MAE 4180: Simulation Competition Report

Sae Na Na (sn447@cornell.edu)

I. Overview of team approach

A. Localization

We used Particle Filter (PF) to localize the robot because we could create the initial particle set using the (x,y) coordinates where the robot may start. On the other hand, using a Kalman Filter (KF) on a multi-modal map would necessitate initializing an inaccurate mean position and a large covariance ellipse to cover all possible positions in the map.

We initially considered using beacon data to adjust the weights in our particle set, but we found that during testing, the PF by itself never caused the robot to lose localization enough to incorrectly signal that a waypoint is reached. So we decided that using beacon data is unnecessary.

B. Path Planning

We generated a roadmap using a modified reduced-visibility node set and approached this as a Travelling Salesman Problem (TSP) to determine the shortest path to all waypoints.

We initialized the map input to the path planning algorithm as the map without the optional walls. We also did not distinguish between normal and ECwaypoints. Whenever an optional wall is detected, we update the map and re-run the path planning algorithm.

To create the set of nodes, we chose the corners of all the walls and moved them away from their respective corner and into the robot configuration space by an arbitrarily chosen distance that was tested to work with the test and competition maps. This was necessary on top of the robot radius buffer as feedback linearization may cause the robot to hit the wall while doing a three-point turn.

In comparison to Rapidly-exploring Random Trees (RRT), having nodes that do not change with each run allowed us recreate the same problem to troubleshoot specific nodes and adjust the corresponding buffer distances.

C. Optional Wall Detection

Every time step, the robot compares its depth readings to the predicted depth readings with the optional walls. If depth readings from more than 5 fields-of-view are within tolerance, the optional wall exists, and the map is updated.

We initially considered trying to bump into the optional walls to determine whether it exists or not, but the difference between the measured and expected depth readings when approaching a wall that actually exists caused localization to diverge, so we went with a solution that updates the map as soon as an optional wall is in sight.

II. Flow chart of solution

Figure 1 shows a flowchart of our motion control algorithm.

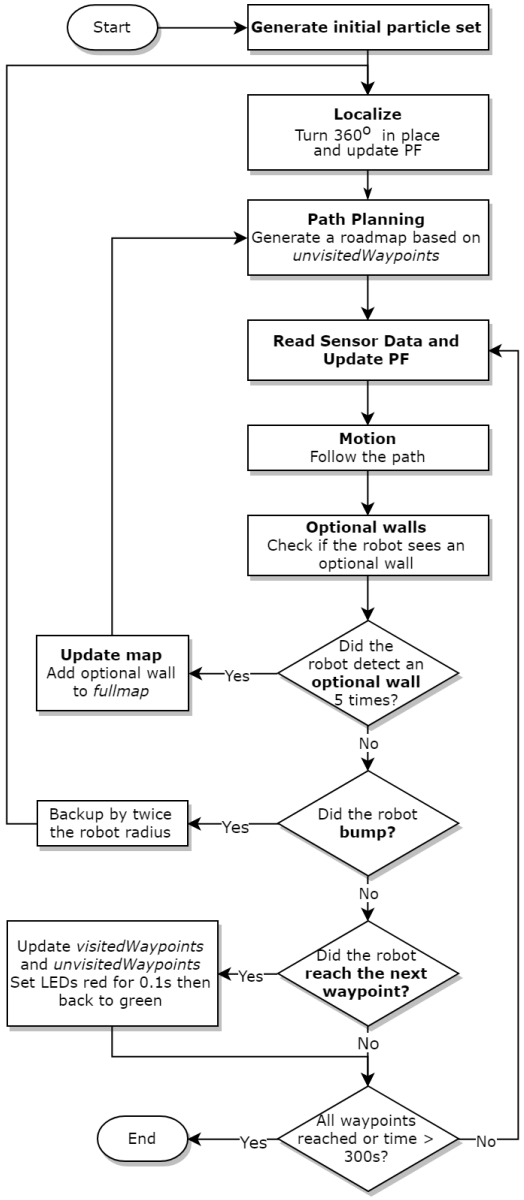

Fig. 1. Flowchart of motion control algorithm

III. Individual Contribution

I worked on localization and the integration of our group's code components into the motion control, following the flowchart logic in Figure 1.

