AlMighty Robot (AMR)

Group 4 Final Report

- Pamraat Parmar (ppp29)

Submitted on: 05/15/23

Approach

Robot was required to do 3 major tasks, localize, visit waypoints, and identify optional walls (not actually mapping, but referred to as mapping in this report).

Localization: Particle filter with depth and beacon data was used to localize robots. The discontinuities in measurements caused severe malfunctions in EKF, but particle filter with appropriate tuning was known to perform significantly better in lab sessions. Moreover, if a certain amount of randomness is enforced on particles (e.g., 20% of particles will always be randomly generated across the map), recovery from incorrect localization is possible. At the boundary, minor localization error might cause particles to move outside the map causing sever delocalization, a case which requires specific handling like assuming previous particle state or redistribution of particles in the region inside bounds.

Mapping: Since the optional walls can be associated with a binary value, i.e., 1 if it is there and 0 if it is not, grid localization with single cell across the subject wall shall suffice. However, for quick implementation, we modified the standard depth-grid localization model to only look for walls in cells where the optional walls are expected. The grid size was kept very coarse as single cell is sufficient.

Motion Planning: PRM using low discrepancy sampling was used. The cellular decomposition would make the robot go to cell and edge centers, making it a non-optimal path. RTT are much slower than PRM, as well as will probably have suboptimal path compared to PRMs. Codes for visibility roadmaps were not available to compare, and visibility PRM offered the most optimal path, however, were slower. Among random, low dispersion and low discrepancy, simulations showed low discrepancy PRMs were usually most optimal, and hence they were chosen.

Contribution

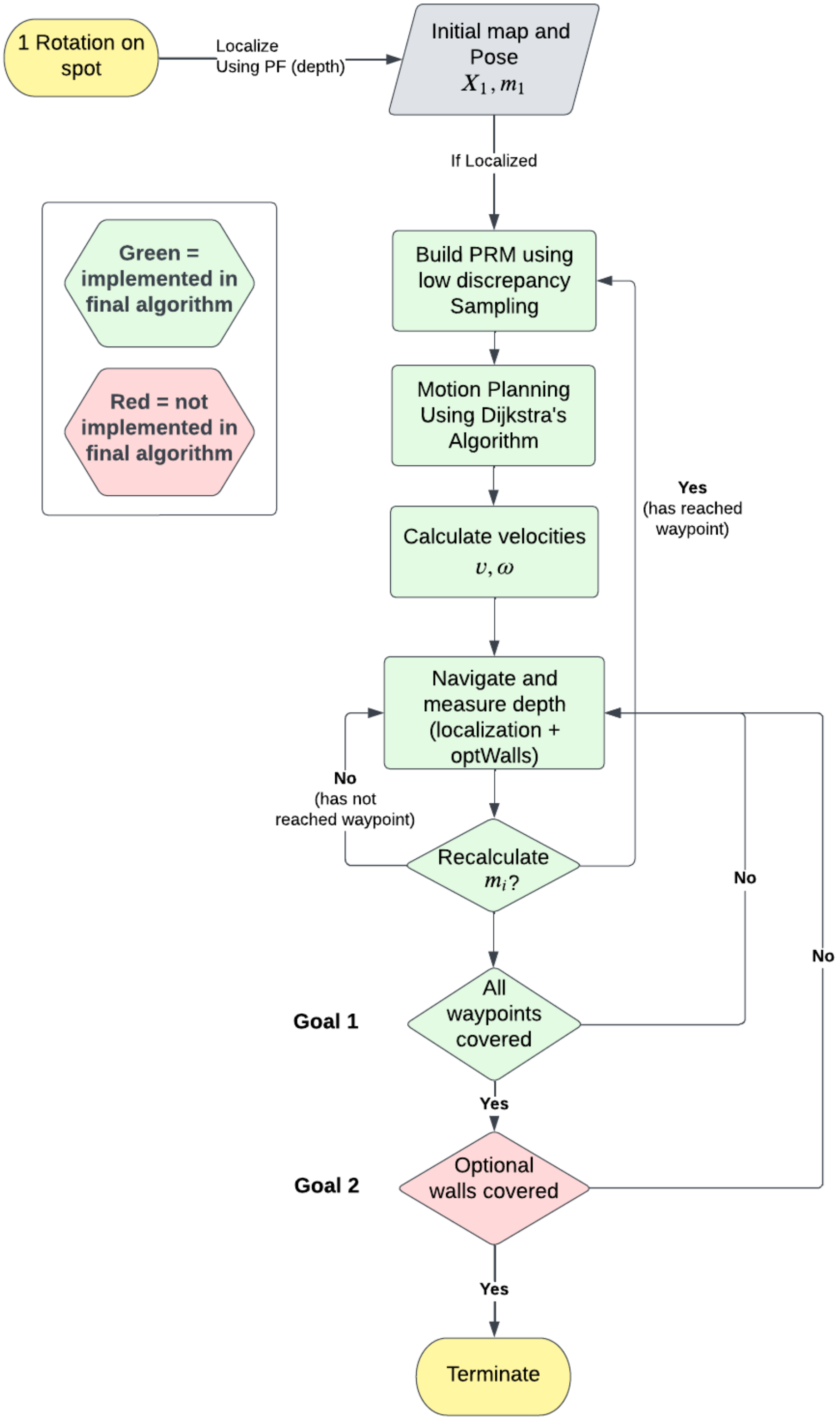
My contribution to the team was to provide individual working modules. Following functions were provided:

Codes	Description
PF.m	Particle filter
hDepthBeacon.m	Measurement function
HjacDepthBeacon.m	Meas. Jacobian function
GJacDiffDrive.m	State Jacobian function
logOddsDepth.m	Log odd calculator using depth data for grid mapping
buildPRM.m	PRM builder
edgeFree.m	Check if edge is free in map
pointFree.m	Check if point is free in map

Codes	Description
lowDiscrepancy.m	Low discrepancy point generator for given map.
integrateOdom.m	Dependency
depthPredict.m	Dependency
modLine.m	Thicken line walls
point2line.m	Convert polygon vertices to line map matrix.
global2robot.m	Dependency
robot2global.m	Dependency
limitCmds.m	Dependency

Designing and tuning PF with beacon was the most time-consuming development ($\approx$14 hours). For given $[\Delta x, \Delta y]$ measurement, there robot maybe anywhere on a circular arc centered at the beacon. However, the orientation of robot on each point of arc will be fully defined to achieve specified $[\Delta x, \Delta y]$. Initially, PF with defined orientation was developed which led to poor convergence. It was solved by not imposing the orientation but by imposing the expected orientation with added noise for command at that time step. Other functions worked as expected.

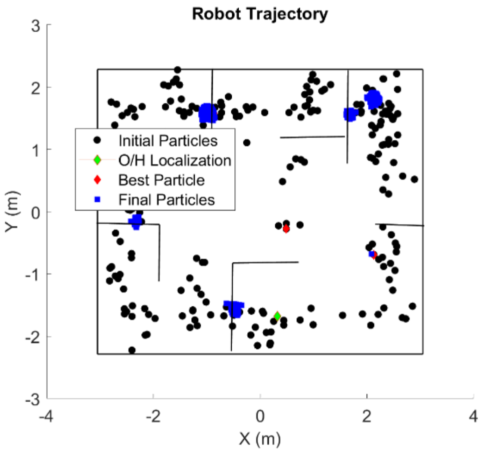
Flow Chart

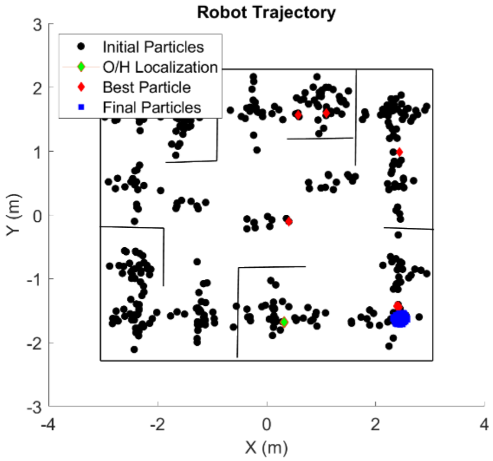

1 Rotation on spot
Localize Using PF (depth)
Initial map and Pose X_1, m_1
If Localized
Green = implemented in final algorithm
Red = not implemented in final algorithm
Build PRM using low discrepancy Sampling
Motion Planning Using Dijkstra's Algorithm
Calculate velocities v, ω
Yes (has reached waypoint)
Navigate and measure depth (localization + optWalls)
No (has not reached waypoint)
Recalculate m_i?
No
No
Goal 1
All waypoints covered
Yes
Goal 2
Optional walls covered
Yes
Terminate

