MAE 4180: Autonomous Mobile Robots
Final Competition Report

David van Wijk, dev37

Sibley School of Mechanical and Aerospace Engineering, Cornell University

Professor H. Kress-Gazit

Spring 2021

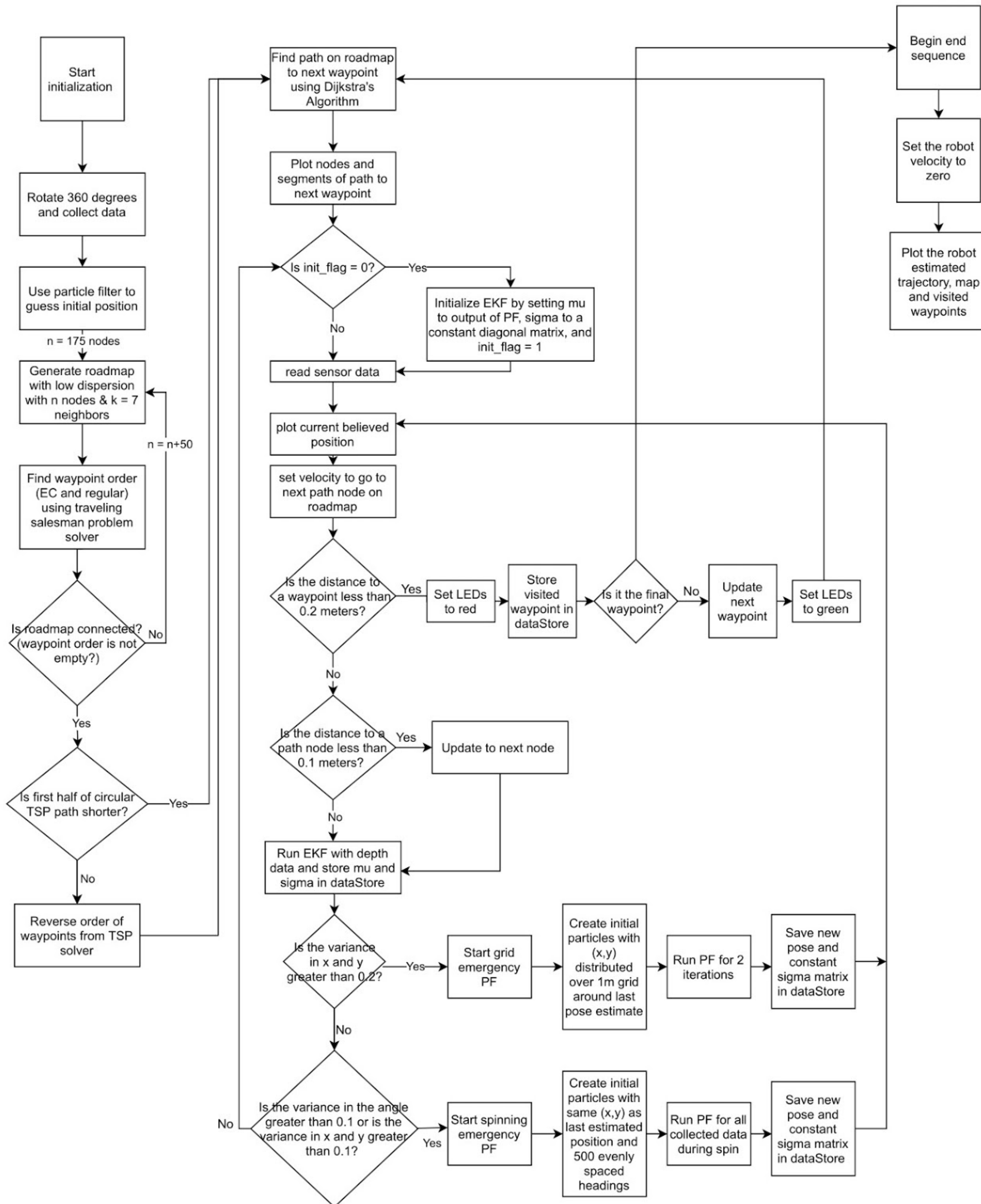
Part 1: Overview of Team Approach

For initial localization of the robot, we rotated the robot 360 degrees and utilized a Particle Filter (PF) along with the data collected in that spin. We chose to use a PF for the initial localization because it was known that the robot would be starting on a waypoint, and thus the initial particle set could consist of particles on the waypoints with varying headings. While navigating, we used an Extended Kalman Filter (EKF) with only depth data. We chose to use an EKF because when the initial location guess is good, an EKF is almost always able to localize the robot, and we can also quantify the uncertainty in the EKF's pose estimate using the sigma matrix. When the EKF's pose estimate reached an unacceptable uncertainty level, we chose to re-localize the robot using one of two possible "emergency" Particle Filter re-localization techniques. The first emergency PF has a particle set of evenly spaced points within a 1 meter grid around the most recent pose estimate with varying headings. We found this technique to be useful in re-localizing the robot when the covariance in x and y was very large. For situations with slightly smaller x and y covariance, or where the covariance in theta exceeded a threshold, we used a similar technique to the initial PF localization algorithm, in which the robot would spin and collect data, and run a PF with the initial particle set located on the most recent pose guess, with varying headings. For motion planning, the first assumption we made was to give extra credit (EC) and normal waypoints equal weight. The reason for this was that we were confident in our localization techniques, and did not feel that the risk of getting lost in a more challenging part of the map (where EC waypoints were usually located) outweighed the bonus points received from EC waypoints. We used a k nearest neighbor PRM with low dispersion sampling, because we would only have to generate the roadmap once at the beginning of the script, and the roadmap produced using this sampling method is quite sparse, meaning that we do not require many nodes to cover the entire map. Additionally, it only requires running dijkstra's once at the beginning of each new path to a new waypoint, which is an extremely fast algorithm. To decide the order of waypoints (both EC and normal) we were to travel to, we used a Travelling Salesman Problem (TSP) solver made by Joseph Kirk, making some slight modifications to tailor his script for our application. We used the TSP algorithm because we wanted to achieve the shortest possible total path to all the waypoints.

Part 3: Individual Contribution

The majority of my purely individual contribution to the final competition was in tackling the motion planning challenge. To make the low-dispersion PRM that we generate at the very beginning of our main control script, I modified the buildPRM function that I wrote for HW6. I changed the script to be able to handle maps consisting of an array of line segments rather than a list of obstacle vertices. Additionally, in HW6 I used the built-in MATLAB function polybuffer (Figure 1) in order to account for the robot radius. Since polybuffer rounds the vertices of the polygons, each line segment obstacle would be represented by a very large number of vertices. To increase