EECS 464 Project 1

Purple Team: Madi DeVore, David Marsh, Matt Maurer, Christian Soto, Ragav Subramanian

April 12, 2020

Background

Project Tasks

Our task goals and requirements were listed in the task specifications for Winter 2020 EECS 464 Project 1. Our task was to design and build a physical and simulated robot that could traverse an arbitrarily sized track with arbitrarily placed waypoints while maintaining the orientation of the direction of a mounted laser. In addition, the robot's movement to and between the waypoints were to be achieved with and without human interference. Due to recent events at the time this project was attempted, any goals and requirements involving the physical robot were considered omitted from the task specifications. The pass-fail criteria for meeting the task goal were as follows:

- Robot stays in the arena, with robot tag within 1m vertically of the floor
- Robot reaches the final waypoint within 15 minutes
- Laser never illuminates in a direction with a negative Y component

Arena Layout

The arena in which the robot operates is a 2m x 2m square delineated by 8 barcodes or tags as they're seen in the virtual modelling. All tags are square and the same size. The waypoints are also modelled using tags and the location of the robot is tracked using its own tag. In order to successfully reach a waypoint the robot's tag must be within a tag's radius of the waypoint tag. Since we switched to modelling the arena virtually the physical dimensions are arbitrary as we now use x and y coordinates.

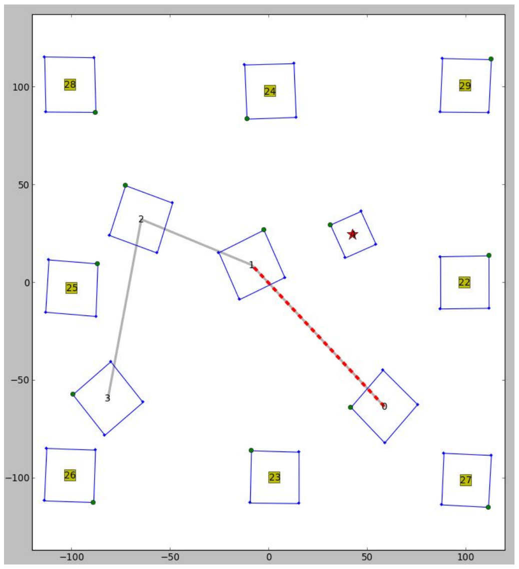

Figure 1: Course Layout

Research and Design Process

Before beginning the construction of the robot, we considered multiple design alternatives. This section describes the three alternatives we showcased in our brainstorming presentation.

Mecanum Wheeled Design 1

Design 1 used mecanum wheels similar to 2019 Red Team. We considered using three pairs of mecanum wheels and continuous rotation (CR) modules attached to the bottom of our proposed triangle base. A mecanum wheel, shown in Figure 2, is a wheel that has cylindrically-shaped rollers mounted around its circumference at a 45 degree angle to the axis of rotation. A mecanum wheel has independent motion relative to the others in the system, and the combination of all these different motions can be vectored into cardinal directions relative to the system, shown in Figure 3. A more detailed explanation can be found at Wikipedia's Mecanum Wheel, and a YouTube demo can be found here.

