CSCI 5552 Project Report

Team: Bharath Sivaram, Isaac Kasahara
Project Option: SLAM With Landmark IDs
Project: https://github.umn.edu/sivar019/CSCI5552-Project

Introduction

The goal in this project was to use camera, lidar, and odometry data to get a robot from a known start point to a goal point. The map used is shown in Figure 1. The blue point shows the goal.

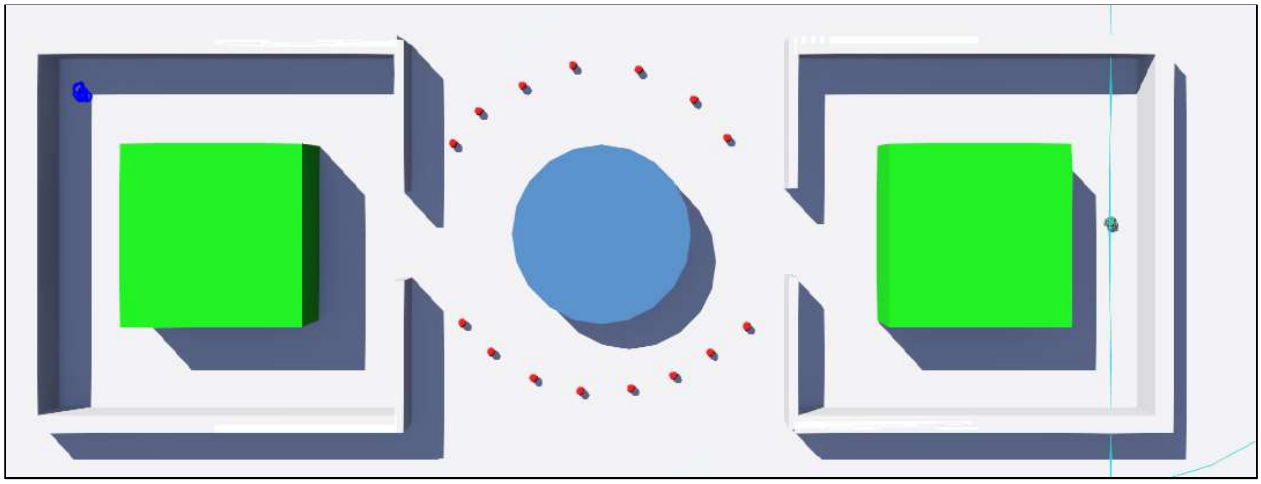

Figure 1. Simulation Map

The following sensors were used when implementing:

- 1 layer LIDAR
- Camera
- Wheel Encoder

The path planning consisted of getting out of the maze, using the red markers to do SLAM, then once again following the maze till the goal was reached. The algorithm was split into 4 modes, each of which will be described in this report. The first 3 modes were motivated by the bug algorithm.

1. Head to Goal
2. Wall Alignment
3. Wall Following
4. Marker SLAM

Mode 1: Head to Goal

At the start, we know our global pose and the goal point. So we begin by first calculating the angle to the goal given our position using the following equation:

$$\theta_g = arctan2\left(\frac{y_{goal} - y_{rob}}{x_{goal} - x_{rob}}\right) mod\ 2\pi$$

This allows us to get an angle between 0 and 2π. Additionally, the robot pose was calculated using differences in wheel encoder values and using the equations shown below:

$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = p + \begin{bmatrix} \Delta s \cos(\theta + \Delta\theta/2) \\ \Delta s \sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\theta + \Delta\theta/2) \\ \Delta s \sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix}. \quad (5.6)$$

Pose update using wheel encoder [1]

Where Δs is the average difference in distance traveled by both wheels, and Δθ is the angle change using the distance changes and axle length. This portion was by far the most challenging, as the wheel encoder noise would gradually accumulate. By the time the robot makes it out of the maze, the position would be very inaccurate going into Marker SLAM mode. Ultimately, the final video required getting raw velocity values and using them to propagate the position of the robot every time step, since the encoder would result in the wrong final position.

Once the robot pose is calculated, the robot rotates until it is in line with the goal, this uses the difference between the goal angle and robot current angle. And once aligned the robot goes forward until an obstacle is encountered. Given that the environment is known, we define a stopping point as when there is a wall within a certain distance of the robot AND the robot does not have a clear path forward. This ensures it doesn't stop when a wall is nearby on the side.

In order to detect the walls, we extracted the data from the LIDAR sensor for each timestep. The LIDAR data gives us the angle and distance of each point in the LIDAR's range. To allow for the robot to robustly understand the environment around it, we perform Hough Line Transform on the LIDAR data, so the robot can tell exactly where the walls are. Our Hough Line Transform works by looping through all of the measured data points and then for each point loop through all potential angles (θ) of the line that the point corresponds to. We can then calculate the corresponding distance (ρ) to each of those lines and increment that value in an array that stores the frequency of rho theta pairs. The most frequent rho theta pairs are then used as the lines the robot detects. Figure 2 shows an example of a situation where the robot's lidar detects 3 walls, as well as shows the frequencies of the corresponding rho theta pairs.