efficiency, I modified the code to convert the obstacle line segments into rectangles (Figure 2), which reduced the number of obstacle vertices from 182 for each line using polybuffer, to only 4. Post-processing using the same n and k values on the final competition map, I found that the first version of the PRM script would have taken nearly 2 minutes to produce a roadmap whereas the one using rectangular obstacles only took a bit over 4 seconds to generate (Figures 3 and 4). Also, since we had originally been aiming to reach every EC and normal waypoint, I found a script by Joseph Kirk that finds the near-optimal solution to a TSP, and made a script that calls dijkstra's algorithm for every combination of normal and EC waypoints and saves a matrix of distances to feed into the TSP script. This outputted a near optimal waypoint order for the specific roadmap we had generated. However, a few days into integrating and testing, we realized that even if we localized ourselves perfectly during the entire competition, it would be near impossible to reach all of the waypoints. Therefore, since the waypoint order for a TSP is circular, I modified the TSP script to check which half of the journey would be shorter (ie. should we travel to the second waypoint in the list or the second to last waypoint in the list, etc.). By doing this, I tried to ensure that if the robot ended up not being able to travel to all the waypoints, it would hopefully be able to maximize the number of waypoints reached, but in case it was able to reach all the waypoints, it would still do it in a near optimal way. I would estimate that in total, I spent around 35-40 hours integrating, testing, debugging and tuning.

Part 2: Flow Chart of Solution



Part 4: Discussion of Competition Performance

During the first competition run, the code was never able to generate a valid roadmap, and therefore our robot did not travel at all. While tuning the roadmap for the practice maps, we found that even for large n and k values that would work almost every time, there were occasionally moments where a valid PRM could not be generated, meaning that all of the waypoints would not be reachable using this PRM. To account for this problem, we had modified my TSP solver script to check if there was any combination of waypoint connections that did not yield a path. If so, we would know the PRM was not valid, and we increased the number of nodes, n, and generated a new roadmap, repeating this process until a valid PRM was generated. This method allowed us to initialize n to a relatively low value, but also be able to handle possibly more complex maps than the practice maps we had tuned n and k for. However, when tuning our PRM for the practice maps, we had increased the robot radius for the buffer from .2 to .3 meters, to try to increase the number of straight paths and avoid unnecessary turning at nodes near each other. For the final competition map, this posed a problem at the third extra credit waypoint located at (.388, 2.47) because as can be seen in Figure 5, this made that section of the map inaccessible, which caused the PRM to always be unconnected and thus our code continued indefinitely increasing n and producing a new PRM. After our first trial, once the competition map had been uploaded to canvas, I tried to generate a PRM using our script and realized the problem was that we had increased the robot radius buffer. Once we changed a single line of code in our PRM builder function from .3 to .2, the PRM was quickly generated in trial two using 225 nodes.