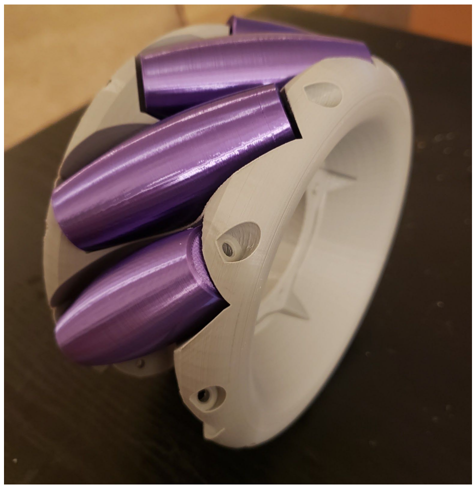

Figure 2: 3D Printed 2019 Red Team Mecanum Wheel

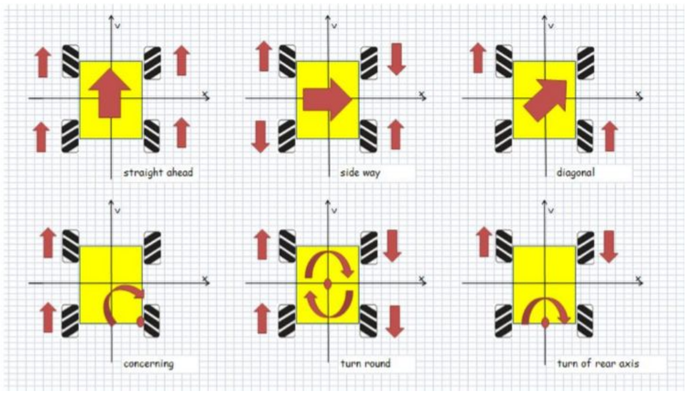

Figure 3: Mecanum System Movement

Wheeled Design 2

We considered attaching a U-bracket module to the end of a CR module. A drive wheel attached to the CR module could be lifted off the ground using the U-bracket module, shown in Figure 4. If we had two pairs of motors in this configuration, mounted orthogonal to each other, we could drive anywhere in the plane without rotating the robot. We planned on putting ball castors, shown in Figure 5, on the bottom of the robot so that the base supporting the motor modules could glide in both directions.

We did not end up pursuing castors on the base because we were concerned that the robot would turn on the castors. However we did use the CR module mounted on the end of the U-bracket module for our final design.

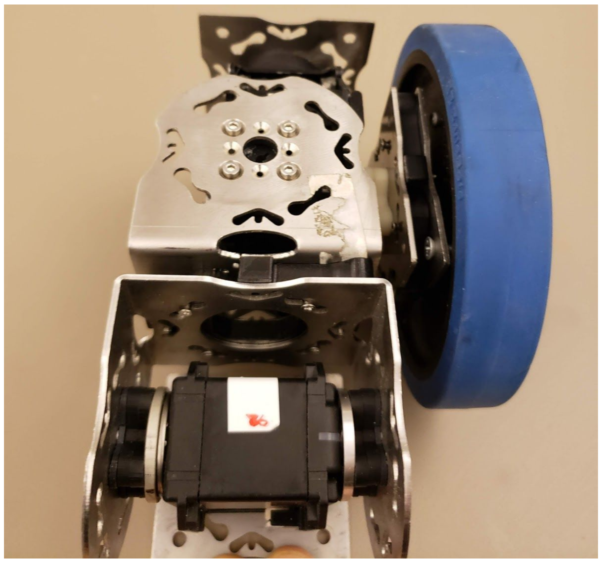

Figure 4: CR module attached to U module and drive wheel

Figure 5: Ball Castor

Moving Turret

This Moving Turret Design was an experimental idea that was not based off of any previous team's design. The basis of the design was to take a lot of the heavy work off of any software control system for maintaining the turret consistently in the +x-direction. The robot was designed to have a rod that is freely held using bearings through the center of the robot chassis. This central rod would hold the turret on top and have a castor wheel on the bottom. This pivotal rod would allow the turret to remain focused in +x-direction as we pivot the robot to turn, the rod would not rotate due to the low-friction skateball bearings that would connect the central rod to the chassis. However, we did not pursue this idea as it was quite experimental and if the frictionless effect of the bearings did not hold up, the mechanical design would be for nothing.

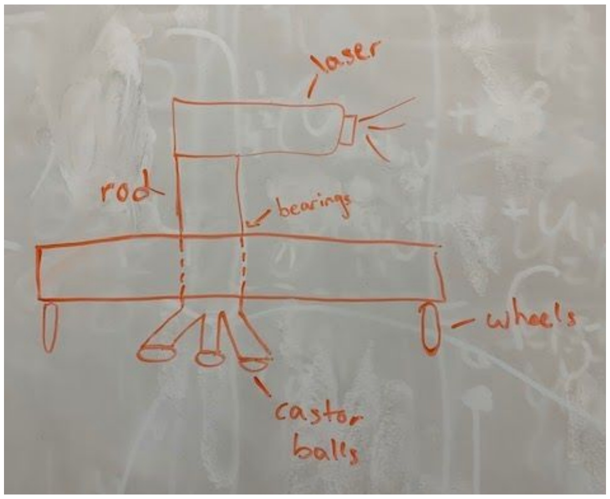

Figure 6: Drawn image of Moving Turret design

Final Design

The project moved to a simulation after completing demos 1 and 2 due to the cancelation of in-person lectures.

Hardware

Our final design was essentially two tricycles in orthogonal directions. By lowering one of the two U-bracket modules, we could tip the base of the robot until only the three wheels oriented in that direction were touching the ground. Then, the robot moved in a straight line by rotating a CR module attached to the end of the U-bracket module.

