MAE 4180: Simulation Competition Report

Sae Na Na (sn447@cornell.edu)

I. Overview of team approach

A. *Localization*

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 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 (, 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.

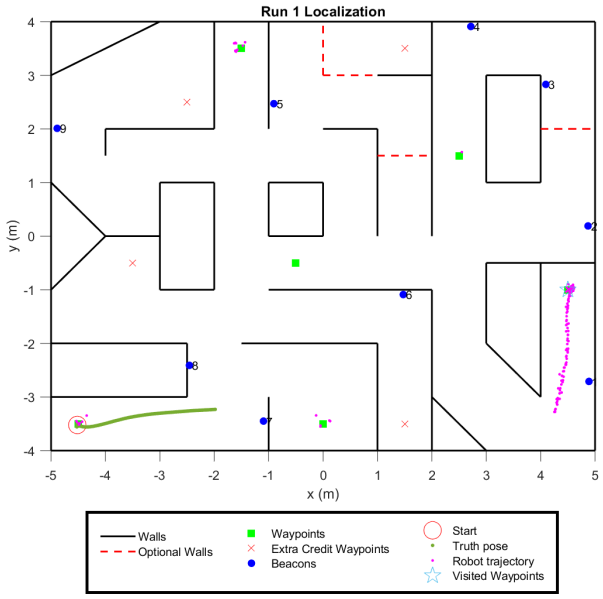

Fig. 2. Truth pose vs localization on run 1

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 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 value in PF**, and The value determines the sensitivity of the particle weights. Increasing 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 after the initial localization causes the robot to lose localization during normal turns. This can be accommodated for by setting a higher during initial localization and a lower 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 that takes on the value of 5 during the 360°-rotation localization sequence and 1.5 otherwise instead of the original $= 1$

APPENDIX

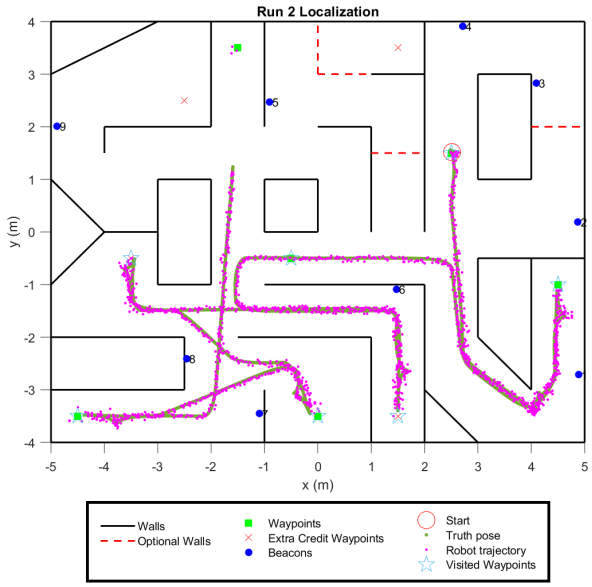

Fig. A.1. Truth pose vs localization on run 2