The PF was initialized with 300 particles, distributed between the (x,y) coordinates of the waypoints at uniformly distributed angles. We have 300 particles as testing with 200 particles generated cases where the robot would initialize to the wrong waypoint. To counteract this, we increased our particle set to 300 particles and increased the σ value used by the PF to allocate weights to particles to allow for more particles to remain in the set during initialization. However, we still encountered a problem with this in the competition and this will be discussed further in Section IV.

The position given by localization was chosen as the highest weight particle at each time step. This was chosen over the mean of all particles as the mean of a multi-modal particle set would give a particle that does not represent a possible robot location.

During testing, we found that the backup bump sequence that uses $travelDist.m$ and $turnAngle.m$ contributes to the error stackup in PF because the motion control code does not re-enter the main while loop until $travelDist.m$ and $turnAngle.m$ have completed running. This gives the PF an odometry input over a longer time interval, which tends to be inaccurate during turns. To counteract this effect, we make the robot turn 360° after backing up from a bump by sending a $fwdVel = 1$, $angVel = 0$ input, instead of using $turnAngle.m$. We chose to turn 360° over 180° because the latter made the robot lose localization due to turns after bumps a few times.

Several changes were made to make the PF take less time. The PF code originally had lines adjusting the weights of particles to 0 for those that exit the free configuration space. However, this was computationally expensive to calculate, so I changed it so that particles do not go through walls but can enter the area within robot radius of the walls. Another area that improved time was calculating the map limits outside of the PF, which saves around 20 seconds.

Our group met for around 2 hours everyday during the week before competition to integrate and debug code. The PF values (σ, noise, turn angle during bump, and number of particles) had to be calibrated during testing.

IV. Discussion of Competition Performance

On our first run, the robot localized itself to the wrong waypoint, as shown in Figure 2. It was initially positioned at (-4.52,-3.52) at 90.0002°, but decided to choose the trajectory of particle B (4.55,-0.99) at 0° as its pose instead of that of particle A (-4.35,-3.35) at 90.9169°, which would have been the closest to the actual position in the particle set. During the initialization sequence, the robot rotates counterclockwise for 360°, so there is a set of particles on waypoints A (-4.5,-3.5), B (4.5,-1.0), and C (-1.5m,3.5m) that would see the exact same features in their first 180° of rotation. So as expected, within the first 3.9 seconds of the run, the weights of the trajectory of particles A and B are in the same order of magnitude, while the particles elsewhere are up to an order of 10^{13} smaller.

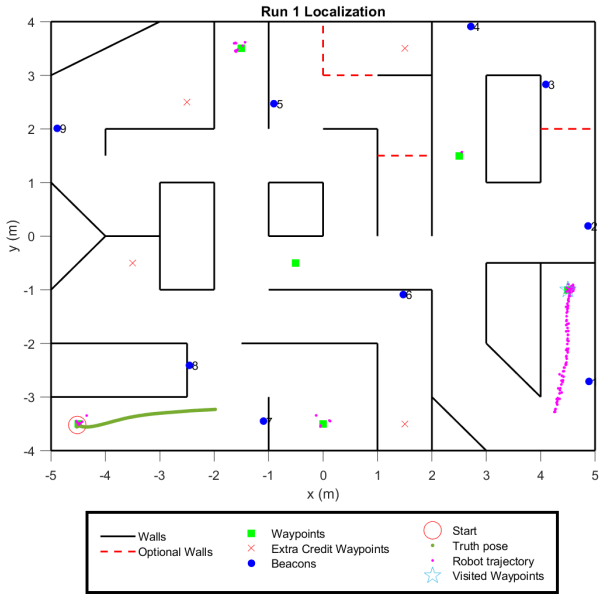

Fig. 2. Truth pose vs localization on run 1

By the time the robot was done with the initialization sequence, PF had converged its particle set to the (wrong) trajectory of particle B because of:

