Final Report

Project Team

Isaac Hershenson [TL], Kylee Graper, Carlos Sanchez, Nandini Garg (Sr-S), Wava Chan (Jr-S), Isabelle Kemp (Jr-S), Henry Connell (Jr-F), Ahlyssa Santillana (Jr-F)

Project Advisor: Professor Victor Shia
Project Liaisons: Mike Schmidt, Dylan Stokosa, and Jacob Huesman

May 2024

1. Introduction

This section will introduce the project, its goals, its deliverables, and give a brief overview of the team's accomplishments this year.

1.1 Abstract

The goal of this project was to program a Doosan Bobcat lawn mower to autonomously mow a field within a given boundary. The team succeeded in this goal by delivering a lawn mower that was able to do so. During the site visit, the team demonstrated a solution that could take in latitude and longitude points, create a given boundary from these points, calculate a path that had optimal overlap and long, straight paths, and then drive that path. The team also developed and tuned a proportional controller to keep the lawn mower on a controlled path.

Throughout the year, the team used 3 different hardware and software architecture setups as the development team at Doosan Bobcat integrated on the mower's design in parallel with the clinic team. The final solution can be equipped quickly and easily on any Doosan Bobcat lawn mower with the right developmental hardware. This let the team demonstrate the final solution during the site visit without having to ship the mower all the way to the innovation center.

1.2 Problem Statement

Mowing of large commercial areas has been identified as a potential opportunity to automate Bobcat mowers. These large areas often have defined boundaries and multiple undefined obstacles. Defining an efficient plan to mow these commercial areas becomes one challenge and then avoiding obstacles is an additional challenge.

The HMC team will work to develop a method to autonomously mow a given area. This method will need to reduce overlap to conserve time and resources, leave a clean stripped appearance, and handle variations in terrain, such as inclines or obstacles. This will be done using a provided mower and sensor suite. The developed software will be designed to handle this and modify paths based on potential dynamic obstacles. The mower will not need to classify or identify objects, just identify that they exist. This project will build on the work of past projects and development from both previous HMC clinic teams and the Doosan Bobcat development team.

1.3 Deliverables

Fall:
Gained a general understanding of the mower and the software and hardware set up of last year's team, developed mathematical understanding of state estimation and control equations necessary to application. Received mower and familiarized the team with its usage. Developed new software packages to keep up with system architecture.

1. Developed an algorithm to draw a clean mowing pattern within a defined boundary area.
2. Used the provided radar units to collect SLAM data on the defined boundary area.
3. Traversed through mowing pattern in defined boundary.
4. Compile results in midyear report.

Spring:
Continued development of mower software architecture and equations. Tuned the control system and made a simulator. Used path planner to test mower control and navigation.

5. Mapped the mowing area with potential static obstacles.
6. Navigated autonomous mowing path.
7. Optimized system to reduce overlap while still providing clean striped appearance.
8. Compiled results in the end of year report, poster, and final report to explain project progress and solution to be shared with Doosan Bobcat liaisons and other Harvey Mudd College Clinic presentations.

2. Project Context

2.1 History of Past Clinic Projects

Doosan Bobcat's leading technology has historically focused on mechanical engineering and manufacturing high-quality machines. The company has a long history of developing work machines for construction and landscaping. Research and development in intelligent systems like autonomy is a relatively new space for the company.

Doosan Bobcat has been sponsoring Clinic Projects since the 2017-2018 academic year, making this year's team the sixth at Harvey Mudd College. However, this specific lawn mower project is only in its third year. The 2022-2023 clinic team was working on the same problem and had the goal of delivering the same set of deliverables. The 2022-2023 clinic team succeeded in creating a wheel controller and path planning algorithm. The wheel controller algorithm directed the mower to smoothly reach a waypoint and execute turns. The team also developed a path planner to process any 2D user-defined mower boundary and internal obstacles into a mow pattern with tunable parameters like overlap ratio and mow direction. These tools were built and iterated upon by the current clinic team, as they were never tested on the mower itself.

2.2 Fall Updates to the Clinic Project

There were many changes that occurred between the fall semester and last year in terms of the clinic project, mostly regarding the mower communication plan. The path to communication with the mower was much more complicated last year, and was one of the limitations of the project at that time. This process can be seen visually in Figure 1 below. Last year's workflow for mower communication relied on the CAN bus to enable control signals and sensor data transfer. CAN messages allowed the control of functions (starting/stopping the engine, setting wheel velocities, adjusting throttle, etc) through 10 different signals. On the hardware side, a Jetson TX2 module outfitted with CAN transceivers interfaced with the mower to receive wheel velocity and GPS/heading data over one CAN bus, while sending control commands over a separate bus. This prevented bus overloading. An Arduino with a CAN bus shield served as an alternative messaging system, running a separate program for reliable CAN transmit/receive. Python scripts were written to convert CAN messages into usable decimal values for the algorithms. A main controller program leveraged Python's multiprocessing library to coordinate and schedule the execution of key submodules like path tracking and mapping, which will be carried over to our new communication workflow. It enabled concurrent execution via separate processes and provided data sharing between submodules using shared memory.

The fall mower communication process, seen in Figure 2, simplifies many of the complex processes mentioned above due to the new hardware (V2.5) given to us by Jacob Huesman. The Python scripts (path planning, mapping, etc) were put into a Docker container that, along with the V2.5 scripts given to us by Jacob, were put onto the V2.5 via wifi or ethernet through packaging and unpacking these scripts. Once on the V2.5 controller, these two new packages will be able to communicate with one another through ROS. We will still be receiving CAN messages, but the script given to us by Jacob, will automatically convert these messages to easier to use ROS messages. This will make it so that instead of sending raw CAN messages, we will simply be sending commands about right and left wheel velocity as well as the throttle of the mower.

Additionally, we received a new mower in the fall that was different from the mower from past years. This mower had all of the same specs as last year except now the mower is equipped with 5 radars instead of 1. Thus, the main difference is within the radar field of view that is now possible with the 5 radars. The radars are placed strategically so that the field of view is 360 degrees up to 10 meters. After 10 meters, there is a drop off in the field of view towards the back of the mower. This setup was used for the fall semester, but before the start of the spring semester all radars were removed from the mower, since they were not sufficiently high quality, which resulted in a slight rescoping of the project to exclude dynamic obstacle detection from this year's goals.

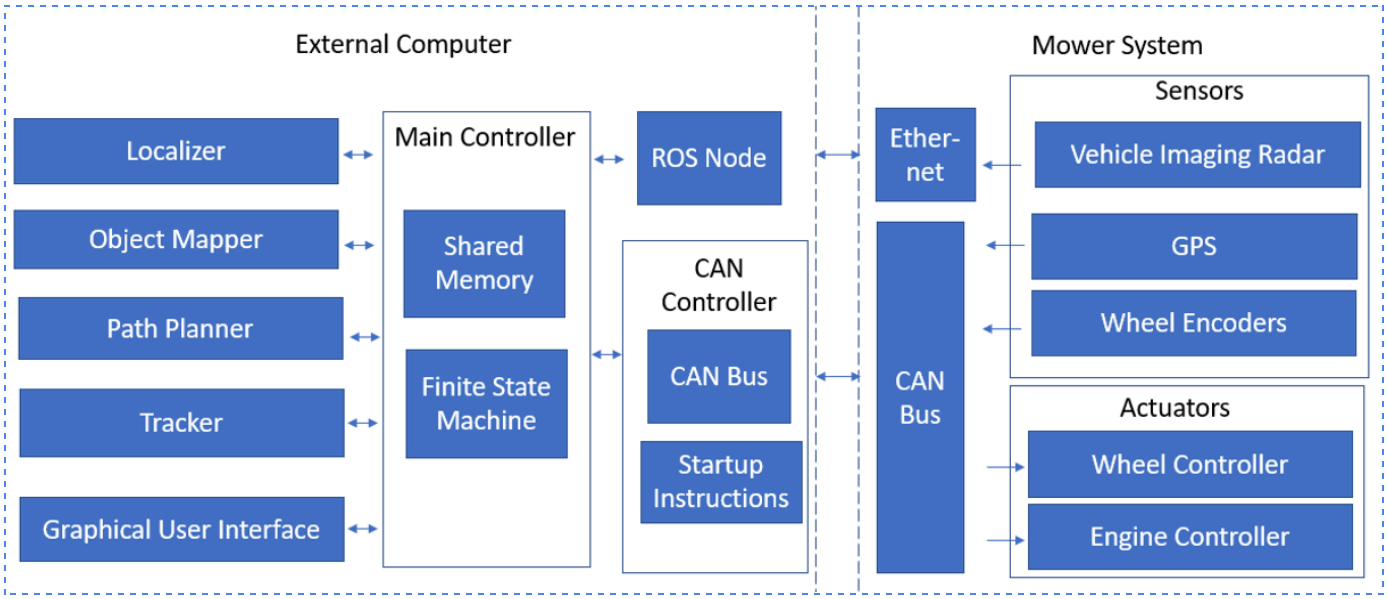

Figure 1. Last year's (2022-2023) Architecture/Workflow for Mower Communication

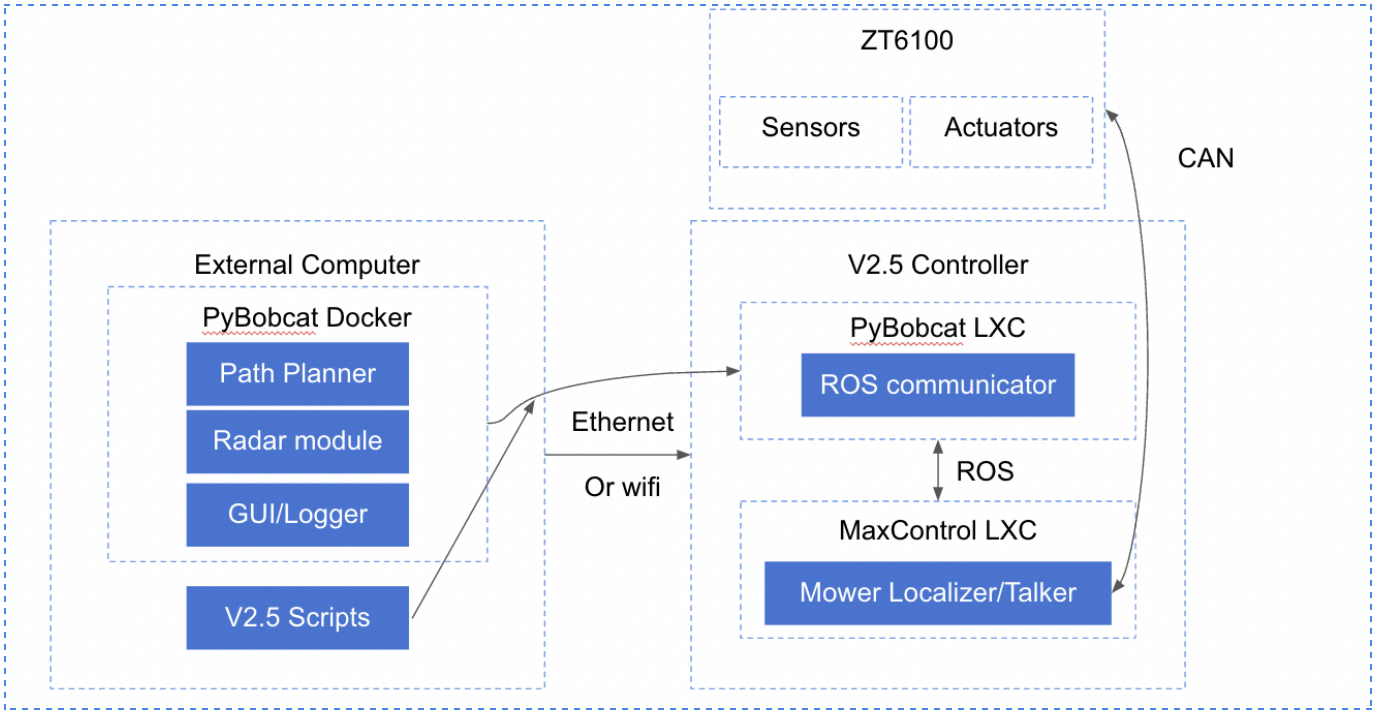

Figure 2. First Architecture/Workflow for Mower Communication in 2023/2024 Clinic

2.3 Spring Updates to the Clinic Project

There were several updates to the project between the fall and spring semesters of this year's clinic project. In the fall, there was focus on utilizing radars for obstacle detection, and the project scoping included stretch goals of incorporating dynamic obstacle detection in the solution for this year. After the radars were removed at the end of the fall semester, the focus pivoted to mowing only a bounded area without dynamic obstacles, with a stretch goal of incorporating fixed obstacles into the path planning process. This is because, as a result of the removal of the radars, there is no longer any way for the mower to sense what is happening around it in real time. Another large change happened with regard to the process of communicating with the mower. Throughout the process of testing, several changes were made to the communication architecture, which are detailed in section 4. These changes ultimately resulted in the removal of the V2.5 controller from the mower.

2.4 High Level Algorithm

1. **Accept user input defining mowing specifications**
    a. What area should be mowed? What areas should be avoided? How should the mowed stripes look?
2. **Localize the robot**. Where is the robot facing, and where is it in 2D space?
    a. Collect GPS data from both antennas, then utilize the RTK correction service to reduce error.
    b. Convert to local coordinates for quick tuning and validation.
3. **Map the surroundings of the robot**. What obstacles are around?
    a. Radar data reports list of objects, and their respective position and velocity in space
    b. Correct the map with any new radar data. Identify what space is occupied, what objects are static vs. moving.
    c. If the updated map is changed, make a new robot path (see step 3) to mow across unoccupied space.
    d. Algorithms: inverse range sensor model, binary occupancy grid
4. **Generate a robot path plan.** Where should the robot go?
    a. Combines user input, map, and robot position to generate a list of waypoints.
    b. Algorithms: trapezoidal cellular decomposition, cell merging, naive nearest neighbor, Dubins curves, A* or RRT*
5. **Follow the path plan.**
    a. How fast the mower wheels must turn to move to its next waypoint in the path
    b. Send the wheel velocities to the mower interface to execute the command.
    c. Algorithms: proportional control for a differential drive robot

3. System Specifications

This section describes the exact lawnmower system used and communication specifications.