During the second trial, the robot was able to successfully localize itself, identifying the correct starting waypoint and orientation with the first guess of its pose being (-2.974, 0.01, 6.266), meaning our initial guess for theta was only about .014 radians off. Including the first waypoint, we detected 1 normal waypoint and 4 EC waypoints, for a total of 90 points. The final competition plot can be found in the appendix in Figure 6. The EKF with depth data proved robust enough to ensure accurate localization nearly the entire time. We only had to re-localize using the spinning emergency PF method two times, meaning that for the rest of the second run, our covariance in x and y position and in angular orientation never exceeded 0.1. Even when the covariance exceeded acceptable levels and we had to use the emergency PF, we were able to accurately estimate our pose again. Since the EC waypoints were placed in locations where the EKF would have more difficulty localizing the robot, the performance of our localization methods in visiting the 4 EC waypoints is very good. This is likely due to the fact that in the noise configuration file used during the competition, the depth noise was zero. When I added noise to rsdepth in the configuration file and ran our control code on the final competition map, I found that with a mean of .02 and a standard deviation of .01, our localization techniques still proved robust enough, and the robot was able to visit a total of 4 normal waypoints and 2 EC waypoints (Figure 7). I increased the noise on the depth measurements with the same mean but a standard deviation of .04, and although it visited only 3 normal waypoints and 1 EC waypoint because it had to call the emergency PF multiple times, it still always managed to re-localize (Figure 8). However, I found that for any noise in depth with a standard deviation exceeding .05, the EKF and PF had too much trouble with localization. In this case the robot was constantly calling the emergency PF and was unable to reach even a single waypoint. On a separate note, I believe that the good performance of the EKF may have also partially been due to a small check within the EKF script which removes measurements that are over 3 standard deviations away from the expected measurements. This seemed to help with situations where the robot was partially facing an opening in a wall, or a corner.

To improve our localization even further, and potentially avoid any emergency PF executions, we could have also used beacon data in our EKF. The only two times we had to re-localize was at the second EC waypoint (-4.46, -2.56). When our emergency PF was called, beacon 9 was clearly in view, and thus we would likely have not even needed this re-localization technique if we had incorporated beacon data into our EKF. The emergency PF takes approximately 15 seconds to execute in total, which means that we could have saved about half a minute on this run. This extra time might have made it possible to visit an

additional waypoint, which, if our trial would have counted, could have made the difference between first and second place.

A modification that I would have liked to make to our overall algorithm was to use a different method in choosing which waypoints to visit, and in what order. As mentioned in Part 1, we originally had decided to use a TSP solver because we believed that if our localization was working well, we would be able to reach every normal and EC waypoint. However, given the size of the map, the number of waypoints, and the limit on the maximum travelling velocity of the robot, I believe that it would be near impossible to reach all the waypoints within the time limit. Therefore we might have been able to visit more waypoints if we had chosen a different motion planning algorithm. For example at each waypoint we could have chosen to visit the closest EC or normal waypoint (using dijkstra's algorithm) that we had not yet visited. This approach would still not have guaranteed a better performance or total shorter path if we had gotten unlucky with the waypoint start location. After the competition, when thinking about possible changes that could have been made to the TSP solver, I also realized that there was a bug in the TSP solver that we ran during the competition. In modifying the TSP solver to account for an invalid roadmap, we made modifications to the structure of the script that caused the distance matrix described in Part 2 to be incorrect. The bug caused only the first row in the matrix to be nonzero, with the result that the waypoint order produced by the TSP solver was simply the chronological order of the indexes in the array TotWaypoints (10x2 array of normal and EC waypoints), beginning with the waypoint the robot starts at. For ease of visualization of TotWaypoint indices, please see Figure 9. To clarify, for the competition map, starting at the 5th normal waypoint, the waypoint order produced was [5; 6; 7; 8; 9; 10; 1; 2; 3; 4]. This means that the order the waypoints will be visited will begin at the 5th normal waypoint, then the 6th normal waypoint, then the first EC waypoint, etc. The reason that we still performed fairly well despite the travelling salesman problem not actually being applied at all to our motion planning was mostly luck with the initial starting waypoint. It turns out that at least the first half of the path produced by a chronological waypoint order starting at 5 is not excessively inefficient. However if we had started at the sixth normal waypoint, the order of waypoints would have been [6; 7; 8; 9; 10; 1; 2; 3; 4; 5], which would have caused the robot to travel unnecessarily across the map. Using the TSP solver with the bug starting at normal waypoint number 6, the path in Figure 10 was produced, in which the robot is only able to reach 1 normal waypoint and 2 EC waypoints, and we can see that it practically passes through the 5th normal waypoint, but doesn't detect it because according to the order produced, it will visit this waypoint at the very end.