1) **the noise introduced in the initial particle set**, Because the error in the starting position was not given, the function that *generates the initial particle set* was designed to introduce a randomly distributed noise with a σ of 0.1. So the particle at waypoint A with the angle closest to the true angle was at (-4.53, 1.63), which is 0.137 m away from the waypoint and 0.1142 m away from the true starting location. For comparison, the robot radius is 0.16 m. So a lower noise could have been introduced.
2) **the number of particles**, There were 50 particles at each waypoint, each 7.2°apart. With more particles, there could have been a particle in the set that is closer to the true starting position. However, more particles slow down the PF which runs every time step, so increasing the number of particles is not a desirable solution.
3) **the σ value in PF**, and The σ value determines the sensitivity of the particle weights. Increasing σ keeps more varied particles that start at waypoints A, B, and C in the particle set, up until the depth readings significantly diverge from the expected. But a higher value for σ after the initial localization causes the robot to lose localization during normal turns. This can be accommodated for by setting a higher σ during initial localization and a lower σ during normal motion.

4) **choice not to use beacon data**.
If we had used beacon data, the robot would not have gotten lost in this case, as beacon 7 is in sight from the waypoint where it started. However, beacon presence is not a certain part of the rules, so the robot would still have gotten lost if beacon 7 was not there. We chose not to use beacon data as

So I modified the code to have a condition-dependent σ that takes on the value of 5 during the 360°-rotation localization sequence and 1.5 otherwise instead of the original $\sigma = 1$, and a noise of 0.05 in the initial particle set instead of the original 0.10, and 350 particles instead of the original 300. Following this change (but still not using beacon data), the robot followed the trajectory shown in Figure 3, with the same initial position as the failed Run 1 (Figure 2) in the competition. It localized properly, was able to reach 7 waypoints, and determined the non-existence of 2 optional walls.

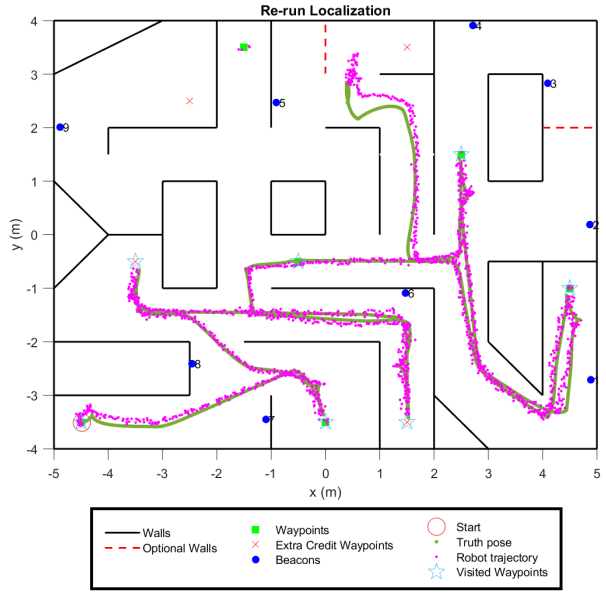

Fig. 3. Truth pose vs localization on the modified re-run of run 1

On our second run, the robot was positioned at waypoint C (2.5,1.5) with unique features visible within the first five time steps, so it localized properly, as seen in Figure A.1. It visited 7 waypoints (5 normal and 2 EC) but was unable to classify any optional walls.

If the robot had visited all the waypoints, it would have classified 3 out of 4 of the optional walls in the map. The fourth wall with corners at (4,2) and (5,2) would not be detected, because *motionControl.m* only triggers optional wall classification when the optional wall is in the robot's path to a waypoint. With this stated, the robot was unable to reach all waypoints because of the 0.2 m/s speed limit. Even with the speed limitation, some improvements that could have improved the score include:

1) **using optional walls as an input to the cost function in path planning**, and
If we created a function to give the waypoints near optional walls a higher value than those away from optional walls, the robot may have gotten more optional walls.
2) **writing a deterministic path planning algorithm**
Figure 4 shows the generated path before and after the bump. The path changed before and after, even though if the path was truly the optimal one, it should not have. This means that our code solution to TSP is not deterministic in outputting the optimal solution. The changes that need to be made would mean significantly changing the algorithm.