3.1 Next Generation Bobcat ZT6100 Mower

For the testing and development of the autonomous system at Harvey Mudd College, a ZT6100 Mower was utilized. The commercially available version of this model is 1500lbs and can move up to speeds of 19mph.

The version at Harvey Mudd contains key differences when compared to the commercial model. Currently, the mower also consists of two GPS antennas, a localization device, a wired connection port, and an autonomous mode switch. The wired connection allows commands to be sent to the actuators, which control the wheels and the engine. The emergency stop ensures safe operation, and the autonomous mode switch allows the user to start the software from the mower and exit the manual control mode.

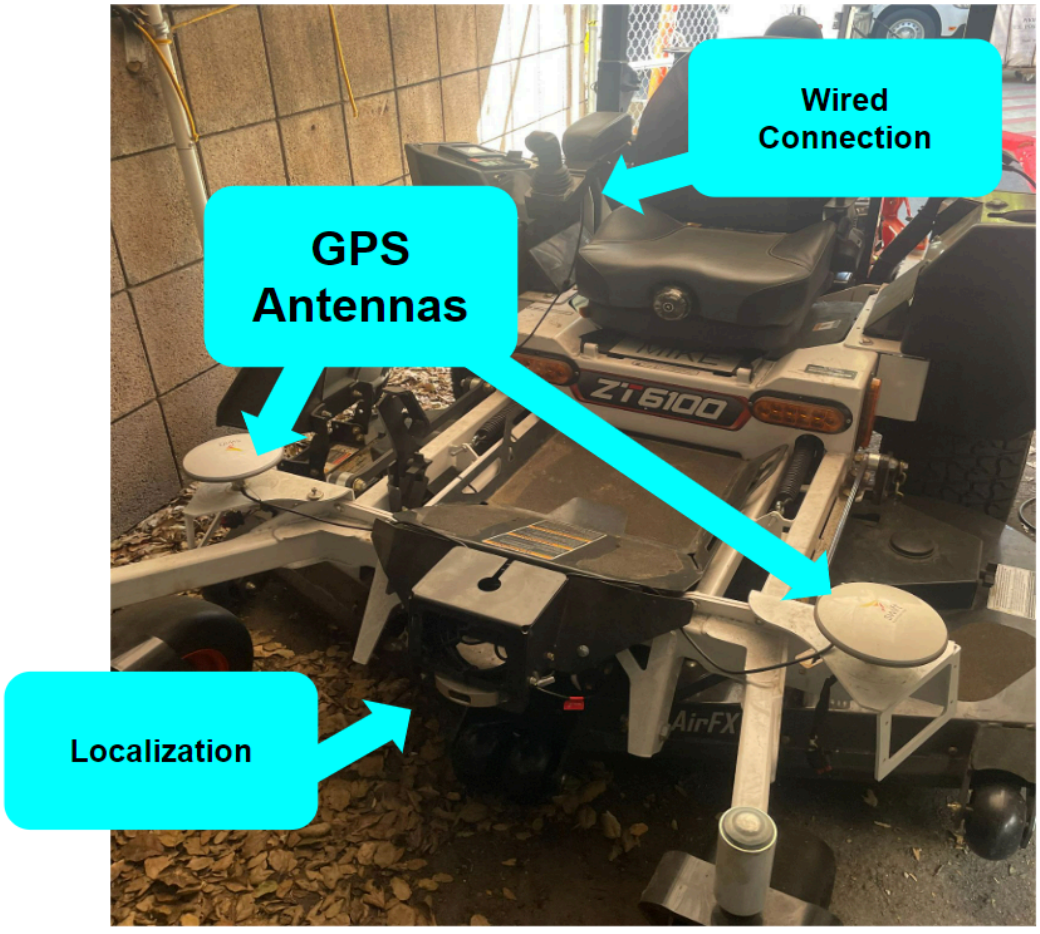

Figure 3: Mower Setup with Sensors and Wired Connection highlighted

There are safety features installed, such as an emergency stop controller and receiver, as well as a seat sensor that prevents the mower from entering autonomous mode if there is a significant weight (i.e. of a possible operator) placed on the seat. In addition, all testing was conducted without the blades actually running to be as safe as possible.

Originally, the V2.5 Board was installed, which allowed programming to be uploaded to the mower itself and be run from there. However, this component was ultimately removed.

It should be noted that the final demonstration presented at the Doosan Bobcat facility was carried out using a ZT6200 mower, which is still in development and not yet commercially available. However, the key system components (GPS, localization, etc.) still operated the same and were compatible with the Harvey Mudd software architecture and controls.

3.2 Communications Interface

Last year, messages were sent to the mower using the AutoControl interface. Essentially, ten different signals could be transmitted to the mower, controlling the mower's engine and wheels.

This year, we initially used an LXC container called MaxControl, which was developed by the Innovation Team & the Clinic Liaisons at Doosan Bobcat. It communicated directly with the V2.5 board on the mower, which alleviates the need for independent CAN communication. MaxControl and the V2.5 board's implementation allowed the mower to accept simple wheel velocities as inputs, simplifying the control process greatly. We connected the computer (and MaxControl) to the mower (and V2.5 board) using an ethernet cable, originally. However, this was briefly replaced with communication over wifi, which allowed for easier development, as the computer did not have to be right next to the mower for operation. A detailed description of the software behind the communications interface is described in section 4.

However, ultimately, the V2.5 board was removed from the mower, and a new communications system had to be implemented. This is because the V2.5 board was getting overloaded and not receiving a GPS fix, so we had to pivot to a less resource-intensive architecture. Now, we communicate with the mower using multi-machine ROS, where the CAN communication is done entirely on the external computer. Commands of wheel velocities are sent in ROS and translated into CAN messages, which are sent to/from the sensors and actuators. The GPS has its own system and node that sends ROS messages straight to the computer, or main controller. Now, all the processing is run on the external computer, and the mower simply receives CAN messages. This system configuration is explained in more detail in section 4.

4. Software

This section describes the work done with regards to software and coding this year.

4.1 First Architecture

In the fall semester, we started by reviewing last year's code repository and deciding what we needed to update for this year. Last year's team wrote a repository in Python that had a number of parts to it. The diagram (Figure 1) above provides information on the software architecture that was used by last year's team. The most important parts of the architecture were the path planning module and the control module. We mostly reused the same path planning code that was written last year, and the only changes were with the control module.

There were a variety of issues that last year's team ran into with the control module, mainly regarding communication and how complex CAN messages are. As shown in the software diagram, last year's team needed to send raw CAN messages to the CAN Bus on the mower system in order to control the engine and both wheels. This required a lot of hardcoding to determine the correct message structure to send, which was a time consuming process. However, the introduction of the V2.5 control unit allowed a simplified communication process. The V2.5 came with software that would translate wheel velocity and throttle commands into raw CAN messages, which alleviated the need for us to write the raw messages ourselves. Instead, we were able to just send two float values from -1 to 1 that were for the left and right wheel throttles. To be able to work in this new environment, we needed to make some updates to last year's code.

As seen in Figure 2, there are two LXC containers living on the V2.5 that communicate with each other using ROS. The first of these containers, the PyBobcat container, contains the control logic and startup procedure. The second container is the MaxControl container, which contains the localization and talking code. These two containers send each other various messages while the mower is moving to make sure that the next waypoint is hit accurately. The PyBobcat container sends wheel velocity commands to the MaxControl container, and listens for GPS, heading, and other mower diagnostic messages. The mower diagnostics are to make sure that the E-stop is active and engaged, and to ensure that the engine is in the correct state for autonomous driving.

The V2.5 came with the MaxControl container already on it, but we needed to port the control logic over from our external computer in order to run the mower autonomously. In order to do that, we repackaged the essential parts of last year's code into a Docker container on our external computer. We also had to add a new file to run the control loop (updating the wheel velocities every tenth of a second) and send ROS messages to the MaxControl container. To keep consistent with last year's work we decided to write this file in Python, using the rospy package. After writing this new comminaction file, we used a script provided by our liaisons to transfer the container over an ethernet connection to the V2.5. In addition, the script changed the container format from Docker to LXC, since LXC containers are less resource intensive, and we were unsure of how much computation the V2.5 board could handle.

During lab testing in the fall, we ran into various issues regarding the computational abilities of the V2.5, and since our liaisons wanted us to run the control logic at 10hz we decided to update our architecture to run less code on the V2.5.

4.2 Second Architecture

In order to minimize the computational load on the V2.5, we decided to run our control logic on the external computer and use multi-machine ROS for communication with the V2.5. In this scenario, the only code running on the V2.5 itself was the localization and talking code. The architecture is diagrammed in Figure 4 below.

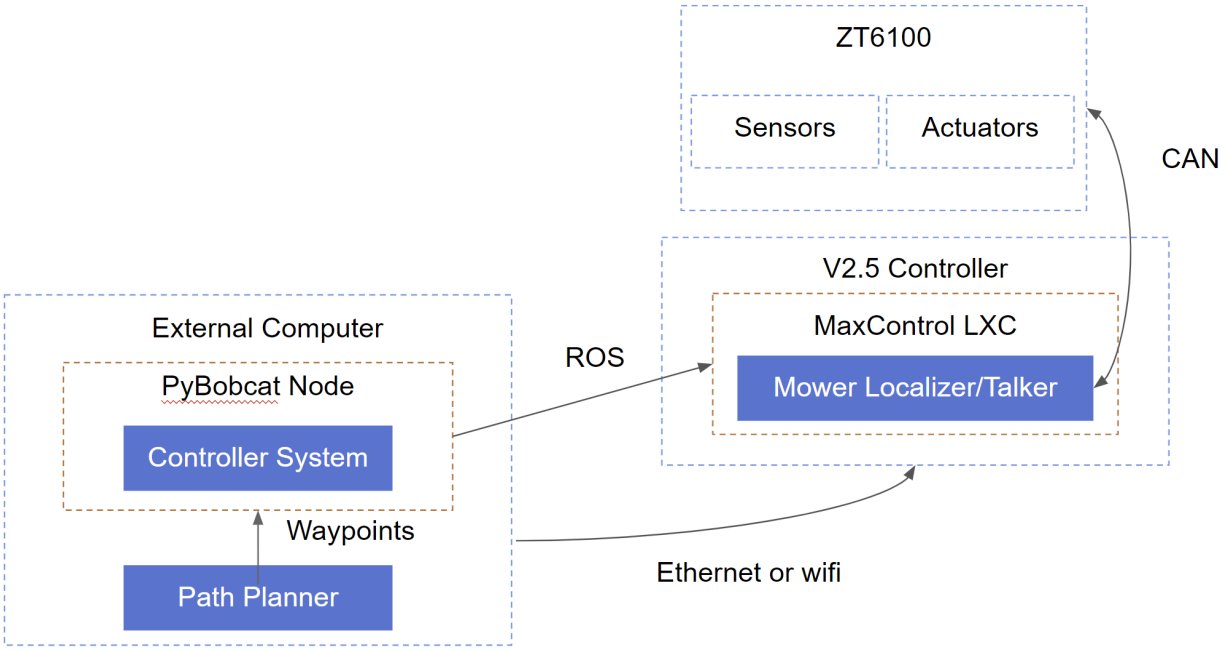

Figure 4: Updated Software Architecture

In order to run the PyBobcat code on the external computer we needed to rewrite the code used from the first architecture. After discussions with our liaisons, we decided that the best way to proceed was to create a ROS package on our external computer and run our control logic inside this package, with a single ROS node used for communicating with the MaxControl container on the V2.5. Due to the fact that the MaxControl code was in C++ and our liaisons preference for working in C++, we decided to write our package using C++ as well. This required a decent amount of translating Python to C++, which was slightly annoying when it came to classes and imports but we were able to get a working ROS C++ package early in the Spring semester.

Running the mower was a slightly different process now since we needed to run code on both the external computer and the V2.5 at the same time. In order to do this, we needed to

launch both the PyBobcat and MaxControl ROS nodes separately, using a single ROS core, and then ensure that the ethernet connection remained in place throughout the duration of the mowing process. This process was working for us early in the Spring semester, as we were able to successfully start the mower without human intervention and then send it hard coded wheel velocity commands. After we tested to verify that our startup procedure and ROS communication for the wheel commands was working as intended, we moved to running the actual control logic instead of just sending hard coded wheel velocity commands.

In order to run our control logic, we needed to use the localizer from MaxControl to receive GPS and heading data from the mower. Doing this required us to subscribe to a new ROS topic which contained the GPS information. However, during testing we noticed that there were issues with the GPS chip on the V2.5 getting a fix. This meant that the GPS data we received was not accurate enough for our purposes, since commercial mowing requires a reasonably high level of precision. Our liaisons helped us debug this issue and they hypothesized that there was a problem with the V2.5 hardware which required us to dramatically change the hardware setup in the mower - and the software architecture as a consequence.

4.3 Final Architecture

Removing the V2.5 changed a variety of things about our software setup. In the second iteration, described above, the V2.5 was doing all of the localizing and CAN communication for us. This made our life easy, since we just needed to send wheel velocity commands instead of the complicated CAN messages that last year's team had to send. Luckily, removing the V2.5 did not change this fact.