Once I fixed this bug, so that the TSP solver actually produced a waypoint order that would try to minimize the total distance travelled to reach all of the waypoints, the robot was able to visit 4 normal waypoints and 2 EC waypoints (Figure 11). Indeed, the robot was able to reach more waypoints with this run, but actually would have received a lower score than the second run in the final competition. As mentioned earlier, we were clearly quite fortunate with the initial waypoint selection in the second trial. Since the fixed version of the TSP solver still would not have earned enough points to win the competition, I think that again, the better motion planning algorithm may have actually been simply to travel to the closest normal or EC waypoint not yet visited. We could also have tried to set up a distance weighting factor based on if a waypoint was an extra credit waypoint or a normal waypoint. To slightly prioritize extra credit waypoints for the point bonus, a factor greater than 1 could have been applied to distances from one normal waypoint to another. This would likely have required tuning though.

Appendix

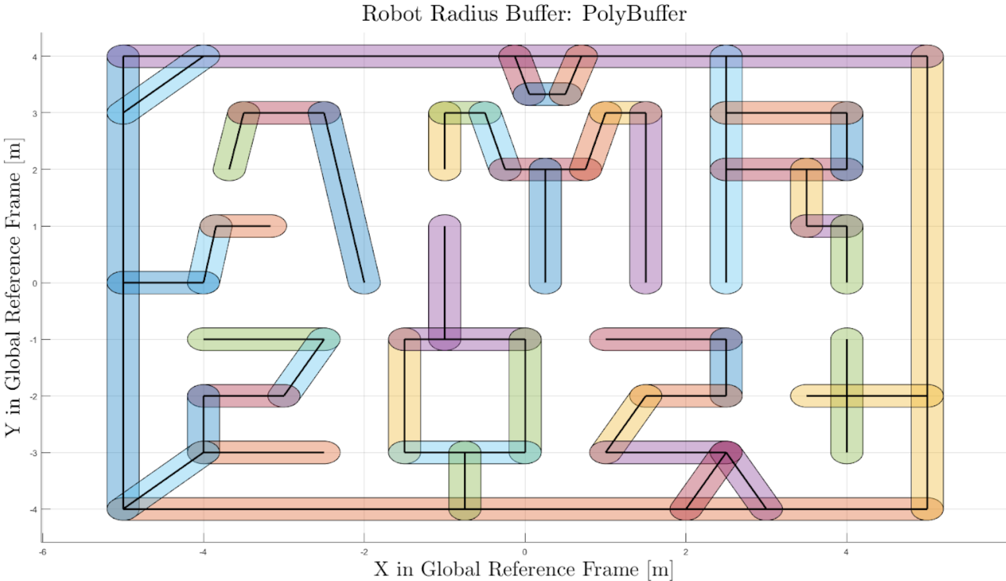


Figure 1: Robot radius buffer using polybuffer.m

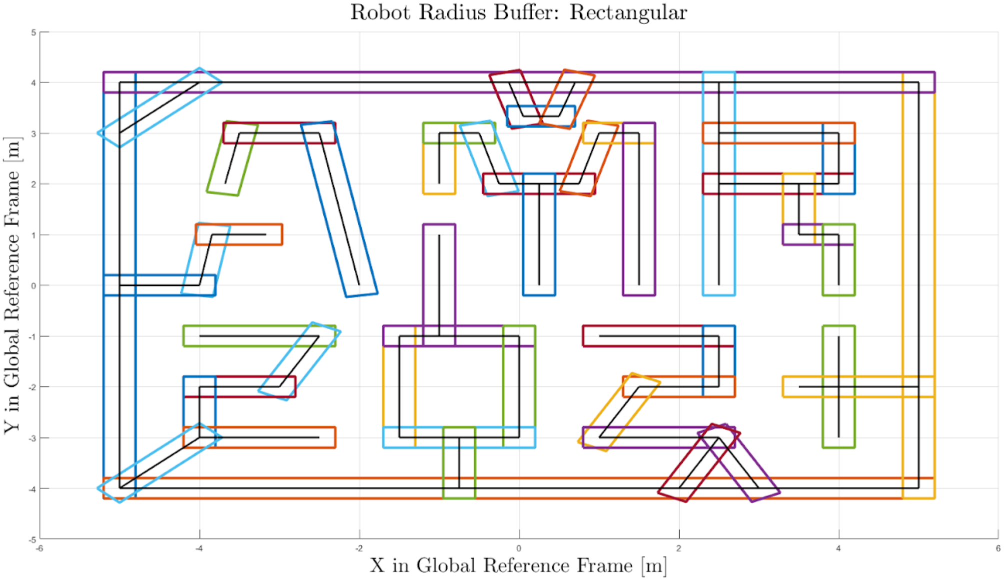


Figure 2: Robot radius buffer using rectangles

buildPRM_v2_copy (Calls: 1, Time: 118.910 s)

▾ **Flame Graph**

Flame graph is not available because the function call history size of 5000000 was too small. Rerun the Profiler with a larger history size. For more information, see profile.

Generated 16-May-2021 21:56:46 using performance time.
Function in file C:\Users\david\Documents\Documents\Senior ('20-'21)\2nd Semester\MAE 4180\Assignments\files\FinalCompetition\group5\buildPRM_v2_copy.m