Lab tests

A major drawback of our code was that it was insufficiently tested in the lab. We attended 2 open lab sessions, 14:30 hrs. to 16:00 hrs., 27th April, and 13:00 hrs. to 16:00 hrs., 8th May. Open hours were booked for Thursday but due to no working progress since the first lab sessions, were not utilized. Particle filter without beacon was implemented by the first lab session and the robot failed to localize at any position. This was expected as random guesses were assigned to Q and R matrices. PF was throwing error as the particles were jumping out of the maps, thereby being assigned 0 weight, and since all the particles had 0 weight, *randsample* command failed. Lab 1 run data was not stored as it was considered meaningless and not useful.

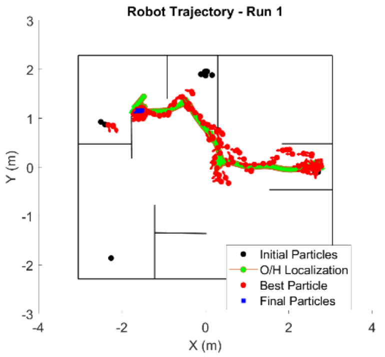
Beacon function was incorporated in PF and tuning was attempted on 27th April after adding error handlers. Figures below show performance of lab runs. Particles were being drawn close to beacon due to minor implementation error which was corrected post-lab. Operational codes were not tested once by the team in the lab hours.

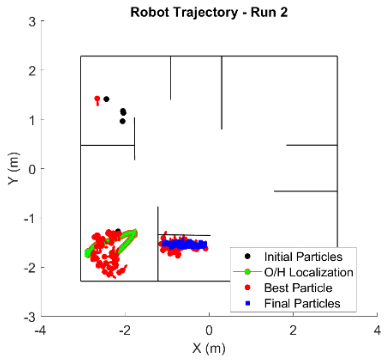


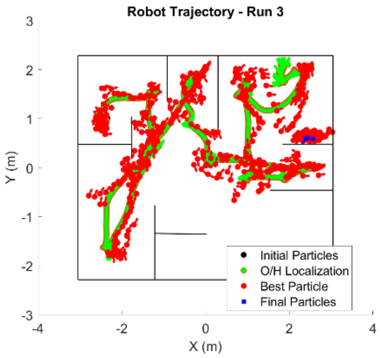


Competition Performance

The team attempted 3 runs. The third run behaved well and was able to visit all the waypoints and 2 extra credit way points. The first extra credit waypoint was detected mildly away from where it was. All other waypoints were correctly detected. The figures of all 3 runs are showed below:

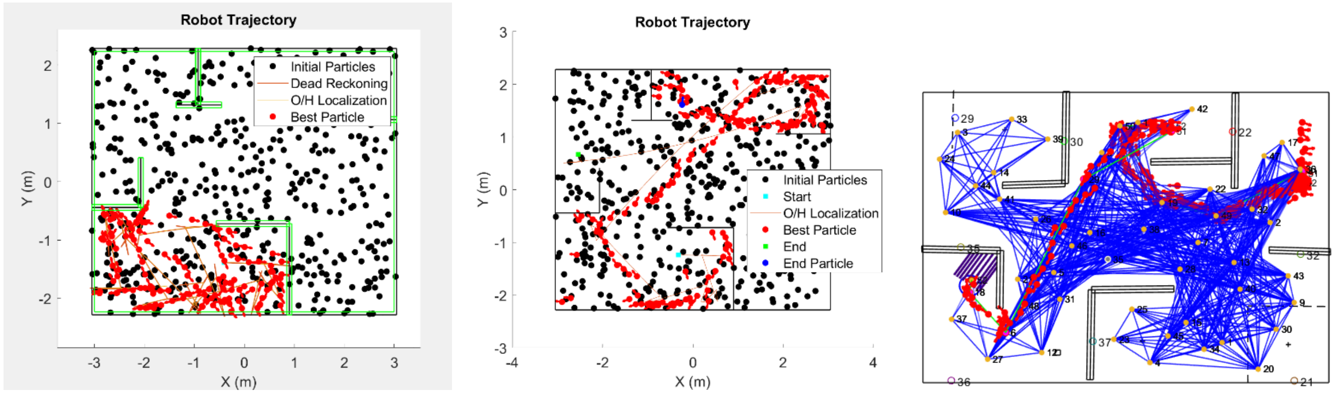






Localization: Particle filter with 300 particles initialized with equal distribution at all the waypoints with orientation equally distributed in [0, 2π] at each waypoint. R = diag([0.005 0.005 0.001]) and $Q = 0.005\bar{\bar{\boldsymbol{I}}}$. The initial localization was noted to be very good, as in all three runs the robot identified its location within 0.2m of its actual location. The PF was delocalized in run 2 when the robot was stuck in an infinite loop of collision with wall (explained in motion planning). The PF was tuned on lab-2 datasets. The PF localized robot for entire run correctly (max error of

≈0.3 m after convergence) with beacon and depth on totally new and untested datasets (Figure 1a). In the second run on untested dataset, the particles failed to converge consistently towards the end of run (Figure 1b).



(a) First run on untested data

(b) Second run on untested data

(c) Simulator run

Figure 1 Simulated testing of Particle Filter

Next, the PF was tested on simulator with beacon, and it was consistently malfunctioning. The simulator localization was excellent with beacon off, as shown in Figure 1c. It was decided to not use beacon in the competition. The strategy described in 'Contribution' section was being used to localize using beacon, and the particles were being redistributed on an arc centered at beacon such that beacon measurement $[\Delta x, \Delta y]$ are imposed. This distribution on arc ignored the dynamics of the robot and if the beacon was significantly farther, the error in reading would grow leading to delocalization. Since the particles were restricted to the arc, the next best choice of particle had to come from that set, thus giving unrealistic positions.

Mapping: Grid based mapping was done, only in the cells in the vicinity of optional walls. Strategy was to declare the wall present and reassess the path based on new walls. The robot was not able to identify walls. The logOddsDepth function performed as expected, however, the log odds were being analyzed in the after the run, i.e., after it has visited all the waypoints. Due to this, even if the walls were identified correctly, they would not be stored in the dataset if the program id terminated halfway.

Motion Planning: PRM with Low discrepancy sampling were used with waypoints being goals. The map was modified such that each line has thickness of 0.1 m. Moreover, extra thickness of robot radius was associated to each line which would make PRM builder reject the edges which may cause robot to collide. 50 points were sampled in Q_{free} for PRM. In bump, the robot was programmed to backup and turn 30 degrees towards opposite direction of where the bump was detected. The PRMs were produced appropriately in the competition, however, the backup bump wasn't implemented correctly. In implementation of backup bump, the robot was commanded to keep moving till time taken to cover 0.25 m with 0.2 m/s speed elapses. Once that is done, the timestamp was assigned value -1, which would indicate the algorithm to move to turning sequence. Robot would turn till time has elapsed such that robot has turned 30 degrees. Once that is done, the timestamp value was assigned value of '-1', which was a mistake. An appropriate value to be assigned would have been -0.5, so that if/else statement can distinguish properly if it needs to go to reversing mode or turning mode. Since the value assigned was -1, the robot turned only the first time, and then every time it would go on the same path and thereby colliding at the same spot repeatedly. The accounting of robot radius was also flawed, i.e., even though none of the edge of PRM crossed regions which would cause robot collision, robot being non-holonomic and usage of feedback linearization meant that the robot was not travelling on the straight lines to the intermediate nodes of PRM. The initial deviation of robot from straight line near the walls would cause it to collide. A better way to implement this would be to have more clearance from walls, by accounting 2 times the robot radius while thickening the walls.