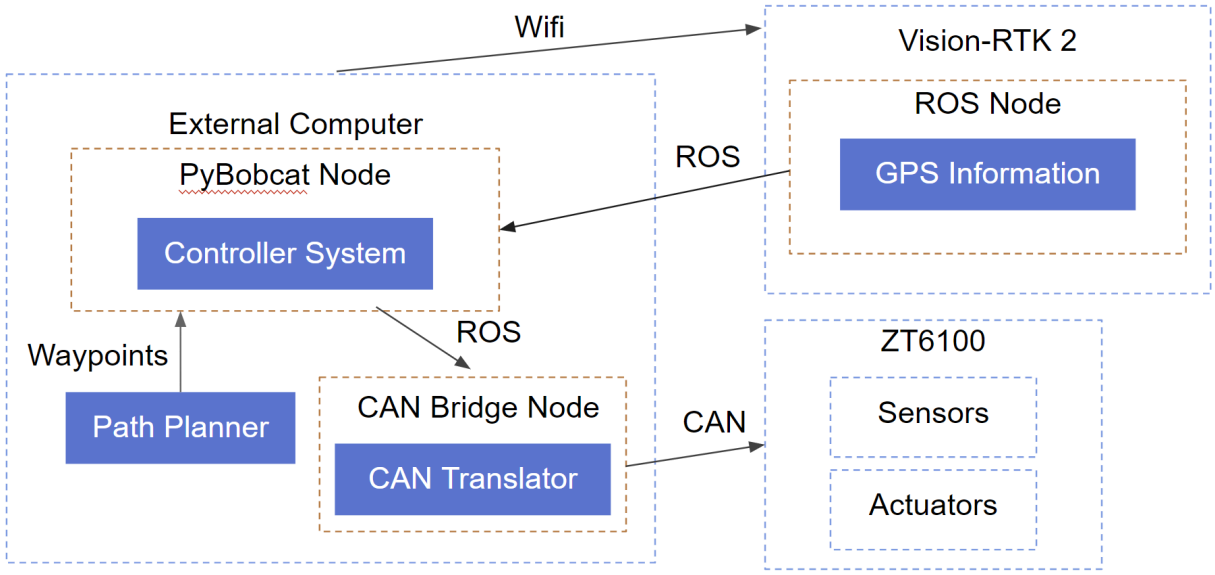

Figure 5: Final Software Architecture

As seen in Figure 5, all of the code now runs on our external computer, and the GPS data is received by the default mower GPS system: the Vision-RTK 2 system. The Vision-RTK 2 system also communicated over ROS, as shown in the diagram above, which meant that our new software architecture has three different ROS nodes. The other big difference from the previous code iteration was that the localization and talking node that previously lived on the V2.5 was now converted to a ROS node running on the external computer. Luckily, running this code on our external computer was simple since both the V2.5 and our external computer ran Linux based systems.

This final architecture was successful in getting a GPS fix and allowed us to run our control code without errors. Path planning was done outside of the control loop, as we generated a list of waypoints from a user inputted boundary before starting up the mower. In order to do dynamic object detection, which would most likely require updating the planned path in the middle of mowing, we would need to include the path planning as part of the PyBobcat ROS package. Once the waypoints have been generated and placed in the correct file location, the CAN translation and GPS nodes can be turned on to prepare for running the control logic. After ensuring that the GPS node has received a fix, the PyBobcat node is started up and the mower moves without further human intervention. It is important to note that even though the software architecture was updated repeatedly, the messages we sent to the mower were still just the float values from -1 to 1 for the wheel commands, unlike last year's team which was sending raw CAN messages.

4.4 Path Planning

The path planning that we used for this year was nearly identical to the system developed by last year's team. The path planning module is responsible for generating a list of waypoints for the mower to reach based on what and how the user wishes to mow. For the purposes of this year's clinic project we did not change the underlying path planning algorithm, called Dubins algorithm. The work that we used is described in Appendix B. The only new work we did on path planning was creating a Python file that we could use to rapidly update the waypoints we wanted to test when we were in the field. The file allows a user to choose a CSV file that is downloaded from Google Maps containing the GPS locations of the boundary vertices, and will output a set of waypoints for the mower to hit in the form of another CSV file. This output file is then read in by the PyBobcat ROS Node to generate the actual mower commands.

5. Experimental

5.1 State Space Representation

The lawn mower control system has several inputs, current GPS coordinates and heading, desired GPS coordinates and heading and outputs necessary wheel speeds to achieve smooth and efficient movement from waypoint to waypoint. Firstly we needed a linear state space representation of the system to be able to achieve any type of controls. So firstly we started with the following state space:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Figure 6: Non-Linear xy state space matrix

Where x,y,θ, are position in a x,y local frame, and θ is the current heading. We can see here however that the system is non-linear so doing controls can only be done using a non-secular equation, meaning we must have the initial conditions every time to run controls, this however is not robust for multiple starting configurations and endings to we must first linearize the system by using new state variables:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$
$$\alpha = -\theta + atan2(\Delta y , \Delta x)$$
$$\beta = -\theta - \alpha$$

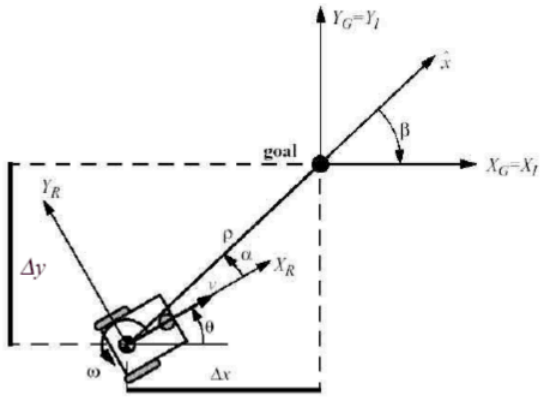

Figure 7: New state space ρ,α,β

The transformation above takes the previous information and describes it as three new variables, ρ the distance between the current position and desired position, α the necessary change in the current heading to face the desired waypoint, and the difference between α and the desired heading. With this change we can write the following control law and come up with the following state matrix:

$$v = k_\rho \rho \qquad w = k_\alpha \alpha + k_\beta \beta \tag{1}$$

$$\begin{pmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{pmatrix} = \begin{pmatrix} -k_\rho & 0 & 0 \\ 0 & -(k_\alpha - k_\rho) & -k_\beta \\ 0 & -k_\rho & 0 \end{pmatrix} \begin{pmatrix} \rho \\ \alpha \\ \beta \end{pmatrix}$$

Figure 8: New state space matrix ρ,α,β

Now, the system has been linearized, now the goal here is to get ρ, α and β to become zero, meaning the mower has reached the desired waypoint. To do this we need to tune the three proportional gains appropriately to ensure stability and this also ensures that our three states reach zero. Finding the determinant of this matrix we get the following necessary inequalities to ensure stability and decay:

$$k_\rho > 0, \quad k_\alpha > k_\rho \quad k_\beta < 0$$

So now we have a linearized state matrix, control gains and using equations (1) which states equations for linear and angular velocities of the mower system we are then able to calculate the wheel velocities. The next step for the control system is to then tune these control gains to produce efficient and smooth paths.

5.2 Control Gain Tuning

With the necessary inequalities we now simply need a way to set the control gains that will result in an efficient and smooth path for the mower. We approached with problem with 3 different solutions:

1. Setting one set of control gains
2. Setting control gains based on distance
3. Setting control gains as a function of distance

1. Setting one set of control gains

Using this method was extremely simple and easy to do. It consisted of simply simulating some paths and updating the control gains until the path produced is efficient and smooth. Below are some simulated paths that use one set of control gains for the entire path:

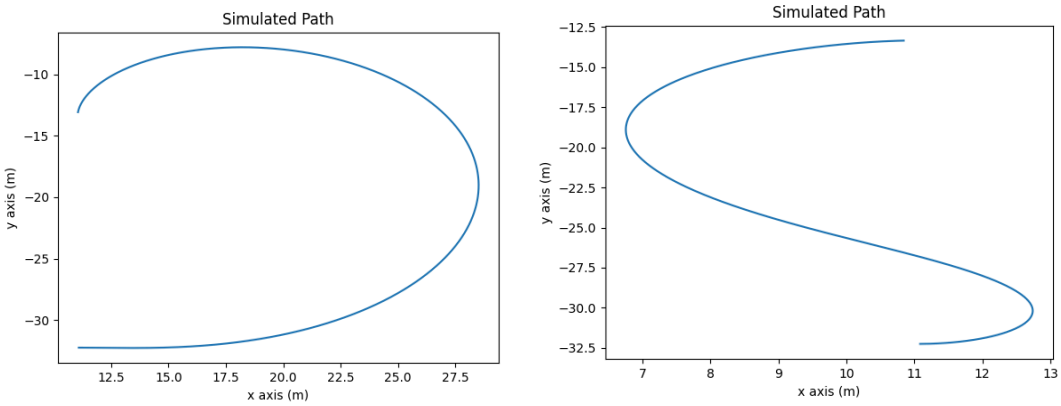

Figure 9: Sim. paths one set of gains

Both of these paths are smooth yet in the case of the left path the mower takes an unnecessarily long path while the right path is shorter. This method is very situational since the gains have to be tuned for each iteration of starting and ending configurations. So despite this method being simple this is not the most efficient way of setting the gains.

2. Setting control gains based on distance

This method consists of changing the control gains once the mower is a specified distance from the waypoint. This allows for the system to focus on different gains/parameters at different points of the path, such as when the mower is close to the waypoint β is more important than α so we can set β to be higher say when ρ=5m. Below are simulated paths that implement this method:

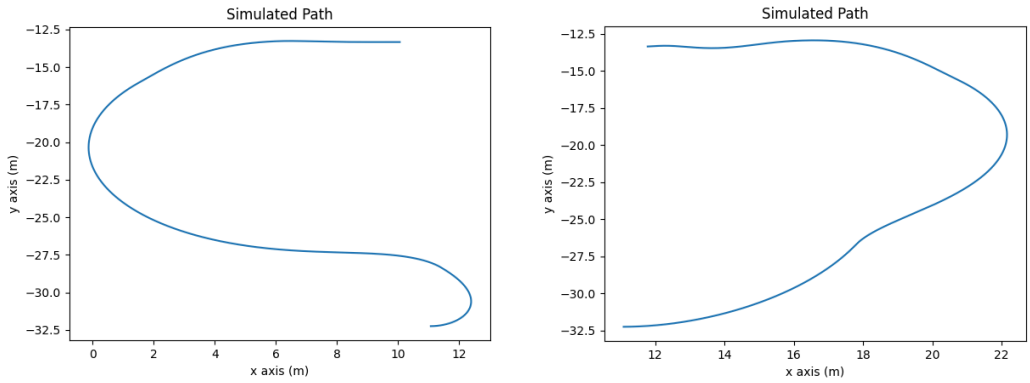

Figure 10: Sim. Paths gains based on distance

We can see that these paths are a step up from the previous methods, i.e. the paths are overall shorter. However these paths still have to be tuned and are situational and these paths may not always be smooth, as seen in the path on the right. This method then is also not as robust as we need for the system.

3. Setting control gains as a function of distance

This method involves writing the control gains k_ρ k_α k_β as functions of ρ. This then allows us to then focus on different gains/parameters at different points of the path. We chose to write the following gains:

$$k_\rho = 5 \qquad k_\beta = \frac{-18}{\rho} \qquad k_\alpha = 5 + \frac{\rho^2}{1.5}$$

We choose to only write k_α *and* k_β as functions of ρ because we found that these are the most important to focus on to ensure straight paths and to ensure the ending heading tolerance it met. These gains resulted in the following simulated paths:

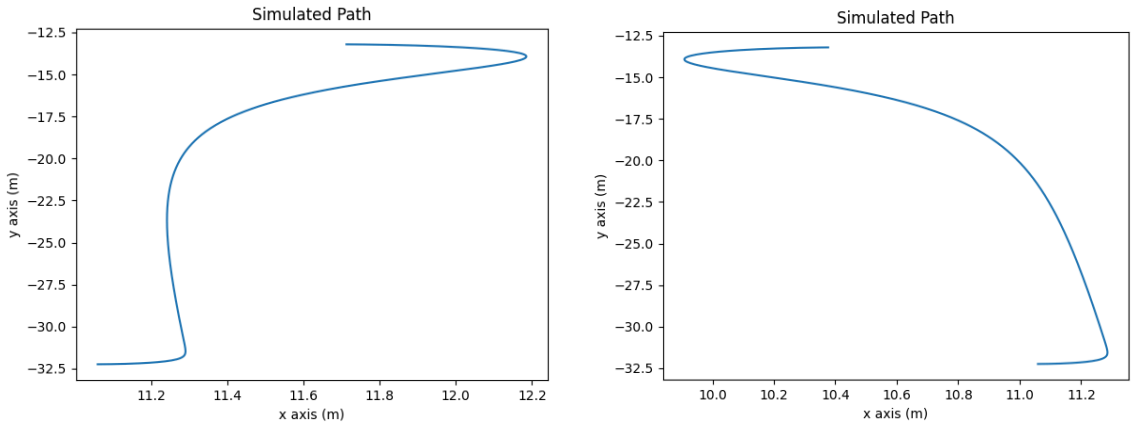

Figure 11: Sim. Paths gains as a function of distance

These paths we see are extremely efficient, the mower does not take unnecessarily long paths and orients itself quickly. This method is by far the most adaptive and robust method for setting the control gains and it is what we will be using for the control system.

5.3 Investigation of setting gains as functions of distance

We found some interesting results when trying different functions to write as functions of distance. Initially we tried to make both k_α *and* k_β linear functions of ρ such as the following:

$$k_\beta = \frac{-18}{\rho} \qquad k_\alpha = 5 + \frac{\rho}{0.5}$$

This set of functions led to a certain behavior we called “snaking.” This behavior’s main characteristic is that the path taken by the mower is not straight and instead turns slightly left then slightly right. This can be more clearly seen in the simulated path below:

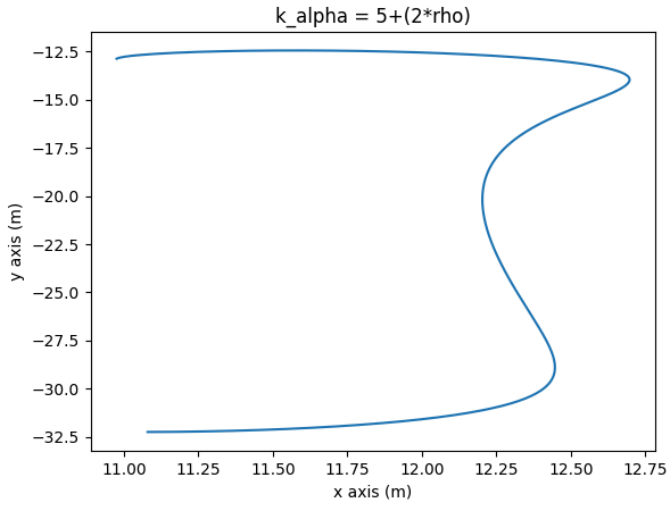

Figure 12: Sim. Path Snaking Behavior

We can see that the path taken here is not straight and the mower deviates by 0.75 meters. This behavior is believed to be caused because both k_β and k_α are linear functions of ρ. Because they are both linear there is a relatively middle of the road point between the waypoint and the initial starting position where k_β becomes larger than k_α. So initially the mower points itself toward the waypoint but then halfway through the path it decides that k_β is more important so it reorients itself to prioritize the ending heading resulting in a “snaking” path.