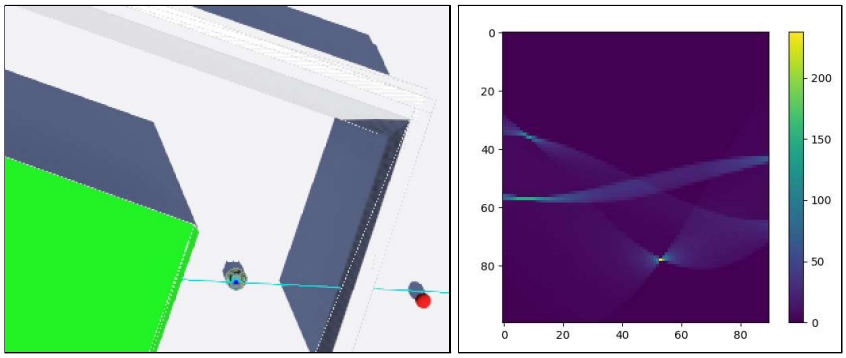

Figure 2. Hough Line Transform

Hough Line Transform **[2]** worked well to detect walls, but would sometimes return non-existent lines or multiple results for the same line. The scipy.cluster.hierarchy was used to get these line segments classified into the same line using a threshold of 0.05. The algorithm also checked for 0 distance lines and removed them to retain detection accuracy. Each wall was then identified by the centroid of its respective cluster.

Mode 2: Wall Turn Mode

Once a wall is detected, the ρ and θ are saved. The robot then turns in place until it is parallel with the wall. A challenge here was making sure the robot turns towards the side with more free space so it does not get stuck. For example, in Figure 3, we would like the robot to turn counter-clockwise.

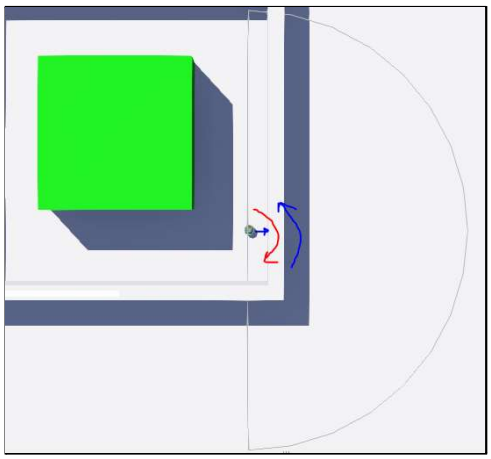

Figure 3. Turn Direction Planning

This was achieved by taking the mean of each end of the lidar and seeing which was larger. The robot would then turn in that direction until it is parallel with the wall, then move onto Mode 3.

Mode 3: Wall Forward

Once the robot is parallel, it follows the wall by traveling in a straight path and keeping track of the wall centroids. In particular, it keeps track of the distance of the closest wall. Should the difference between that distance and the original wall become larger than 0.05, we assume we have passed the wall. The challenge in this portion is the Hough Transform would not be able to classify the last portion of a wall, and would think it has passed early. This was fixed by adding a buffer of 40 timesteps for the robot to continue going forward.

Once the robot passed the wall, our path planning algorithm had it go back into Mode 1. This is because once the robot is past the current object, we want it to start heading in the direction of the goal again, until it runs into another object. Once back in Mode 1, it will enter Mode 2 or Wall Turn Mode again if it runs into another wall, and repeat this process until it reaches the goal. The only other mode is when it reaches the open area in the middle of the map. Here it is

challenging as the robot has a large cylindrical object to navigate around, which proves challenging for our wall detection. To solve this problem we introduce our final mode, Mode 4.

Mode 4: Marker SLAM

Mode 4 utilizes the camera on the robot to both localize the robot, and plan the robot's path. We activate Mode 4 once the robot's camera detects at least 3 red spheres. In order to perform SLAM, we need to know the positions of each of the spheres. To do this, we get the bearing of each sphere, and compute the distance to that sphere using the lidar information. We can then compute the relative positions of each of the spheres using the bearing,distance, and world to robot rotation to get its global position, as shown in Equation 3. Figure 3 shows the robot obtaining the bearing and distance using the camera and lidar of the spheres. Using the relative positions and ID's of each sphere, we perform Extended Kalman Filtering to update the estimated position of the robot, similar to homework 3, but this time we manually calculate the spheres locations and only update using the spheres the robot can both see and get a lidar measurement from. Extended Kalman Filtering allows us to keep an accurate estimate of the position of the robot in the wide open space, which is important for our path planning.

$$p' = R * p + T$$

Where p is the relative position of landmark, T is current position of robot, and R is rotation matrix from global to robot frame using the current robot angle.

