test[0]-1)]),0)
#print POPULATION
#print POPULATION[99]
POP = POPULATION.tolist()
print "First Population Guess", POP[0]
print " "
POP[0] = [20,151,283,415]
#print POP[0]
def GA(PLANT, args=(POP,N,FLEET,Tender,L,Field,Robot,sets), maxiter = 20, verbose =
False):
    best_guess = []
    best_score = []
    Trap = False
    for M in xrange(maxiter):
        if (M/1000.0).is_integer() and M > 1:
            print "Best Score", best_score[-1], "Iteration", M
            #winsound.Beep(1000,200)

        #print INITIAL_GUESS
        Score = []
        '''set up the conditions'''
        for Row in POP:
            FLEET = []

            for i in xrange(N):                   # loop to create N planters
                Planter=Robot(i)                  # planters created from robot class
                FLEET+=[Planter]                  # Planter added to the fleet
            Tender=Robot(0, Hvol=7.464, V=4.0,) # create the tender, 6 Bu capacity
            '''evaluate the guesses'''
            Test = PLANT(Row,FLEET,Tender,L,Row_Locate,Verbose=False)
            Score += [Test[-1]]
        if M == 0:
            print "First Score", Score[0]
        if verbose:
            print " "
            print "Score in Position 1", Score[0]
            print "Best Score", np.min(Score)

        best_guess += [POP[0]]
        best_score += [Score[0]]

        '''Trapping Operation'''
        if  M > 2:
            Trap = True
```

```
if Trap == True and best_score[-1] > best_score[-2]:
    verbose = True

'''Move Best to Top'''
Best_ind = np.argmin(Score)
if verbose:
    print " Index of Best Score", Best_ind
a,b = 0,Best_ind
POP[a],POP[b]=POP[b],POP[a]
#print IG

'''Tournament Selection'''

POP2 = np.array(POP)
Score2 = np.array(Score)
inds = Score2.argsort()
POP3 = POP2[inds]

if verbose:
    print "Score   ",  Score
    print " "
    print "Population  ", POP
    print " "
    print "Population 3", POP3

#POP[0]=POP3[0]
for i in xrange(1, (sets)):
    R1 = np.random.randint(0,sets)
    R2 = np.random.randint(0,sets)

    if R1 < R2:
        POP[i] = POP3[R1]
    else:
        POP[i] = POP3[R2]

'''Mutation Operation'''

lower_bound = []
for i in xrange(N):
    lower_bound += [Field[-1][i][4]]  #print the row number of the lower bound of the zone

L0 = len(POP)
R = np.random.rand(L0)
#print 'R', R
num_mutations = 0
```

```
        for i in xrange(1, (L0-1)):
            if R[i] < .03 :                 # 3% chance of mutation
                G = POP[i]
                L1 = len(G)
                z = np.random.randint(0,L1)
                #print G
                G[z] = lower_bound[z]+np.round(np.random.rand()*(lower_bound[1]-
lower_bound[0]-1))
                            # this is where row changes within a guess
                #print G
                num_mutations = num_mutations+1
        if verbose:
            print "Number of Mutations", num_mutations, " in Iteration", M

        '''Crossover Operation'''

        A = (range(1,sets))
        shuffle(A)
        B = (range(1,sets))
        shuffle(B)
        S = [0] + A             # makes list for first choice
        #print S
        T = [0] + B             # makes list for second choice
        #print T
        J = [0] + np.random.randint(0,N,sets-1).tolist()    # makes list for random slice point
        #print J
        for i in xrange(1,int(sets*.60)):
            POP[S[i]][J[i]:],POP[T[i]][J[i]:] = POP[T[i]][J[i]:],POP[S[i]][J[i]:]
        #print " "
        #print IG
        #POP = POP
        if verbose:
            print " M equals iteration #",  M
            print " "
    #print " Best Scores"
    #print best_guess
    print " Best Scores" , best_score
    return [Score,POP,POP[0]]
Test = GA(PLANT, args=(POP,N,FLEET,Tender,L,Field,sets,Row_Locate), maxiter = 100,
verbose = False )
GO = True
if GO:
    #print "Number of Iterations", maxiter
    print "Test Row"  ,  Test[-1]
    #print " Problem POP",  Test[-2]
    print " Score" , Test[0][0]
print " "
row_test_best = Test[-1
BEST = PLANT(row_test_best,FLEET,Tender,L,Row_Locate,Verbose = True)
hop = Tender.Hopper_Vol(32000, L)
print hop
print " "
print " Best score", BEST[-1]
print " "
print " Best row guess", BEST[-2]
#best_guess = Best[-1]
#return best_guess
#print best_guess
```