Here in these simulations the mower starts facing East and wants to end facing West, and the only thing it needs to do is go straight up (+y direction).

To combat this issue of snaking we decided to make k_α a quadratic function of distance to hopefully ensure a more straight path. Using that equation seen in the previous section:

$$k_{\alpha} = 5 + \frac{\rho^2}{1.5}$$

We get the following simulated path with the same setup as before:

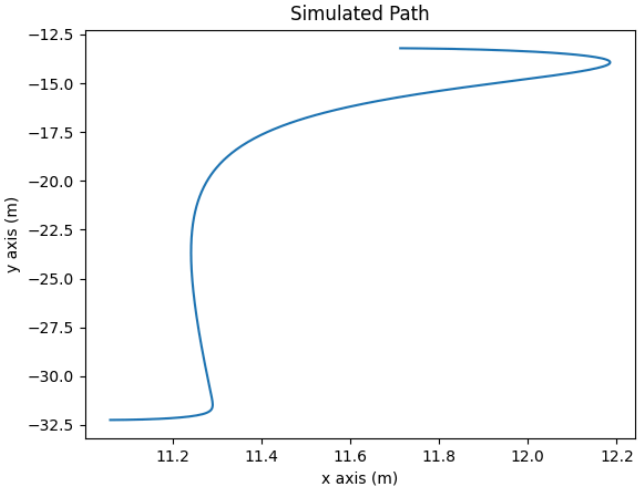

Figure 12: Sim. Path k_α as a quadratic

Here we see the main issues of the previous path being resolved. We see that there is a less than 0.05m deviation from a straight path until it reaches the need to make a turn which is far better than the previous 0.75m deviation.

Another behavior we saw was that because these gains are functions of distance, they may explode (Increase rapidly). This is mainly the case with k_β since k_α was designed to decay over time/distance in contrast with k_β which was made to increase over time/distance. This behavior has occurred here when testing at Mudd and here is a simulation of what that behavior looks like:

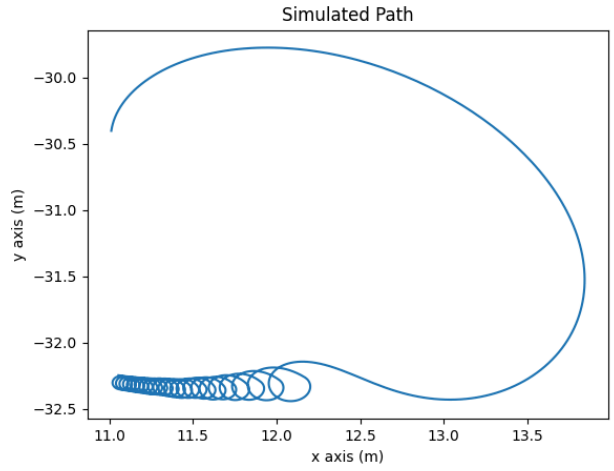

Figure 14: Sim. Path Gains Explode

We can see that once the mower got close to the desired waypoint that it began to have issues. Once the mower got this close it began to correct for its heading but since the gain was so high it overshoot, leading it to turn more and once again overshoot again. This problem can be mitigated by further tuning and setting a tolerance for both β and ρ. Meaning that we say we are

close enough to a waypoint when ρ<1m and β<0.35rad and therefore we switch to the next waypoint.

5.4 Tuning Observations

This year we had the opportunity to tune on two different kinds of mowers, the ZT6100 and the ZT6200 which had different gain needs. The ZT6100 used the tuned gains shown above in the previous section which resulted in no overshooting and extremely efficient paths. The ZT6200 had slightly different tuned parameters that resulted in similar behavior:

$$k_\beta = \frac{-6}{\rho} \qquad k_\alpha = 5 + \frac{\rho^2}{1.75}$$

This is noticeably different from the other mower type. There is far less emphasis on k_β than in the ZT6100 case. This might be explained by the change in correction service for the RTK system. When we tested in ND with the ZT6200 we had to change the correction service to a different tower in the area, this then noticeably gave GPS data and heading data slower than the set-up in Claremont therefore the system needed to move more slowly to 'keep up' with the data coming in and be able to check if it has reached the waypoint more carefully.

In general to tune these parameters start a simulation with a guess that will ensure stability and then tune around the observed behavior. And in general this is what has been observed from the team

- k_ρ: Only needs to be set as a small positive constant integer such as 5. The most important thing for the system to be able to do is face the waypoint and execute turns efficiently meaning, k_α and k_β need to be flexible and adaptive and not necessarily k_ρ.
- k_β: A good start is $\frac{C}{\rho}$ where C is a negative real integer such as -10. If the system overshoots then decrease C's magnitude and vice versa if the system does not reach the desired ending heading.
- k_α: A good start is a quadratic function with k_ρ added such as k_ρ + V* ρ^2 and a similar recommendation as k_β, if it overshoots then decrease V and if it has trouble following a straight line increase V.

5.5 Radar

Last year the team started work towards detecting stationary obstacles (this exploration can be found in the appendix). The mower is equipped with a O-79 Ainstein radar which the team worked with to successfully create a visualized map of occupied and unoccupied space.

Below is last years team's experimental setup and results:

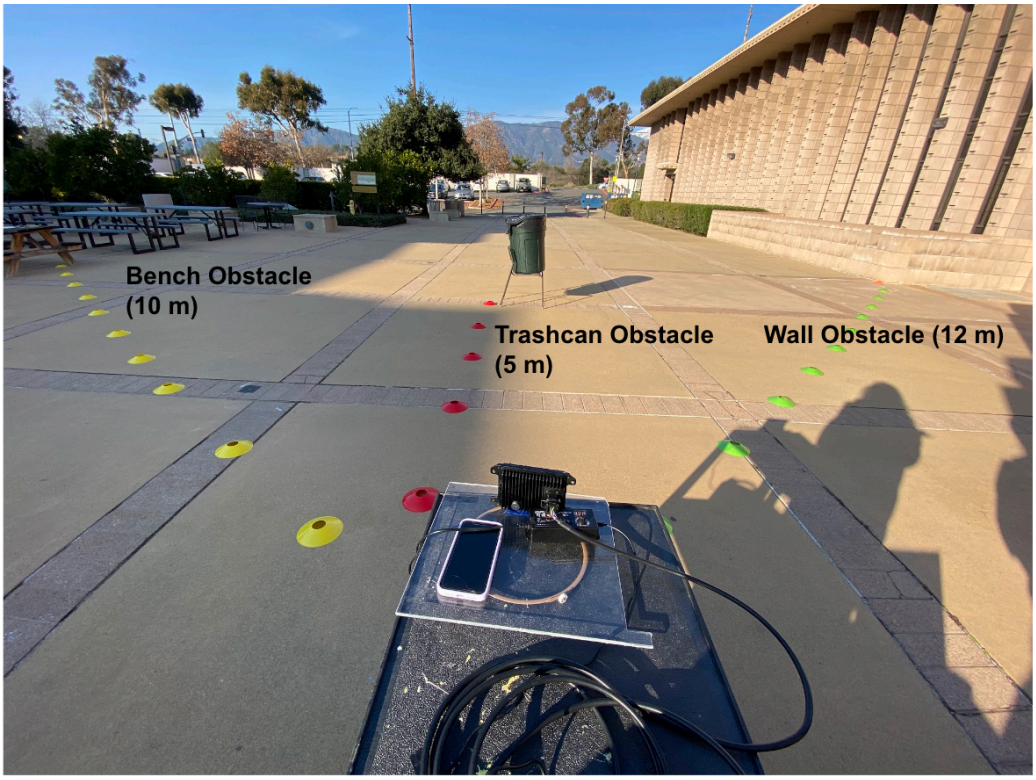

Figure 15. Experimental setup of the radar with a trash can on a chair as our main obstacle. Cones on the ground are spaced 1m apart for scale and reference. The radar and phone are rotated on a turntable to collect data.

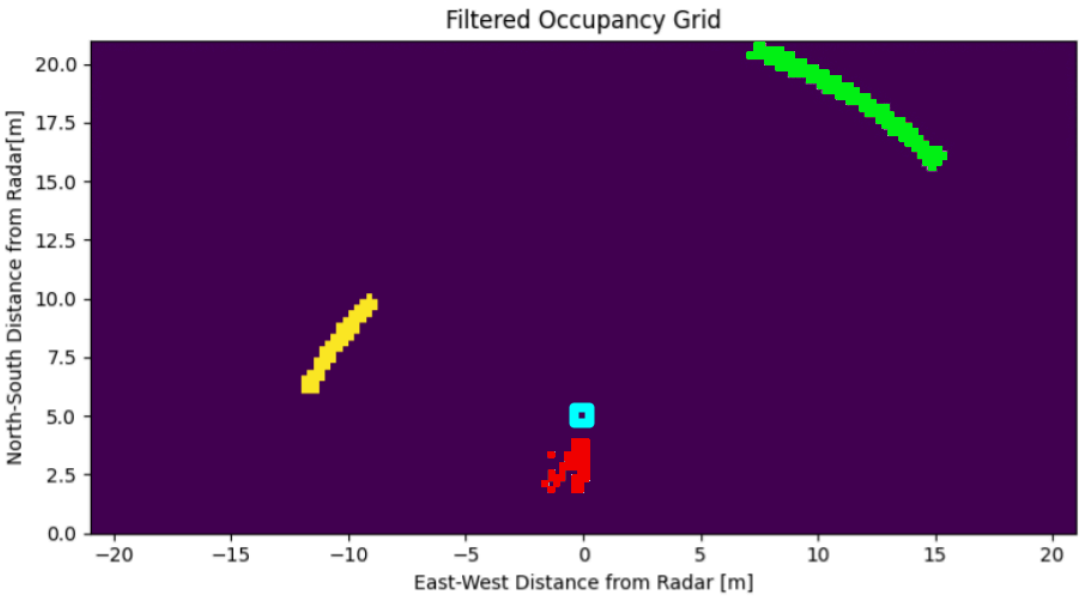

Figure 16. birds eye view occupancy grid plotted with the mapping system. The cells are discretized into squares with quarter meter resolution. Purple represents unoccupied space on the map.Three distinct occupied clusters match the objects' respective cones (yellow left, red center, green right). Blue square in the middle represents the ground truth position of the main trash can obstacle.

Last year the team was able to successfully map static obstacles in a discretized space. The setup included using one radar, and compass data from a cell phone. The results were promising with low error when the radar was static. However when the radar was turned the error increased.

This year the problem changes as the mower is now equipped with 5 radars instead of 1. This allows the mower to have a greater field of view and will help decrease the uncertainty for any detected objects as the objects will be seen by multiple radars allowing for the state of the object to be more certain.

Work on this will include sensor fusion, combining the information from all five radars to create a more accurate map of the mower's surroundings. Firstly a new local map of the mower will need to be constructed. With the five radars the mower has a 360 degree up to ten meters. We will discretize this rectangular space with 0.25m by 0.25m blocks discretizing the space resulting in 6400 discretized states. Each state will be represented by a total length vector, from the midpoint of the state to the origin, and an angle, positive counterclockwise respective to the local x axis. To make this case simpler we will assume static obstacles and a static mower.

The map will be centered on the center of the mower seat with 8m going in the +x, -y and -x, -y directions. With 0.25m by 0.25m states we will need 4096 states to fully discretize that

space. Additionally the position and orientation of the radars in this frame will be needed. Each radar's position and orientation (angle positive CCW) is listed below:

Radar	Local Position (x,y)	Local Pose (deg)
1	(0.568m, 1.21m)	25
2	(0, 1.328m)	90
3	(-0.568m, 1.21m)	155
4	(-0.461m, -0.312m)	250
5	(0.461m, -0.312m)	290

Table 1. Position and orientation of radars in local mower frame

Each radar has been numbered according to the following map below, shown as well is the local frame, the positive x and y directions are outlined in orange:

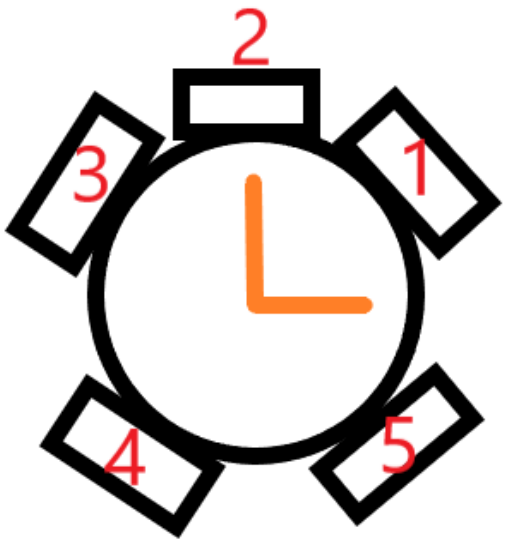

Figure 17. Radar Numbering

With these orientations the radar data can be transformed to be in the local mower frame. The radars report azimuth angle, range, and an elevation angle, for the purposes here we will assume the elevation angle is not necessary, since we will assume a flat terrain. The data reported from the radar can be transformed with the given equations:

$$x_{local} = x_{radar,pos} + Rcos(\theta_{radar,pos} + (-\theta_{radar,mes})) \quad (1)$$

$$y_{local} = y_{radar,pos} + Rsin(\theta_{radar,pos} + (-\theta_{radar,mes})) \quad (2)$$

Where $x_{radar,pos}$ is the local position of the radar in the local frame, $\theta_{radar,pos}$ is the angle of the radar in the local frame, $\theta_{radar,mes}$ is the reported azimuth angle from the radar, and R is the reported range from the radar. We can then take this one step further to place objects in the global reference frame to be able to place obstacles in the input map for path planning. This transformation is as follows:

$$x_{global} = x_{mower} + x_{local} cos(\theta_{radar,pos} + (-\theta_{radar,mes}) - (90^o - \theta_{mower})) \quad (3)$$

$$y_{global} = y_{mower} + y_{local} sin(\theta_{radar,pos} + (-\theta_{radar,mes}) - (90^o - \theta_{mower})) \quad (4)$$

Where x/y_{global} is the x/y position of the detected object in a global frame with +y pointing to true north and +x pointed 90^o clockwise, x/y_{mower} is the current position of the mower in that global frame in meters, obtained from gps data, θ_{mower} is the current heading of the mower respective to true north CW positive. The rest of the constants are the same as in equation #1.