The robot was able to move continuously in +- X and +- Y with minimal rotation in the z direction. The physical robot was abandoned before we were able to gather quantitative data.

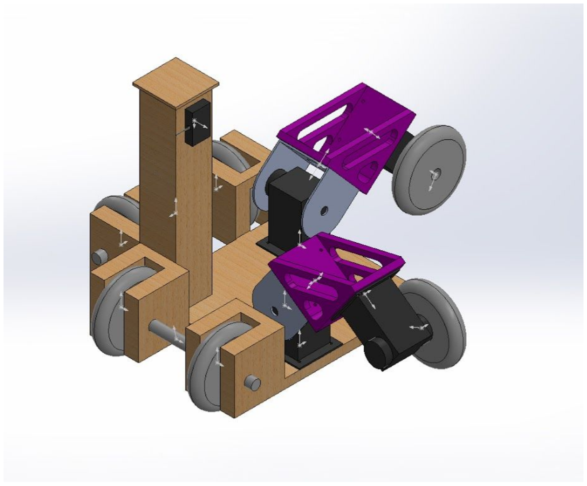

Figure 7: Image from SolidWorks of the final design

Software

The software we created can be found in the Purple Team How-To Directory. We chose to use a Python program to control the simulation, utilizing the robotSimIX library provided and a scorpionSimIX file of our own. Our control program mimicked the translational motion of the mobot we designed allowing it to go left, right, forwards, and backwards without turning. When in manual mode the program created an event handling system to handle keyboard presses as shown in Table 1 so that it would move when the corresponding buttons were pressed. Upon pressing 'A' the software would start the autonomous plan which would calculate the distance from the first waypoint to the next and attempt to move that distance in the x and y directions. If the waypoint was missed then it would spiral from where the simulated robot moved to until it reached the next waypoint. The worst case of the spiral feature would be to extend the spiral to spanning the entire arena.

Key	Left Arrow	Right Arrow	Up Arrow	Down Arrow	'A'
Simulated Action	Move left	Move right	Move forward	Move backwards	Start autonomous mode

Table 1: Indicates simulated action (row2) that occurs after a specific button press (row1)

P-Day Results

P-day consisted of 10 different arenas that each team attempted to complete within the time constraints of the class period. This averaged to about 10 minutes per arena. Each arena had a different seed that would randomize the waypoints' locations.

Pass/Fail Requirement

Due to the project moving to a virtual format the pass/fail requirement was changed. The conditions relating to the physical movement of the robot were dropped and now each team needed to reach the final waypoint in under 15 minutes. Each team completed the pass/fail requirement during P-Day and were able to submit multiple runs with completed arenas. Most teams were able to complete all 10 arenas, but Purple and Maize ran into technical difficulties and were only able to submit 5 arenas.

Overview of Teams performance for each arena

Each team had the capability to move their robot manually to the final waypoint. Maize and Red teams chose to manually move their robots and not attempt autonomous mode. Green team attempted autonomous mode but failed and then drove their robot manually to the final waypoint. Purple team attempted autonomous mode but ran into problems with correctly generating the randomized arena. Purple team managed to successfully complete a couple of arenas in autonomous mode but did not have enough time to attempt all arenas. Blue team was the most successful attempting autonomous mode for each arena and managed to hit all of the waypoints. A breakdown of how each team performed in each arena is given in Table 2.

Team	Arena 0	Arena 1	Arena 2	Arena 3	Arena 4	Arena 5	Arena 6	Arena 7	Arena 8	Arena 9
Maize	----	2/3	M	----	----	M	-----	M	M	-----
Red	M	M	M	M	M	M	M	M	M	M
Green	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
Purple	AA	------	AS	AS	AS	-----	AS	-----	-----	------
Blue	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS

Table 2: P-Day results for all teams.
M - Manual mode only
AS - Autonomous mode finished successfully
AA- Autonomous mode attempted but switched to manual

Maize team had problems getting their robot to find the first waypoint, their missing trials corresponded to the runs where they were unable to locate the first waypoint and got stuck. Maize team was able to successfully reach 2 of 3 waypoints in Arena 1. Purple team's missing trials were from running out of time during P-Day and also from running the arenas before realizing that we were incorrectly randomizing the waypoints. The data produced from those runs correspond to the same waypoints given to us to practice with.