Copy to new window for comparing multiple runs

▾ **Parents** (calling functions)

Function Name	Function Type	Calls
PRMtester	Script	1

▾ **Lines that take the most time**

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
262	`obs_isect = [obs_isect; isect];`	9100000	72.484	61.0%	
261	`[isect,~,~,~] = intersectPoint(x1,y1,x2,y2,obs_seg(1),obs_seg(2),obs_seg(3),obs_seg(4));`	9100000	35.252	29.6%	
256	`obs_seg = [obstacleVerts{1,z}(p,1) obstacleVerts{1,z}(p,2) obstacleVerts{1,z}(p+1,1) obstacleVert...`	9050000	7.545	6.3%	
260	`node_last = nodes(end,:);`	9100000	1.213	1.0%	
263	`end`	9100000	0.471	0.4%	
All other lines			1.945	1.6%	
Totals			118.910	100%	

Figure 3: Computation time for original PRM builder using polybuffer

buildPRM_v2 (Calls: 1, Time: 4.174 s)

▾ **Flame Graph**

Generated 16-May-2021 22:21:40 using performance time.
Function in file C:\Users\david\Documents\Documents\Senior ('20-'21)\2nd Semester\MAE 4180\Assignments\files\FinalCompetition\group5\PRM\buildPRM_v2.m
Copy to new window for comparing multiple runs

▾ **Parents** (calling functions)

Function Name	Function Type	Calls
PRMtester	Script	1

▾ **Lines that take the most time**

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
217	`obs_isect = [obs_isect; isect];`	250000	1.935	46.4%	
216	`[isect,~,~,~] = intersectPoint(x1,y1,x2,y2,obs_seg(1),obs_seg(2),obs_seg(3),obs_seg(4));`	250000	0.933	22.4%	
132	`IN_poly(j) = inpolygon(randpt(1),randpt(2),obstacleVerts{1,j}(:,1),obstacleVerts{1,j}(:,2));`	57500	0.503	12.0%	
211	`obs_seg = [obstacleVerts{1,z}(p,1) obstacleVerts{1,z}(p,2) obstacleVerts{1,z}(p+1,1) obstacleVert...`	200000	0.144	3.4%	
106	`hold on;`	50	0.124	3.0%	
All other lines			0.535	12.8%	
Totals			4.174	100%	