To test the initial radar transformation, let us assume the following:

- 3 radar inputs from 1 radar
- Radar local position and local pose: [1,1,45^o]
- Radar inputs:

Range (m)	**Azimuth (deg)**	**Elevation (deg)**
1.4	10	0
1.75	25	0
2	-15	0

Table 2. Test input radar data

With this input the map generated is the following:

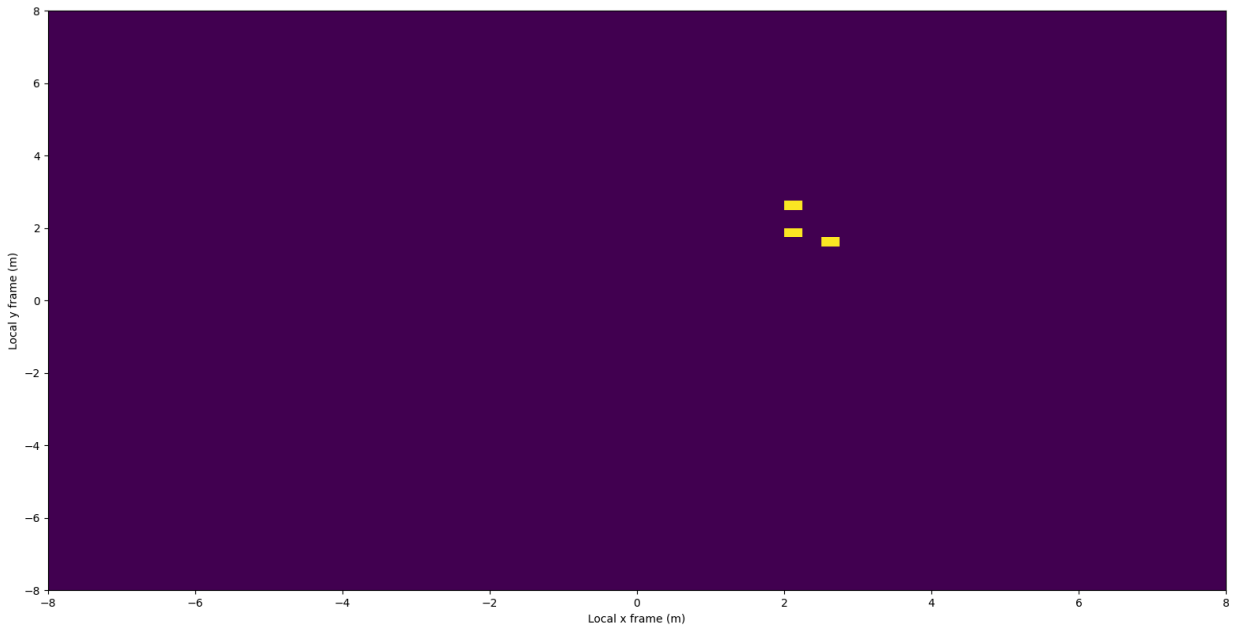

Figure 18. Generated Occupancy Grid map

Now since we know the inputs we can calculate where these potential obstacles should be:

For the first input, data=[1.4,10,0] (Range (m), Azimuth (deg), Elevation (deg)) and equations 1,2:

$$x_{local} = 1 + 1.4 * cos(45 + (-10)) = 2.146 \text{ m}$$
$$y_{local} = 1 + 1.4 * sin(45 + (-10)) = 1.803 \text{ m}$$

All calculations were done in degrees.
We can repeat this for the other two inputs [1.75,25,0] and [2,-15,0]:

$$x_{local} = 1 + 1.75 * cos(45 + (-25)) = 2.644 \text{ m}$$
$$y_{local} = 1 + 1.75 * sin(45 + (-25)) = 1.598 \text{ m}$$

$$x_{local} = 1 + 2 * cos(45 + (15)) = 2 \text{ m}$$
$$y_{local} = 1 + 2 * sin(45 + (15)) = 2.732 \text{ m}$$

Now we have the following viewed obstacle positions:

X position in local frame(m)	Y position in local frame (m)

2.146	1.803
2.644	1.598
2	2.732

Table 3. Transformed radar test input

Looking closely at the generated OGM we can see that all of these obstacles are reported as occupied by the algorithm, validating it:

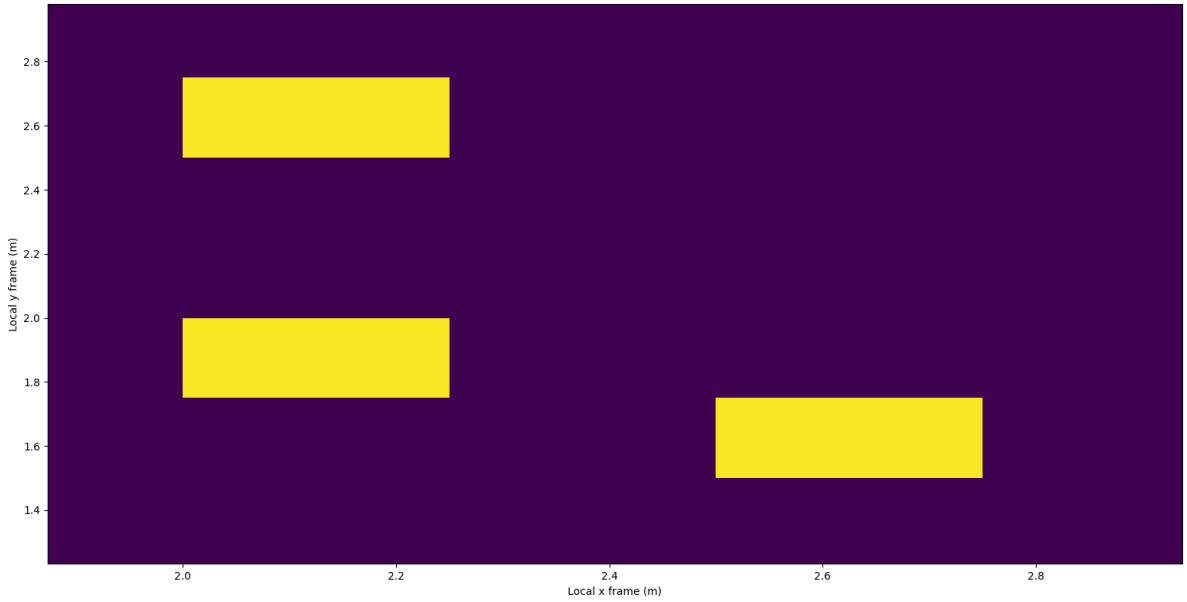

Figure 19. Zoomed in generated OGM

Throughout the fall semester we tried to work with the radar system on the mower. However, we had a multitude of problems getting this data. The radars that we have from previous clinic projects all turned out to be broken. They were also different from the sensors that are mounted on the current mower. We unfortunately did not get the opportunity to collect data from the radar sensors mounted on the mower. This is something that can be explored further.

Conclusions and Future Work

All in all, the team was able to accomplish the goals that it set out for this year. We were able to have the mower mow a bounded area autonomously both at Linde field and during our site visit to North Dakota. The mower made relatively smooth turns and was able to run for several minutes without issues getting data or calculating and sending commands to the mower.

In the future, there are a variety of improvements future Clinic teams could make for the system to run more smoothly and be more usable for end users. The first and most pressing issue is to update the control system by adding both integral and derivative control, since the current system is only based on proportional control and suffers on the turns because of this. More testing on Linde Field would be needed to tune these controls, and this should be a main priority for next year's team in the Fall semester.

In addition to improving the controls for the mower, another main priority for next year's clinic team should be focusing on more complicated areas to mow. This year's team was able to successfully mow bounded rectangles, but designing more complicated shapes should be a focus for next year. In particular, finding out how to link multiple bounded areas (as seen in figure A15) when generating paths is something that was not tested this year and would be important for future years. Testing even more complicated areas to mow, such as curved areas or areas that are not flat would be the next step.

There should also be improvements to the front end experience for users, as this is currently non-existent. Next year's clinic team should try to build an autonomous system such that it can be used by non-technical users easily. This would mean that users would be able to upload a file containing lat/lon data of a boundary to mow, and with only button presses, get the mower to do so autonomously. It would be helpful if next year's team had at least one or two C.S. majors who could focus on this aspect of the project.

References

[1] Doosan Bobcat Final Report 2023

Appendix A: Radar

Introduction

Mapping is an important process for autonomous vehicles (AVs) to safely navigate their environment. Maps capture the environment around AVs by identifying obstructed areas and free areas. This distinction is important because obstructed areas will impede robot movement and should be avoided to continue uninterrupted robot operation. Detection of obstacles in the environment is therefore essential to generate maps that are representative of the environment. One form of hardware capable of detecting objects is radar, which can be used to support mapping applications in AVs.

Application Scenario

The high-level goal for the Doosan Bobcat 2022-2023 clinic team is to implement path-planning algorithms to make a lawnmower autonomous. To achieve this goal, one of the key tasks is to implement a mapping algorithm using static obstacle detection data from the supplied O-79 Ainstein radar. This radar is already mounted on the smart mower prototype, shown in Fig A1., ready for mapping algorithms to be applied. This project seeks to make progress on this application.

Figure A1: Doosan Bobcat prototype smart ZT6100 mower. Mounted Ainstein O-79 is circled in green

Problem Definition

The objective of this project is to accurately represent the positions of objects near a robot, and the function of the solution is to generate a visual map that represents the object's position. The

problem constraints are twofold. First, the provided O-79 Ainstein radar must be used. Second, the minimum size for object detection is the size of a crouching adult, approximately three to four feet tall.

To simplify the problem for the first mapping prototype, two further assumptions are also made. First, the objects being detected are static. Second, the radar can only rotate in the yaw direction with a stationary position. This means the radar only has a single degree of freedom, and the relative velocities of the radar and its nearby objects are zero.

Data & States

Input Measurements

The input measurements to the system are: 1. distance from the robot to nearby objenorthct(s), 2. angle or direction of a nearby object(s) from the robot, and 3. global pose of the robot relative to the true north.

For this project, the robot is defined as an unmounted O-79 radar placed on a turntable 36 in. (0.91 m) above the ground, shown in Fig. A7.

Data Sources

Two pieces of hardware are responsible for providing the input measurements. An unmounted Ainstein O-79 radar provides the input measurements 1 and 2 as numbered in the previous section. An Apple iPhone 11 smartphone compass provides the third input measurement. Supporting software that extracts these measurements are described in the **Appendix**.

State Variables

The state variables consist of 3528 cells in a 2D discretized space or grid around the radar. The value stored in each cell/state variable is binary–either 1 for occupied or 0 for unoccupied.

The grid around the radar represents the plane parallel to the surface of the Earth spanning 21 meters to the left, right, and front of the radar. The orientation of the grid is fixed to true north. Each cell is a square with an area of 0.125 square meters or 0.25 meters on each side. This creates a rectangular grid 84 cells wide (east to west from right to left) and extends 42 cells in front (north to south from front to back) of the radar when it is pointing north.

Each cell is also centered on a x_g and y_g coordinate in an egocentric 2D Cartesian space hereafter referred to as the “global frame.” Each unit in the global frame represents 1 meter, and zero degrees is measured from the radar towards the north with positive in counterclockwise rotation looking at the ground. Therefore, the radar is at the origin (0,0). The cell at the northwest corner of the rectangular grid is centered on (-20.875, 20.875) at a relative angle of 45 degrees The cell at the southeast corner of the rectangular grid is centered on (20.875, 0.125) at a relative angle of -90.343 degrees.

Data, Coordinate Systems, and Intermediary Variables

Each data source uses a slightly different coordinate system, and both are different from the one used for the state variables. Intermediary variables are then used to combine the coordinate systems.

Consider the radar first. Raw byte data from the radar reports two values for each target tracked: range R, azimuth angle θ, and elevation angle ϕ. The relative coordinate frame fixed to the radar follows the ROS REP 103 standards, with the X axis pointing forward, Y axis pointing left, and Z axis pointing upwards. Targets in 3D space are visualized in Fig. A2.

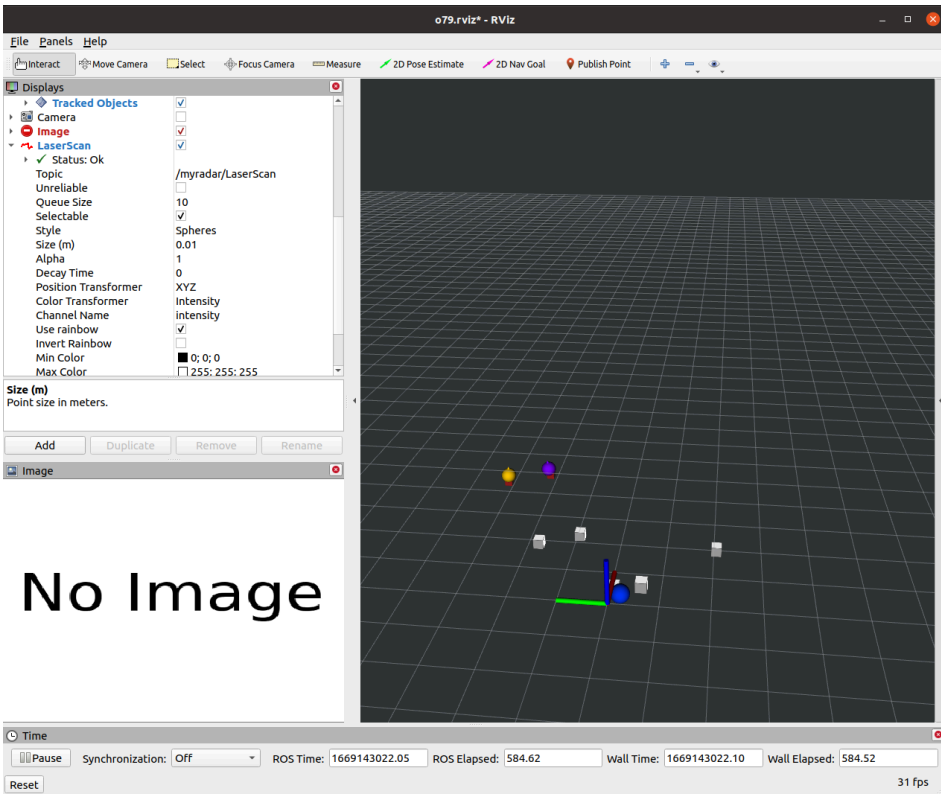

Figure A2. Raw data from the Ainstein O-79 Radar visualized in Rviz, a ROS app. Targets are highlighted in the white boxes and colored spheres. The axes use the ROS coordinate frame (X,Y,Z in red, green, blue).