Discussions

Overall analysis

Blue team was the only team to successfully use autonomous mode for every arena. Green and Purple also attempted autonomous mode but were either unsuccessful or sometimes successful. Maize and Red used manual mode for each arena.

Individual Team Designs

Purple

Once we fixed the problem with our code not correctly generating new arenas we had successful autonomous trials. Since it took us about half an hour to work out why our code wasn't functioning and about 20 minutes more to debug the problem we did not have enough time to run all the arenas. This bug came up because we created our own class that inherited some of Professor Revzen's code, but it was not updated when new software updates were

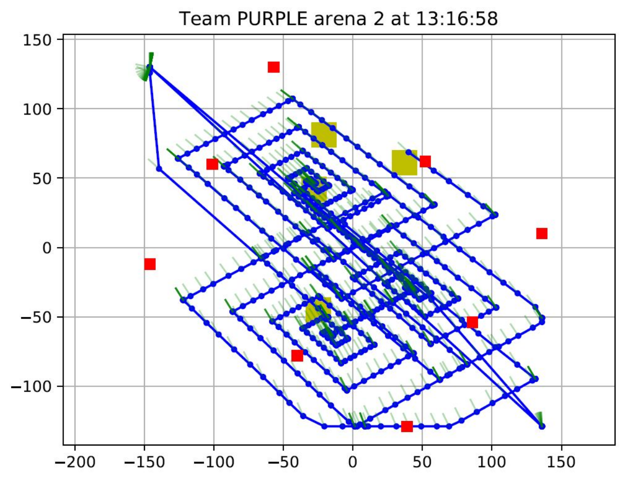

Figure 8: Team Purple arena two

released. In the arenas we did have time to redo we had originally run them without regenerating new waypoints so they all had waypoints corresponding to those of the practice arena. Our simulation moved translationally. Once we drove to the first waypoint we would assume that the waypoint was our starting location and would calculate what y and x distances we would need to travel in order to reach the next waypoint. If the robot missed the waypoint it would spiral until it hit the next waypoint. In the visualization generated for arena 2 we missed the waypoint three times so there are three different spirals.

Blue

Blue team had a working autonomous program. They were able to reach all waypoints in all maps. Their program used a particle filter and attempted to drive directly to each waypoint. In the event that the robot missed the waypoint, it entered a scanning recovery procedure. They were able to finish 5 of the 9 maps directly without using the recovery procedure.

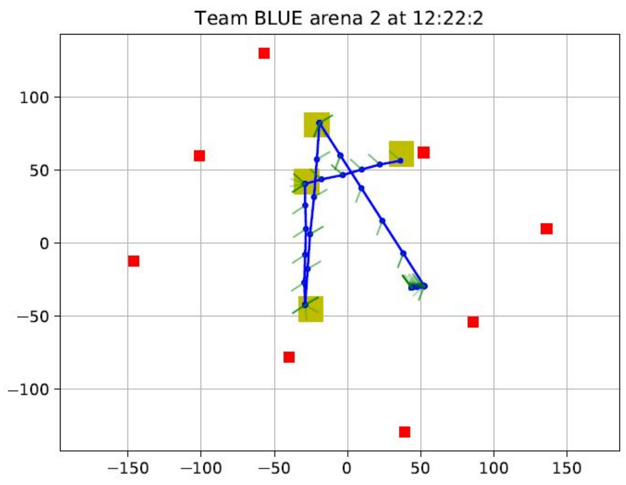

Figure 9: Team Blue arena two

Maize

Maize team had computer hardware issues. They were unable to stream video and run the simulation smoothly at the same time. They drove manually to the waypoints. Their control system relied on needing to constantly reorient the arena. The more their simulated robot rotated the less they were able to control it. They used a spiralling recovery strategy that would swing the “arm” of the robot while driving it forwards and backwards to form a spiral.