Figure 4: Computation time for improved PRM builder using rectangular obstacles

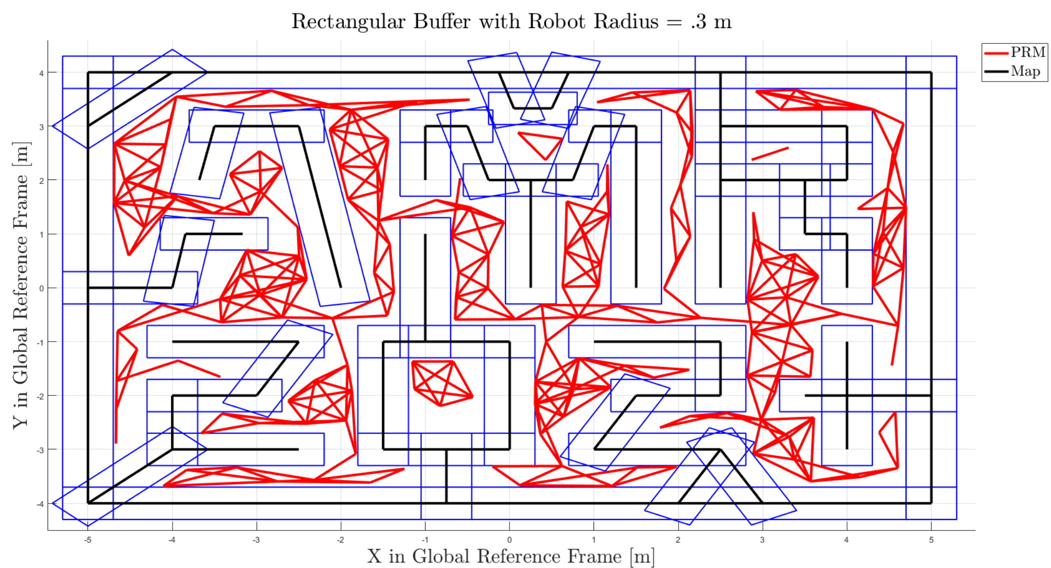


Figure 5: EC waypoint number 3, located at (.388, 2.47) inaccessible

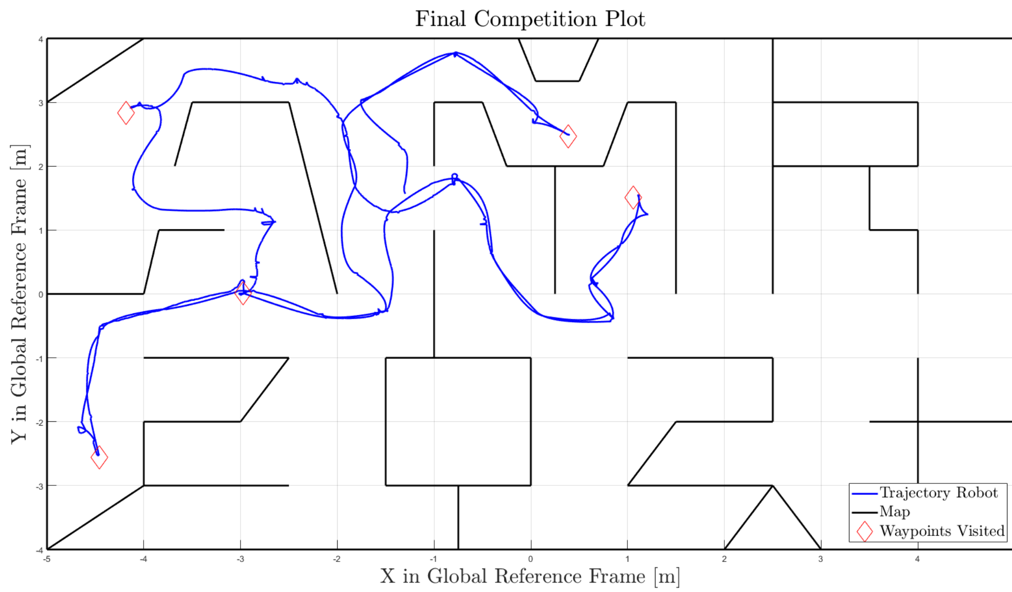


Figure 6: Final competition plot