Range and azimuth data is first used to convert the target coordinates into X and Y following (1), which compresses the coordinate into 2D space by dropping ϕ. Using (A2), X and Y coordinates are then converted into another intermediary coordinate frame x_{local} and y_{local}, which is almost equivalent to the global frame, except the orientation follows the heading of the radar instead of true north.

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} R \\ R\tan\theta \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} x_{local} \\ y_{local} \end{bmatrix} = \begin{bmatrix} -Y \\ X \end{bmatrix} \quad (2)$$

These two radar-centered coordinate systems are illustrated in Fig. A3.

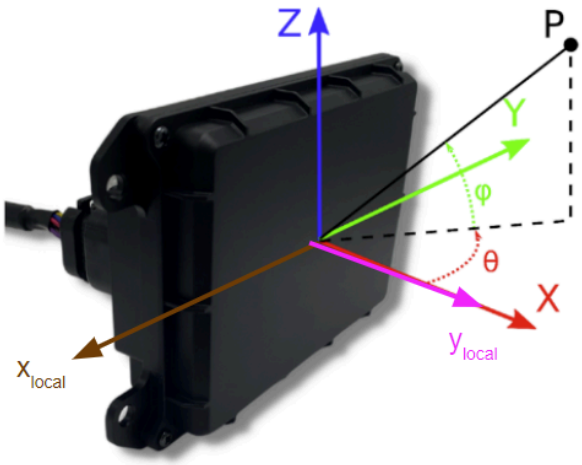

Figure A3. The O-79 Ainstein radar reports the positions of detected targets using the ROS coordinate frame (X,Y,Z in red, green, blue). Local coordinates x_local and y_local in brown and pink follow the heading of the radar as well.

Now, consider the iPhone compass. Heading data *θ'* changes with the orientation of the iPhone pointing from the bottom to the top of the touchscreen. The heading is reported in degrees with 0 north, 90 east, 180 south, and 270 west, shown in Fig. A4.

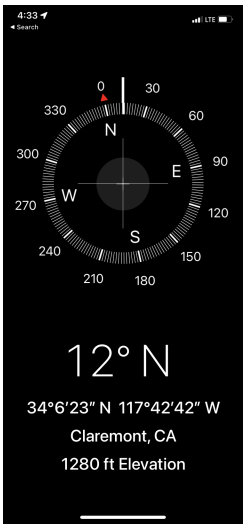

Figure A4. iPhone Compass Measurement.

This system is therefore positive clockwise looking down–reverse from the global frame heading To convert the angles into Cartesian form and match the global frame, simply flip the sign of θ'. When placed behind the radar, the iPhone heading can be used as a proxy for the global heading of the radar.

Finally, using the global heading of the radar, the local coordinates of the targets oriented to the radar can be rotated into the global frame in (A3), where z represents the measurement to be used in the mapping algorithm.

$$z = \begin{bmatrix} x_g \\ y_g \end{bmatrix} = \begin{bmatrix} \cos(-\theta') & -\sin(-\theta') \\ \sin(-\theta') & \cos(-\theta') \end{bmatrix} \begin{bmatrix} x_{local} \\ y_{local} \end{bmatrix} = \begin{bmatrix} \cos(\theta') & \sin(\theta') \\ -\sin(\theta') & \cos(\theta') \end{bmatrix} \begin{bmatrix} x_{local} \\ y_{local} \end{bmatrix} \quad (3)$$

Method

The team implemented the occupancy grid algorithm described in *Probabilistic Robotics*[1]. The prediction step of static obstacles is trivial; with static obstacles that do not move relative to the static robot, the occupancy of the grid should not change between measurements. The correction step uses the inverse measurement range model to calculate the log odd probabilities (A4) for the occupancy of each cell, where m_i represents one cell in the grid, $z_{1:t}$ represents measurements of all past measurements from the radar, and x represents the pose of the robot.

$$l_{t,i} = \log \frac{p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})}{1 - p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})} \quad (4)$$

The inverse range model classifies cells as occupied, free, or unobserved. The pseudocode provided by *Probabilistic Robots* for the model is shown in Fig. A5. Herein, x, y, θ are the pose of the object, which is (0,0, θ) for the stationary radar with yaw θ. θ_j,sens is the angle of the radar measurement in the global frame relative to θ. α is the thickness of the object set to 0.5 m. β is the angle width of the object in the perceptual field of the radar set to 1.5 degrees.

1: **Algorithm inverse_range_sensor_model($\mathbf{m}_i, x_t, z_t$):**
2: *Let* x_i, y_i *be the center-of-mass of* $\mathbf{m}_i$
3: $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
4: $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$
5: $k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$
6: *if* $r > \min(z_{\max}, z_t^k + \alpha/2)$ *or* $|\phi - \theta_{k,\text{sens}}| > \beta/2$ *then*
7: *return* l_0
8: *if* $z_t^k < z_{\max}$ *and* $|r - z_t^k| < \alpha/2$
9: *return* l_{occ}
10: *if* $r \leq z_t^k$
11: *return* l_{free}
12: *endif*

Figure A5. Inverse Range Model Pseudocode.

In Fig. A5, lines 6 and 7 implement the unobservable case. This occurs when the cell grid is at a range beyond or behind the measured target, or outside the angle width of the perceptual field. With no new evidence, the posterior equals the prior probability l_o. Lines 8 and 9 describe the occupied case when the grid cell is close to the target's range and azimuth. Lines 10 and 11 describe the unoccupied or free case, which applies to grids located between tracked targets and the radar. In the inverse range model prototyped for this project, the log odds posteriors l_{occ} and l_{free} are set to 1 and 0 for simplicity, rendering the final output as a binary occupancy grid.

This method of state estimation is similar to the mapping algorithms discussed in the E205 State Estimations class. The main difference is that x is fixed at the egocentric origin in the global frame. The general class of the inverse range model is the Bayes filter–it applies the prior belief of the occupancy l_o by default, and updates the posterior beliefs l_{occ} and l_{free} given new evidence from the radar measurement.

Altogether, the overall system block diagram is shown in Fig. A6.

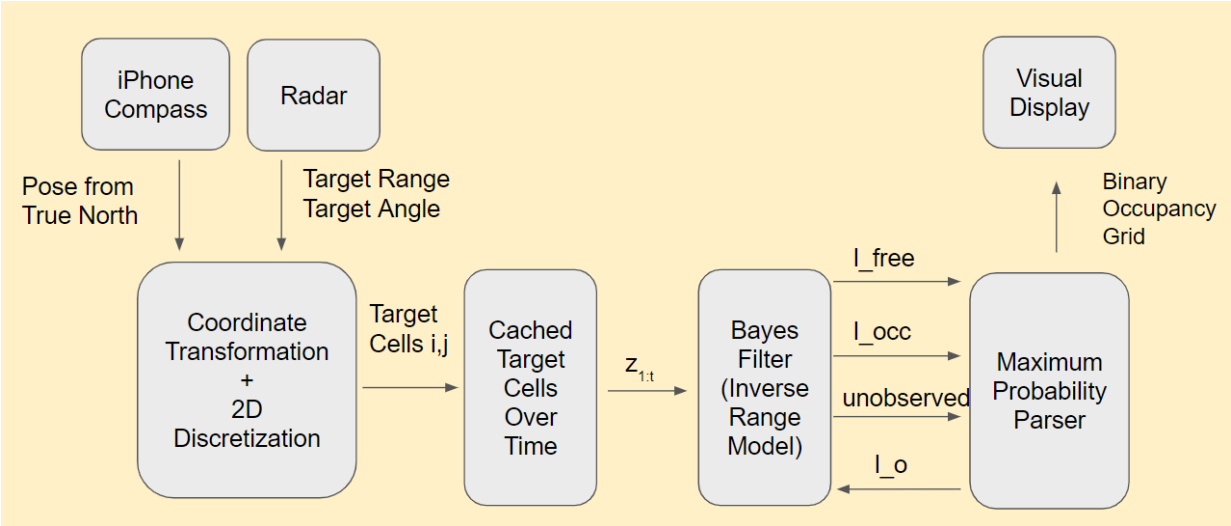

Figure A6. Block diagram of the mapping algorithm.

Experiment

The experimental setup is captured in Fig. A7. Experiments were carried out in an open quad between Sprague, Parsons, and Olin buildings to avoid reflections of radio waves. The O-79 Ainstein radar and an aligned Apple iPhone 11 were mounted on a turntable to allow rotation in the yaw direction without significant displacement in position. The turntable is placed on a cart, placing the radar 36 inches or 0.91 m above the ground. The perceptual field of the radar extends 90 degrees left and right from the *X* axis of the radar and 21 meters away from the radar.

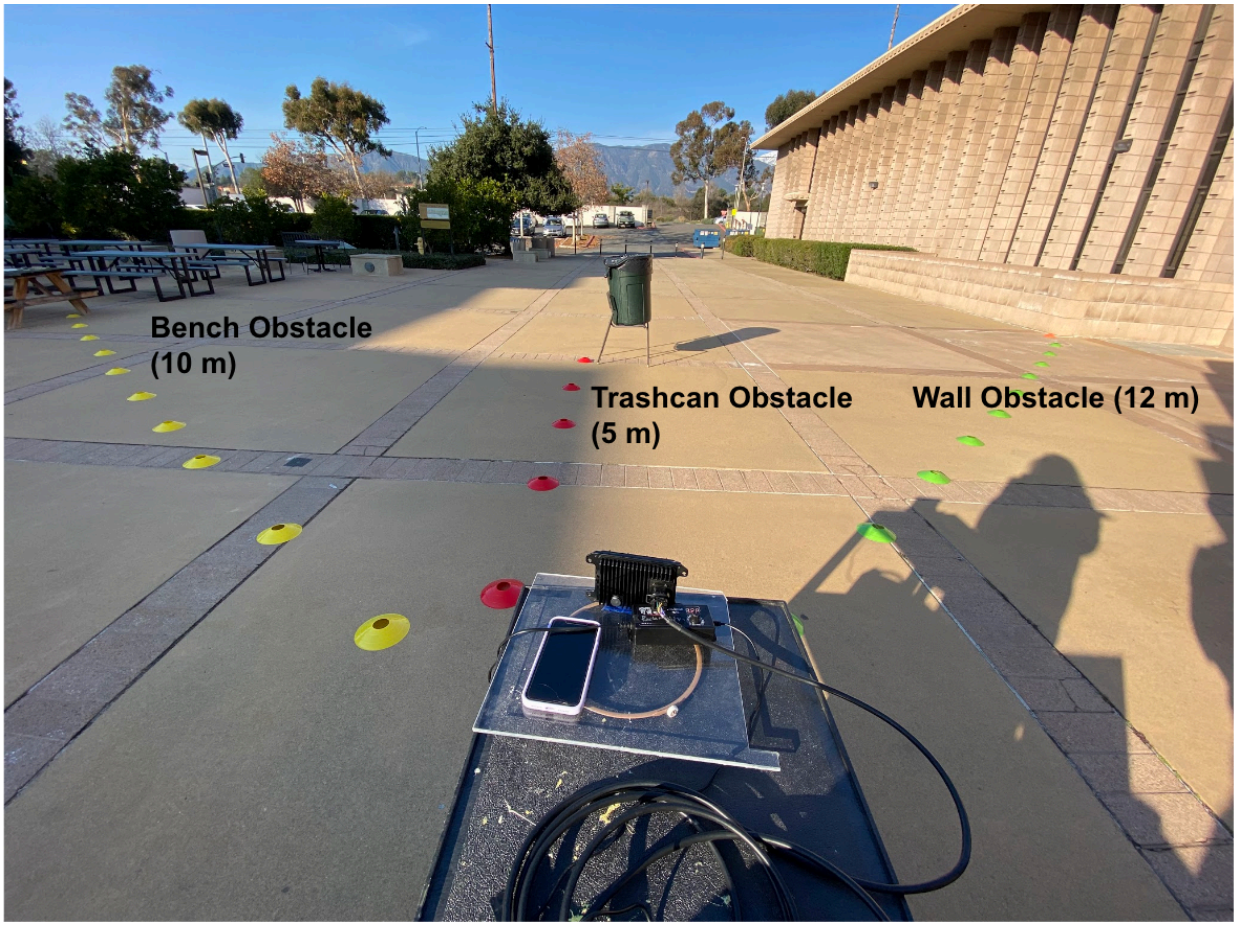

Figure A7. Setup of the experiment with radar mounted on an acrylic turntable atop the black cart. 3 distinct obstacles in front. Cones of the same color are separated at 1-meter increments. Yellow cones lead to the benches, red cones lead to the trash can lifted on a wire chair, and green cones lead to the adjacent building (Parsons).

Two objects fixed in the environment were placed in the radar's range. First, a building wall 12 meters away at about 45 west of north. Second, an array of benches 10 meters away at about 45 degrees east of north.