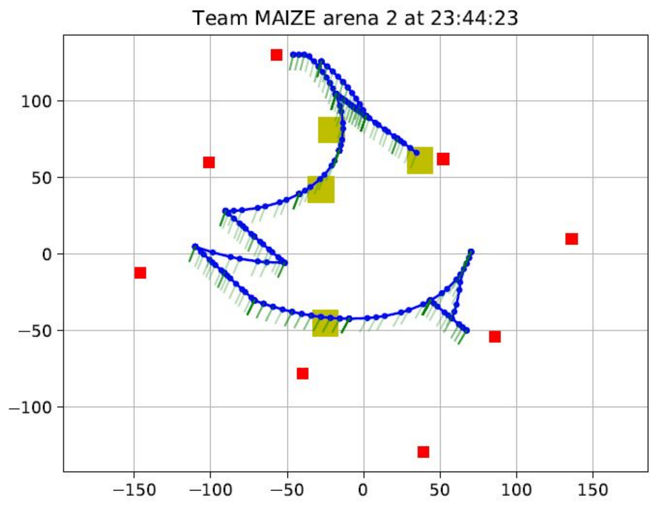

Figure 10: Team Maize arena two

Green

Green team's autonomy did not work. They drove to the waypoints manually. Their attempt at autonomous mode resulted in their robot veering off in the opposite direction of the waypoint and continuing indefinitely. Their strategy for recovering their robot during autonomous mode was to keep a history of their movements and backtrack if it missed the waypoint.

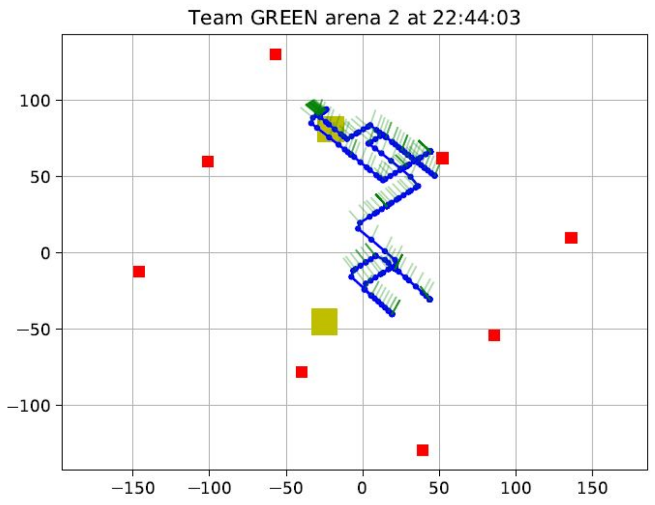

Figure 11: Team Green arena two

Red

Red team had their laser pointer pointing in the wrong direction. They drove to all the waypoints manually. They did not attempt to do autonomous mode and had no planned recovery strategy if their robot was to go off course. Their movement used a continuous position estimation by using a particle filter. Their simulation was able to rotate towards the direction they wanted to move and then advance in a line towards it.

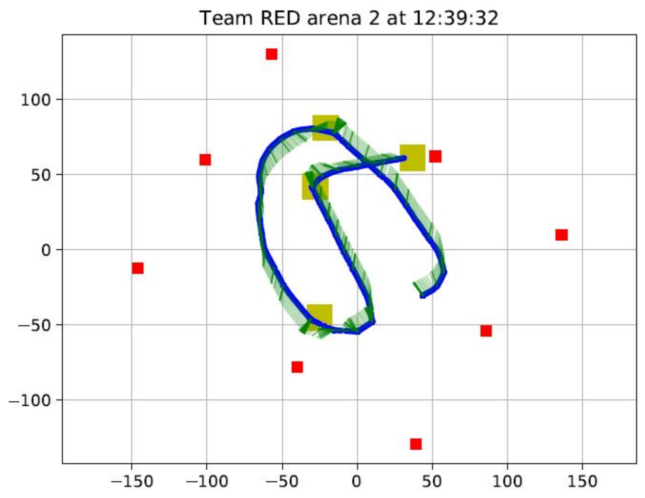

Figure 12: Team Red arena two

Future Improvements

Hardware

The passive wheels on our robot were able to rotate independently from each other because they were both free on the axle. If the passive wheels were fixed to the axle and instead the axle spinned freely it could prevent the robot from rotating.

Software

Update moving between waypoints

Our robot moved in consistent steps when in manual mode, but when run in autonomous mode we had trouble calculating how many steps to use in each direction to reach our goal. Often the robot would overshoot or not move far enough in the x and y directions and would have to resort to spiralling to find the next waypoint. In the future we could improve our software design by making the movement between waypoints more precise and relying less on spiralling in order to reach the next waypoint.

Modify the recovery plan

Our recovery plan would be to just have it spiral until it reached the goal. In order to make this more effective we could add in counters to help us keep track of if we were more likely to over or undershoot or goal. Using this we could either backtrack or take an additional step before we start the spiral so that it wouldn’t need to spiral for very long. Additionally, a smaller step size might help prevent the robot from overshooting waypoints.