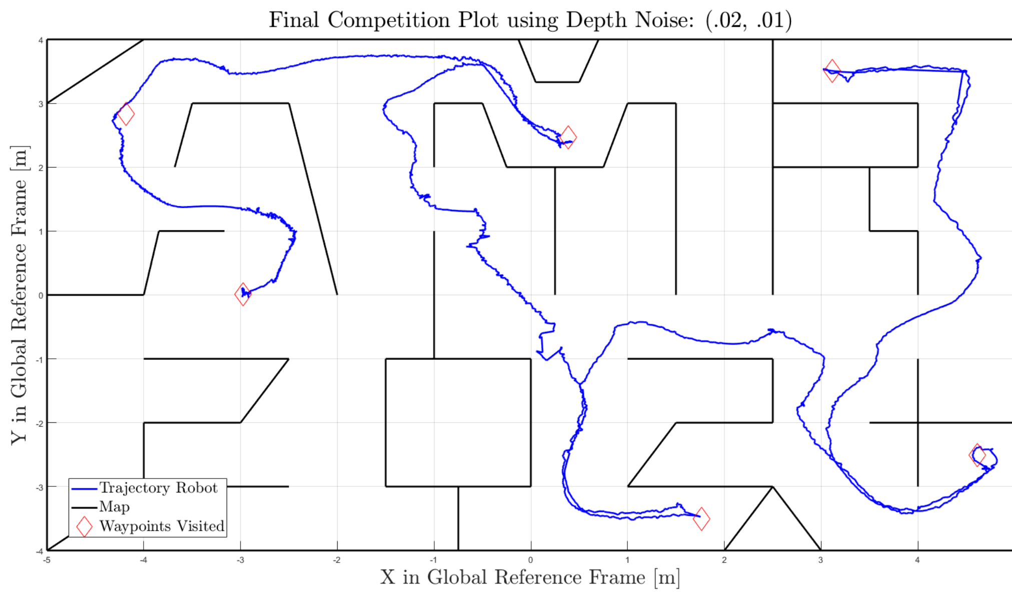


Figure 7: Trial run with depth noise, mean = .02, standard deviation = .01

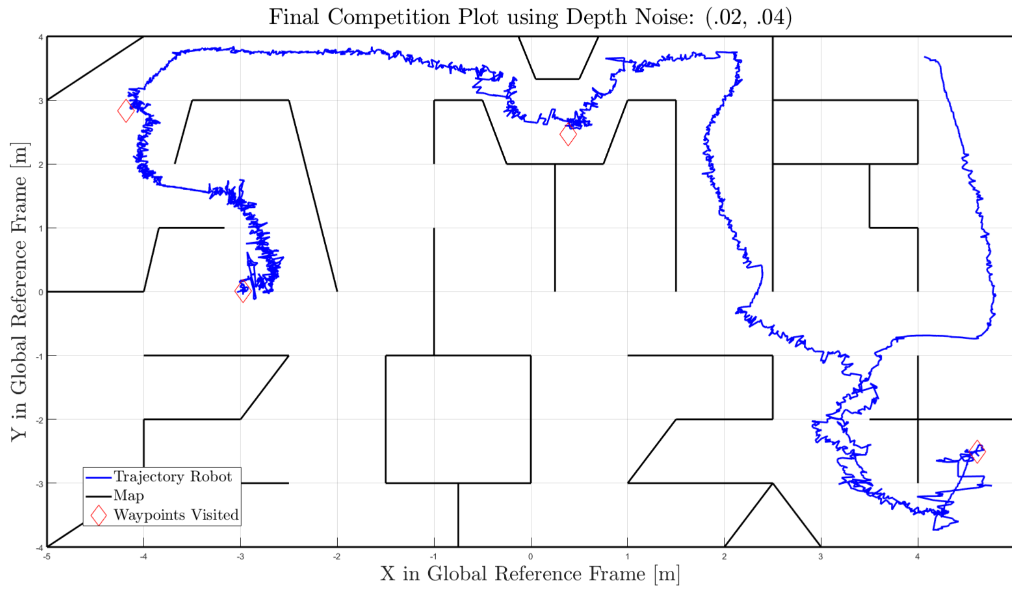


Figure 8: Trial run with depth noise, mean = .02, standard deviation = .04