The experimental conditions were varied by two independent variables. First, a moveable object is positioned as the independent variable–a trash can about 0.5 meters in diameter and 1 meter in height is placed at 2, 3, 4, 5, or 6 meters straight ahead in the north direction. A second independent variable is whether the turntable is static facing north or turning between the line of yellow cones to the line of green cones. For each trial, the system would run for about 15 seconds, and a screenshot of the occupancy grid plot would be captured.

To baseline results, the trash can was removed and data was captured from the environment in both static and turning turntable trials. To demonstrate the validity of using a lifted trash can, trials with a standing and crouching person were also performed.

Results

Baselining

For all occupancy grid plots, purple represents unoccupied cells and yellow represents occupied cells.

The baseline results of the occupancy grid are shown in Fig. A8. As expected, no occupied cells appeared in the static case, but the environment of the building wall was captured at around 15 meters from the radar.

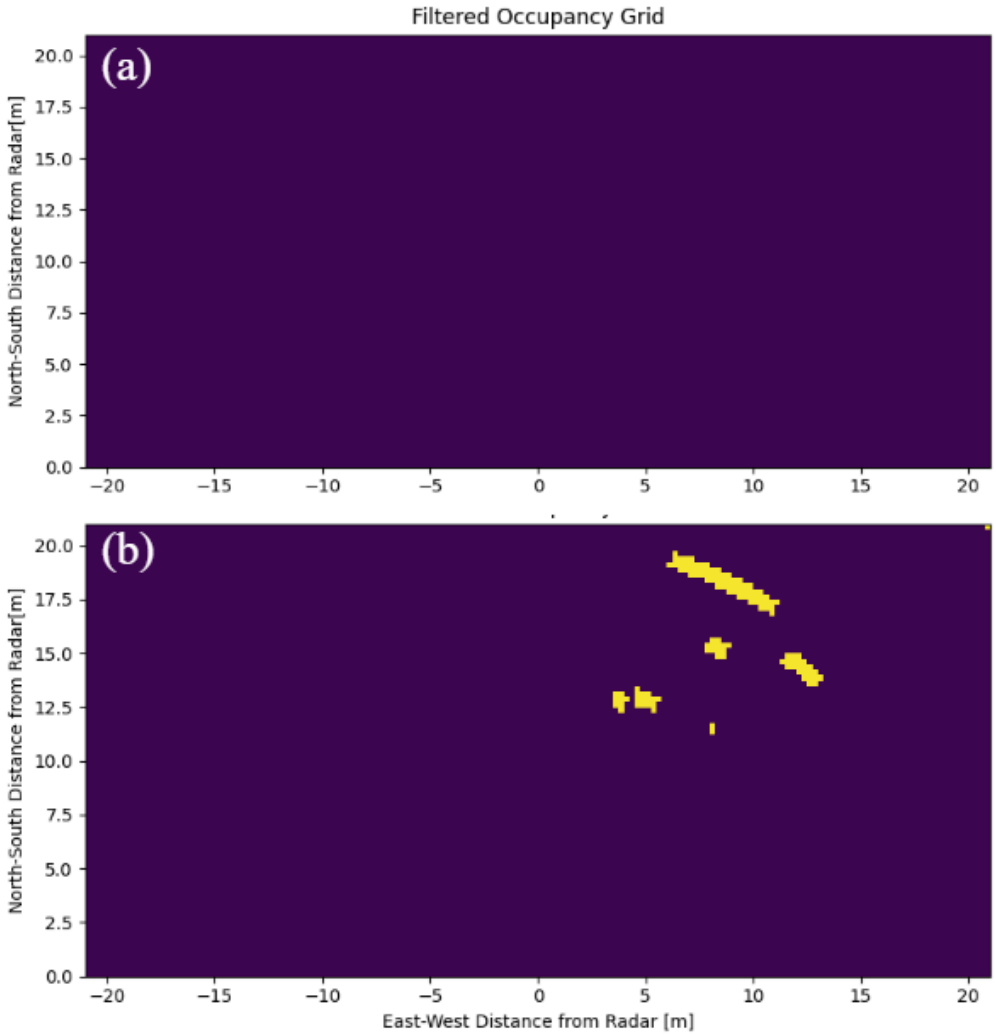

Figure A8. Occupancy grids of open field baseline test in the (a) static radar and (b) turning radar trials. Empty space from 0 m to 10 m in front of the radar is confirmed in both trials. Fixed environment obstacles representing Parsons building are seen in (b).

Next, the results of the trial to compare a person and a trash can of similar size are shown in Fig. A8. By inspection, the cluster of occupied grids are very similar and comparable. The results comparing crouched and standing persons are also visually similar.

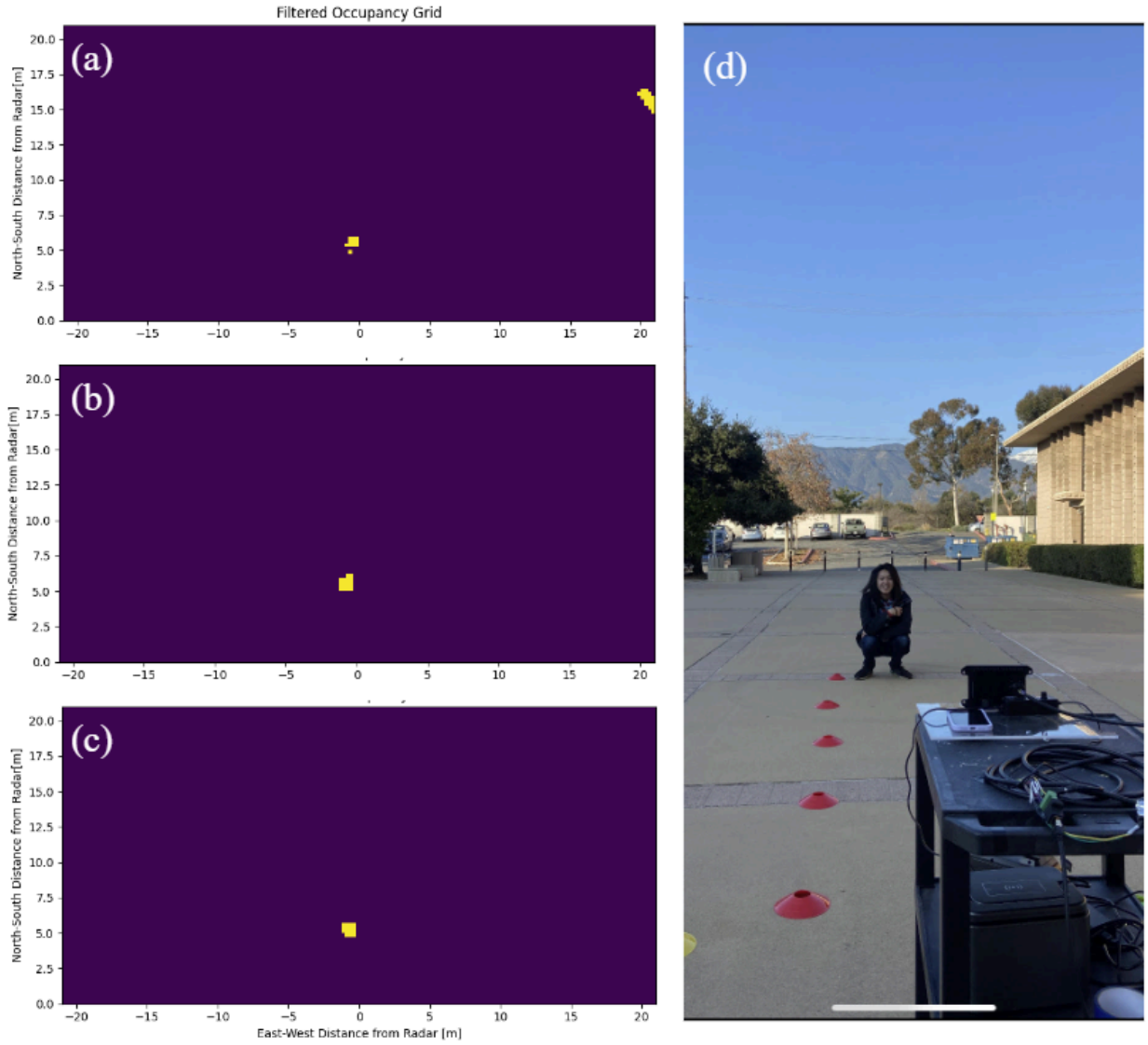

Figure A9. Validity of using a trash can to represent a crouching person for object detection. Occupancy grids for an (a) trash can, (b) standing person, and (c) crouching person each at 5 meters in front of a static radar. (d) Photo capture crouching person in front of radar.

Quantifying Performance

Recall that the objective of this project is to accurately represent the positions of objects near a robot. To evaluate the performance of the experiments conducted, two metrics for accuracy were used.

The first metric ε_A was the error in the area outlined in (4). In Fig. A9, the red outline indicates the shape of the grid cells clustered that the occupancy grid determined was a trash can; the green box is the ground truth location of the trash can.

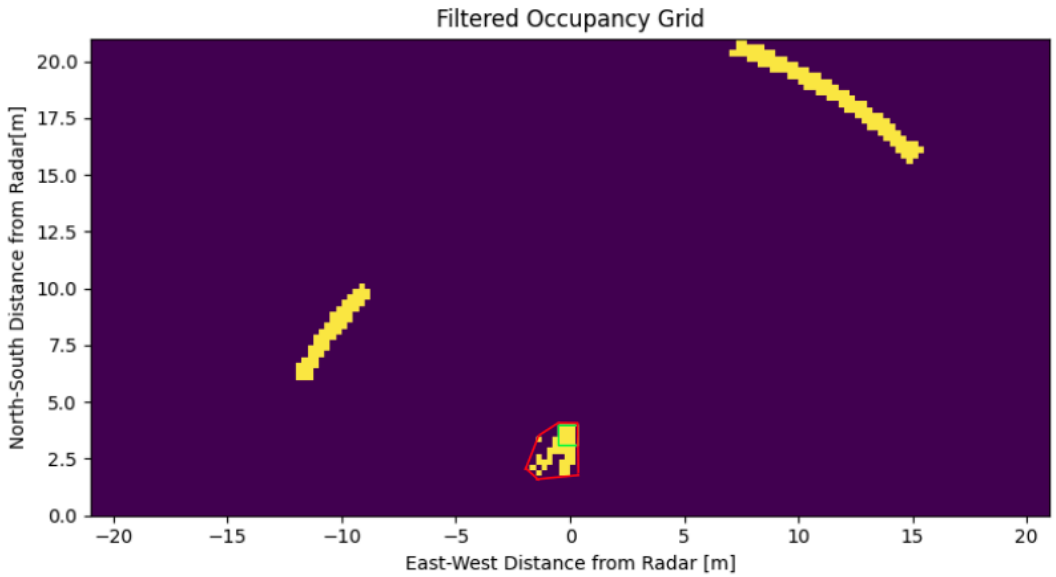

Figure A10. Annotated occupancy grid for trial with trash can 2 m away from a rotating radar. The green square marks the occupancy grid cells representing the ground truth location of the trash can. The red polygon marks the cluster of occupied grid cells used to evaluate the performance of obstacle detection.

With the trash can approximately 0.5 m in diameter and square grid cells 0.25 m on each side, the trash can is expected to occupy four cells in the occupancy grid. The x_g position of the four-cell center is constant at 0 across all trials. The y_g position changes from 2, 3, 4, 5, 6 depending on the trial and placement of the trash can object. The difference between the number of cells of the red n_r and green n_{gt} outline was taken to evaluate the accuracy of the radar at pinpointing the obstacle width.

$$\varepsilon_A = n_r - n_{gt} \tag{4}$$

The second and third metrics ε_x and ε_y were the normalized x_g and y_g error in the ground truth center of the trash can $(\bar{x}_{gt}, \bar{y}_{gt})$ and the center of the area of the field in the occupancy grid via radar scans $(\bar{x}_r, \bar{y}_r)$, expressed in (5) and (6). The trash can was placed at a fixed distance from the radar and the x and y distances were recorded as the ground truth. Again, in Fig. A9, the

center cell of the red outline was determined as the x and y distances recorded by the occupancy grid via radar scans.

$$\varepsilon_x = \left|\overline{x_r} - \overline{x_{gt}}\right| \quad (5)$$

$$\varepsilon_y = \left|\overline{y_r} - \overline{y_{gt}}\right| \quad (6)$$

Performance Plots

The values of ε_A are plotted in Fig. A10 across the trials. The values of ε_x and ε_y are plotted in Fig. A11 across the trials.