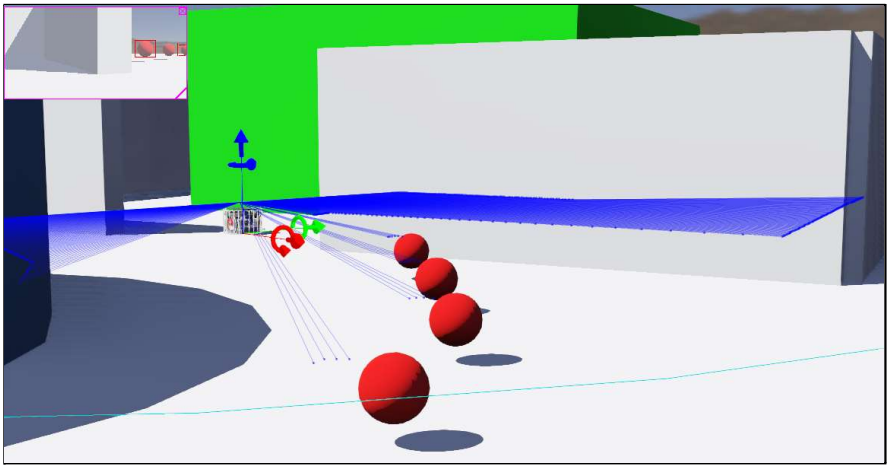

Figure 3. EKF Marker Measurements

To navigate in this open area, we utilize the fact that the spheres are placed in a similar direction we want the robot to take. We program the robot to head towards the furthest sphere that it can see with its camera and is also within its lidar range. We implement a proportional controller for this step, to allow for the robot to smoothly navigate towards the sphere. As the robot slowly moves and turns towards the furthest sphere, it rotates and this allows for the next sphere in the path to come into view, which changes the robot's goal to that sphere. By constantly having the robot aim for the furthest sphere, it naturally follows a wide arc around the center cylinder. Once

there is only one sphere left in the robot's camera, we have the robot change back to Mode 1, to complete the rest of the trip to the goal.

Final Path and Conclusion

After the robot successfully goes around the center cylinder, it returns to Mode 1 and proceeds to navigate to the final goal location by avoiding the rest of the walls and obstacles. The final path can be seen in Figure 4.

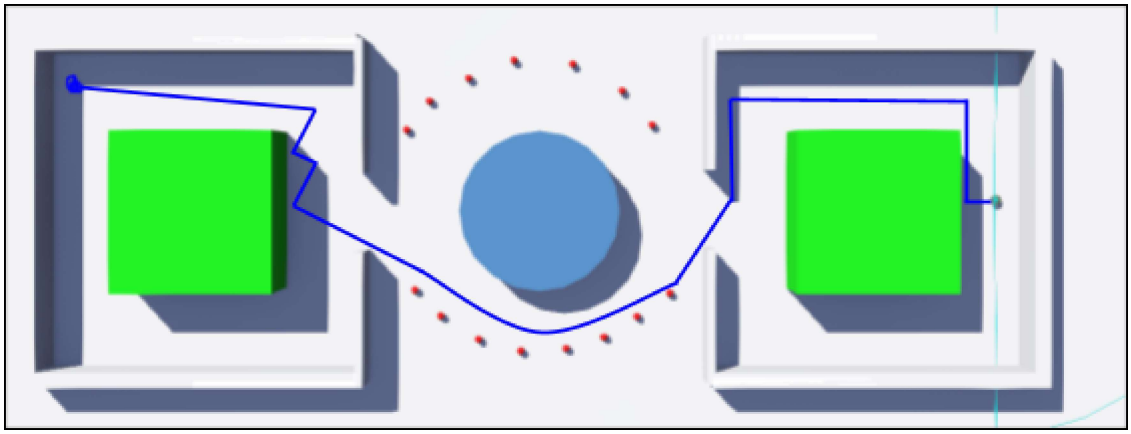

Figure 4. Final Path Taken By Robot

In conclusion, we implemented EKF to estimate the position of the robot, Hough Line Transform to detect objects in the robots path, a proportional controller to allow for smooth movement, and an overarching pathing algorithm to allow for the robot to reach the final goal. We ran into challenges with each of these algorithms, as well as issues with the robot clipping into the ground and simulation physics glitches. Overall we successfully overcame these challenges to create a robot that can navigate to a goal on the other side of the environment.

Future Work

To improve on this project and increase robustness, one method would be to use a spinning Lidar to get more information about the world before making a decision. That way, the robot can save time by turning and running into an obstacle. Additionally, a more accurate wheel encoder would be valuable since the present one has too much noise accumulation to be useful. Though SLAM could be used from the start, in our opinion, wall-following is ideal for a maze and for that reason an encoder would be the best way to keep track of position for that portion.

The camera in this case is only used for EKF SLAM, but it could also be used while getting out of the maze by identifying corners in the image. The robot can then head towards the corners rather than having to use the Hough line transform which can be noisy and inaccurate in certain situations. The Harris corner detection would be a good way of doing this.

Sources:

[1] R. Siegwart and I. R. Nourbakhsh. Introduction to Autonomous Mobile Robots. MIT Press, 2004

[2] Lee, S. (2022, January 30). Lines detection with Hough Transform. Medium. Retrieved May 8, 2022, from https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549