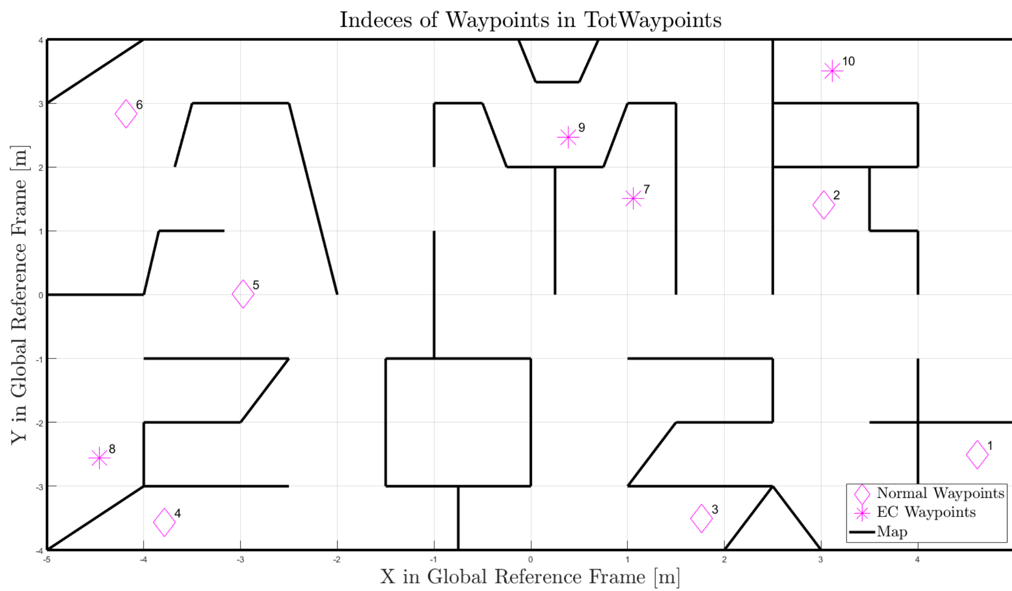


Figure 9: Indices in TotWaypoints

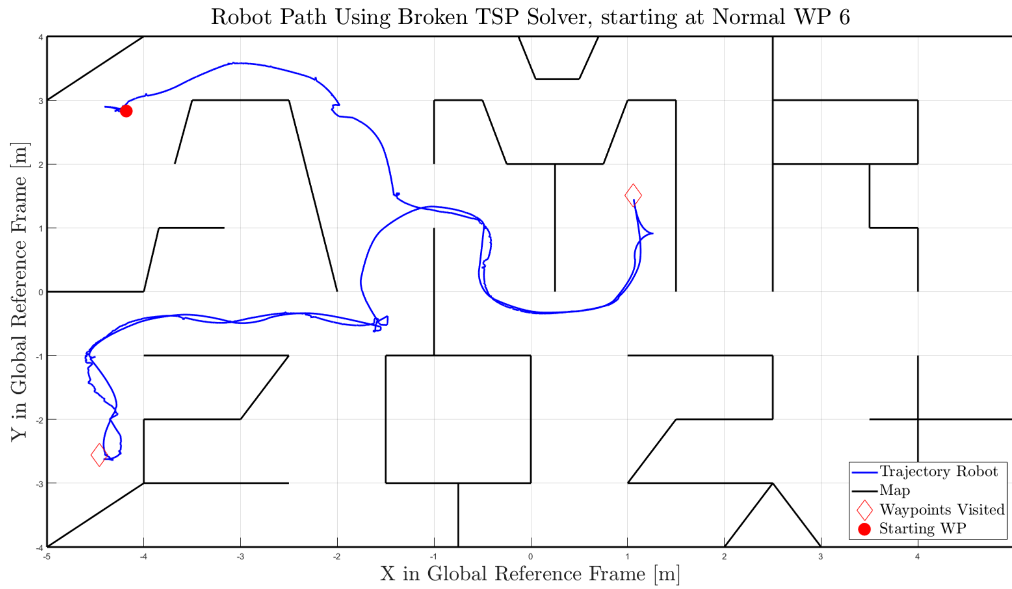


Figure 10: Faulty TSP starting at the sixth normal waypoint

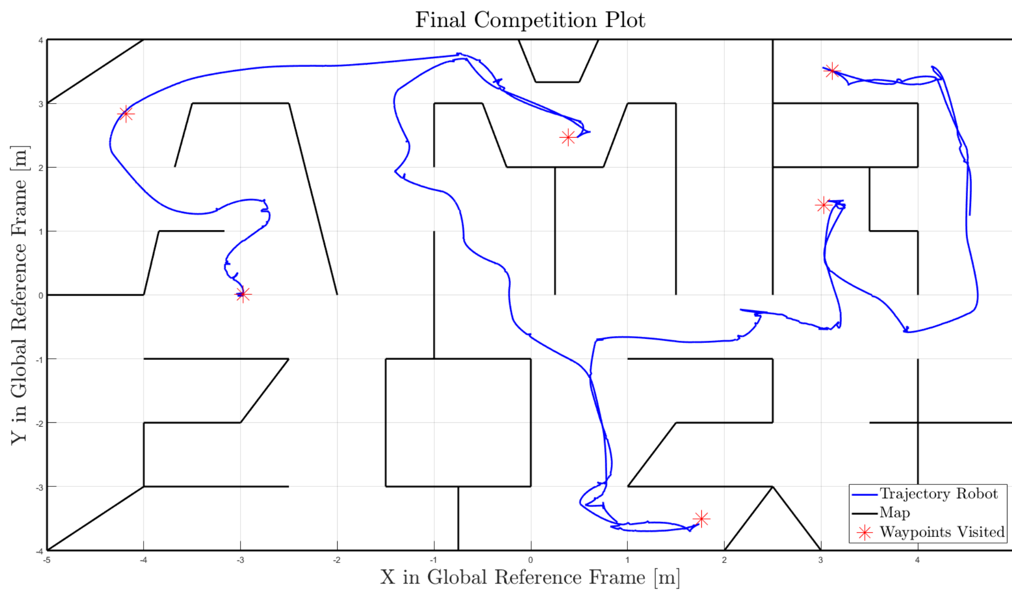


Figure 11: Fixed TSP visited 6 total waypoints (normal & EC)