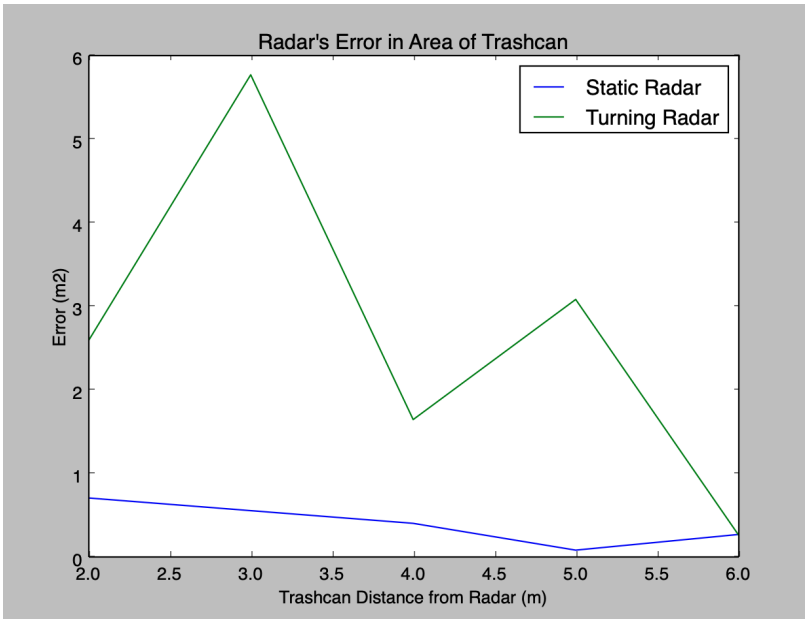

Figure A11. Error in the area of the radar's detection of the trash can as a function of trashcan distance from the radar. Green curve represents turning radar scans, and blue curve represents stationary radar scans

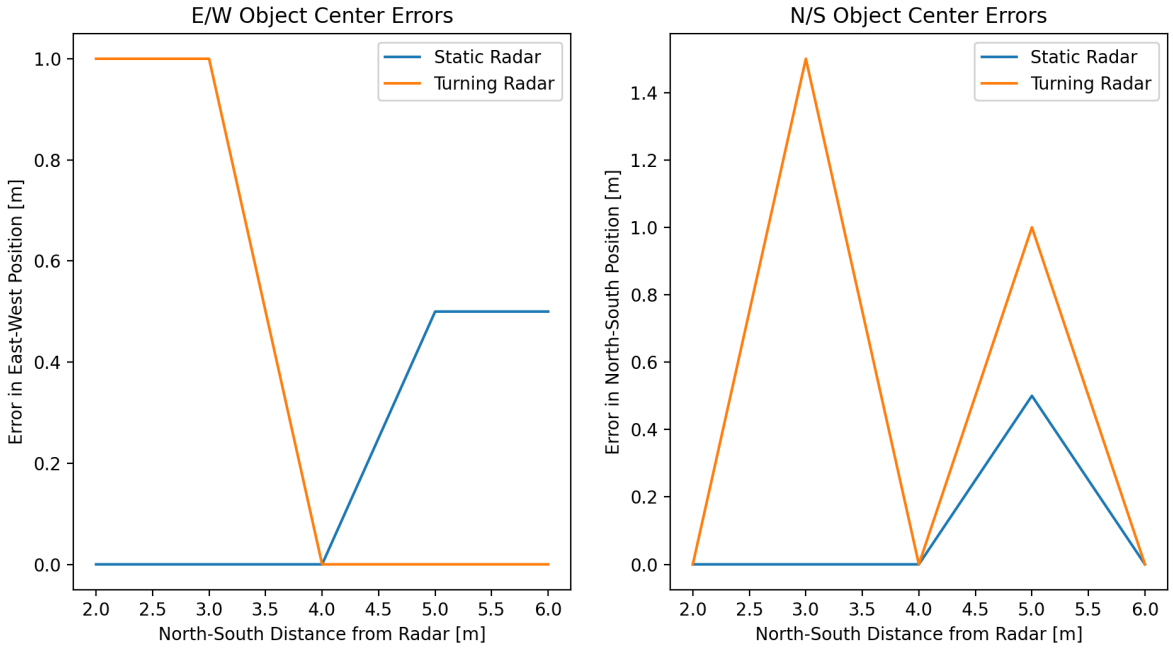

Figure A12. (Left) Error in the East/West or x_g location or (Right) Error in the North/South or y_g location of the trash can's center as a function of trash distance from the radar in the North-South direction. Orange curves represent turning radar scans, and blue curves represent stationary radar scans.

Fig. A12 shows a significant difference between the estimated and actual area of the trash can. Especially while turning, the occupancy grid produced significant errors in estimating the area the obstacle is occupying on the grid map. The error for the turning radar appeared highest when the trash can was near the radar at about 3 meters and decreased at ranges beyond that. However, if the radar is static, the error in estimating the area of occupation is nicely constrained below 1 m^2. Therefore, the occupancy grid was able to accurately map out the environment with the case of a static radar but struggles to do so while turning.

Fig. A12 shows errors of object centers below 1.5 meters or 6 occupancy grids in either direction, with slightly higher accuracy in east to west position estimation. In both directions, the maximum error of the turning radar is about double the maximum error of the static radar.

One possible explanation for the greater errors in ε_A is the stationary constraint of the radar. When all objects can only be viewed from one position, this forms a monocular vision that is geometrically limited in depth perception. If the radar can observe an object from multiple perspectives, unoccupied space behind the targets can be observed and refine the area estimation of targets.

Conclusions

The system successfully uses the O-79 Ainstein radar to generate a visual occupancy grid map that captures the environment around a robot. The objects detected in the occupancy grid are near where they were placed in front of the radar, with position accuracy below 6 occupancy grids. Furthermore, the occupied regions represented in the occupancy map correspond to real-life objects in the path of the radar's beam that are larger than a crouching person, while smaller objects like colored cones 4 inches tall are considered unoccupied space. In this sense, this prototype solution satisfies all design functions and constraints while pursuing its objective.

Basic yaw heading odometry provided by a smartphone shows that measurements of a rotating radar can be transformed and applied to a fixed global occupancy grid. Further work should involve adding two more degrees of freedom: translation in the 2D plane, which changes the radar's position away from (0,0) in the global frame.

Another simplifying assumption to drop in future work is the static obstacle assumption. Dynamic obstacles with nonzero relative velocities to the radar will require a different prediction step in the Bayes filter, each cell probability should be extended from binary to a continuous probability.

In addition, when the robot or radar is no longer stationary and uncertain in position or pose, the team may explore SLAM algorithms and incorporate GPS latitude and longitude data along with the occupancy grid maps to improve state estimation on the environment and pose of the autonomous lawnmower. While this project makes significant progress in mapping, there is much more work to be done to ensure the safe navigation of Doosan Bobcat robots.

Appendix B: Path Planning Algorithm

The algorithm is implemented using two Python packages: shapely and dubins. Shapely simplifies all point, line, and shape objects and manipulation, along with being consistent with the Shapefile data standard. This library is used everywhere in the algorithm. Dubins is a library that helps make dubins paths, which are the shortest paths between two robot poses given a fixed turn radius. This library is used for all the U-turns.

I. Import the mow area and obstacle areas.

> Use the shapely polygon, which is a data structure that can capture both mow area and obstacle area. The `exterior` field holds a list of latitude and longitude points that mark the mow area boundary. The `holes` field holds a list of lists of latitude and longitude marking the boundary of internal areas to avoid (obstacles).

II. A user can mow in any angle relative to the North.

To simplify the problem, so that the mow lines go up and down along the y axis, we rotate the object by the angle. Rotations can be used by shapley's affine transforms. Equirectangular projections were first used to change the coordinate frame into meters.

III. Pad the mow area polygon

During U-turns, the mower crosses the boundary to complete the turn, so the polygon is padded with Shapely buffers to avoid this. Future work should fine tune boundaries to never cross the obstacles.

IV. Decompose padded mow area into cells.

We can break down an area into smaller polygons without any internal obstacles, as shown in Fig. 7. The smaller polygons are called “cells”, which simplifies adding paths. Bigger and fewer cells are better for efficiency. Cells are created through bounding boxes in Shapely and then are decomposed through the following process:

- Iteratively put vertical lines from left to right of the bounded area
- Detect any intersections with obstacles
- Delete intersected lines, leaving segments, to generate the smaller cells

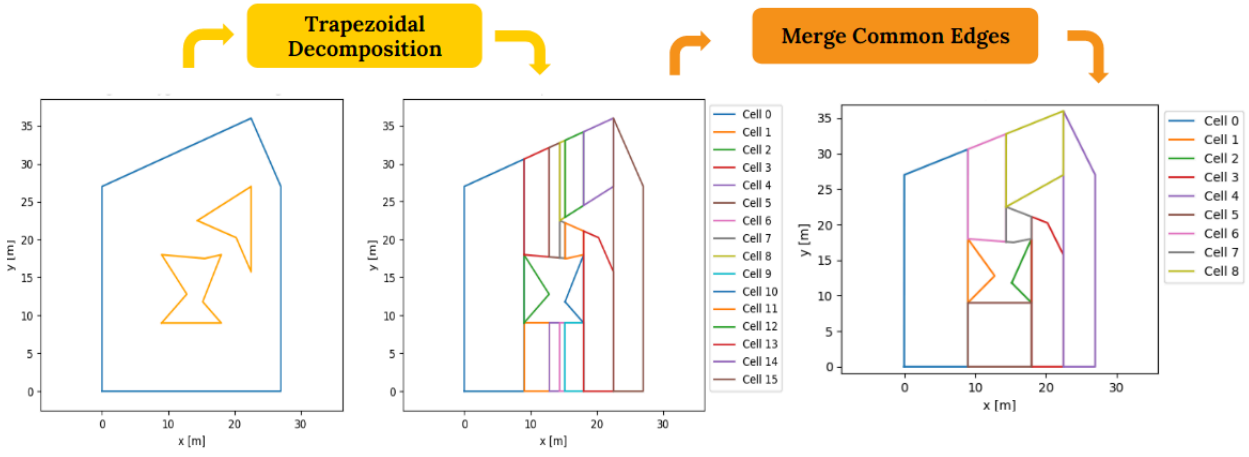

Figure A13. 2D boundary of path planning workspace in blue. Obstacle boundaries are specified in orange. Two steps in cell decomposition. Merging common edges minimizes the number of cells, which is more efficient by reducing computation in Step VI.

V. Create paths within the cells.

The optimal path creates S-shaped curves within each cell. This is accomplished through the use of Dubin’s Curves, a mathematical term for the shortest curve between two robot

poses given a fixed turn radius. Essentially, Dubin's Curves allows efficient turning in the path plan. An online python library is used to implement Dubins into the path plan. In addition, proximity of the straight lines to each other is defined by an overlap ratio. Increasing the ratio makes the mow lines closer together and decreasing the ratio makes the lines further apart.

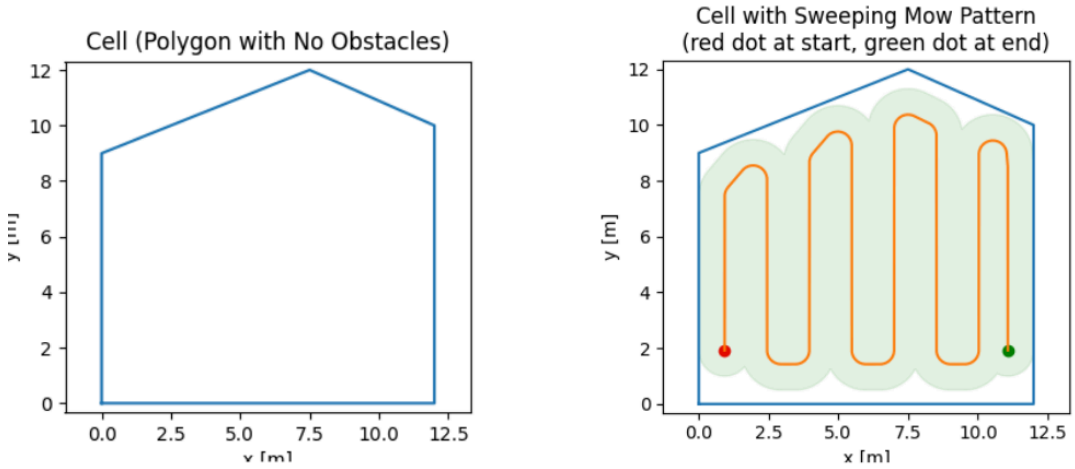

Figure A14. Path within cells

VI. Cell to cell path planning (Future work)

Once the cell has been divided into smaller cells, the path plan must link the paths in each cell together while still avoiding obstacles. Two optimized algorithms that could solve this problem are A* and RRT*[2]. A* is a search-based algorithm that uses discretized occupancy grids and weighted direction heuristics to find the most efficient and optimal path, RRT* (rapidly exploring random tree star) is a sample-based algorithm that extends branches that find another optimized path. It is up to future teams to decide and implement A* or RRT*. The rest of the path plan uses vectorized geometries, so RRT* should integrate easier. However, the discretized grid provided by the mapping module is better suited for A*. It could be vectorized with an algorithm like alpha shapes, making RRT* more viable.

VII. Optimize cell order (Traveling Salesman Problem)

The next step is to optimally visit each cell. This is commonly known as the Traveling Salesman Problem (TSP). To do so, there are several approaches and algorithms.

1. <u>The naive approximate solution:</u>

This approach would use Shapely to get the centroid of each cell and use a nearest neighbor algorithm to find the minimum distance path that visits each cell at least once.

2. <u>The better approximate solution:</u>
   This solution would navigate around obstacles more efficiently by using a cost function. The cost function would weight paths that included obstacles such that obstacle-free paths would be prioritized.

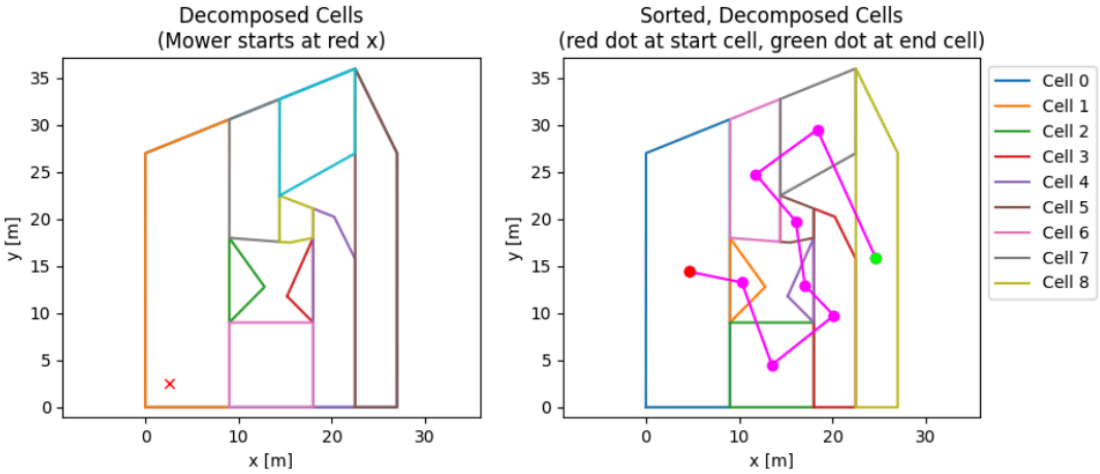

Figure A15. Ordering of decomposed cells

3. <u>The optimized solution:</u>
   This solution would use an algorithm like A* or RRT* on the corners of each cell, which would be the potential starting and ending points for each cell. This solution may be computationally costly, and would be reserved for future work.

VIII. Assemble final path

The bounded area was initially rotated to keep the mow line direction vertical. Thus, the final step of this process is to rotate the cells back into their original positions, rotating the path lines along with it.