Something the robot did well in Run 2 is its recovery after the bump. On the robot's way to waypoint B (4.5,-1), it bumped into the corner at (4,-3). After the bump, it backed up by twice its radius and re-localized. Then it generated the new path in the second image in Figure 4. The path is only supposed to change when the map is updated with optional walls, so this change indicates that the path planning algorithm is not deterministic in outputting the shortest path.

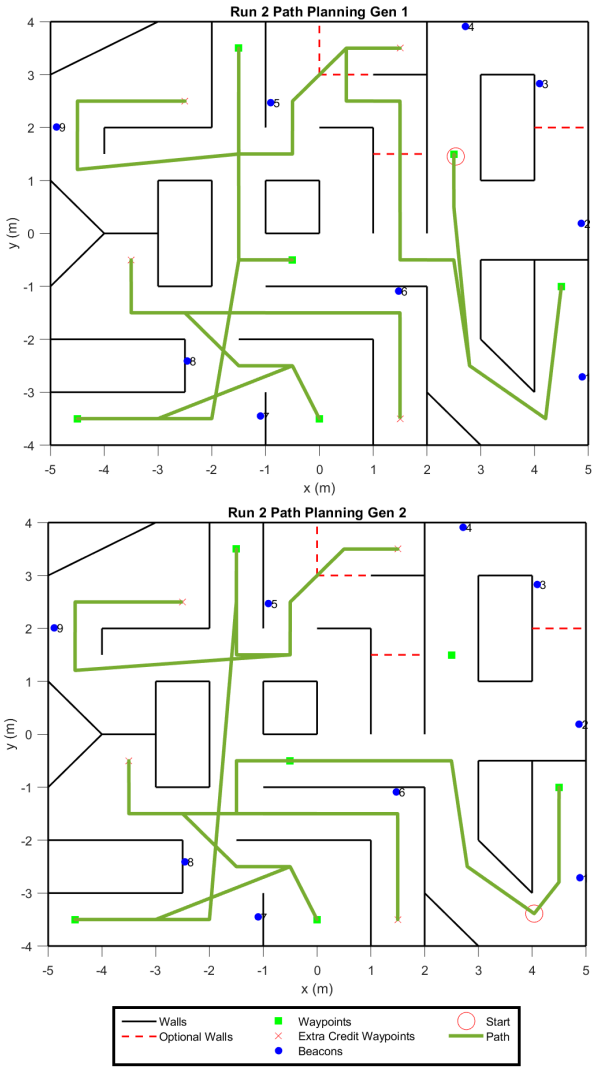

Fig. 4. First path generated on run 2 before bump

APPENDIX

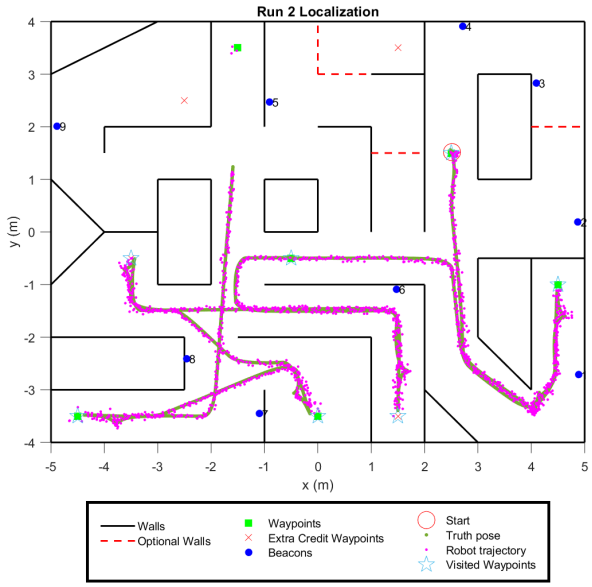

Fig. A.1. Truth pose vs localization on run 2