Brendan Caporaletti

Dara Behjat

Mobile Robotics Path Planning

Project Write Up

**What was the goal of your project?**

Building a path planning algorithm for the Neato was the big picture goal of this project. Given a map of a world we wanted to be able to create that efficient path, have the robot effectively navigate through that path, and avoid obstacles along the way. Our main focus was on learning our path planning algorithm and implementing it efficiently. Although we did make some intelligent improvements to our algorithm, the bulk of our work revolved around path following rather than path finding.

**How did you solve the problem? (Note: this doesn't have to be super-detailed, you should try to explain what you did at a high-level so that others in the class could reasonably understand what you did).**

We decided to use Dijkstra’s algorithm tweaked with some heuristics (A\*) to find the shortest path through the map. We essentially broke out the map into a grid and applied costs to every node transition in the grid. This allowed us to determine the path with the least amount of costs based off nodes traveled. To avoid obstacles, we applied a cost of nearly infinity to every node occupied by something so the neato would never travel to or through them. To account for real world error, and the dimensions of the robot, we added a buffer to increase the area of the walls to stay away from bumping into obstacles in passing. Our A\* heuristic was simple, by adding a cost associated with absolute distance from our goal we caused to algorithm to favor paths towards the goal.

To follow this path we created waypoints to help the neato localize itself and determine its path. To get to these waypoints accurately we implemented simple proportional control both rotationally and linearly. This forced the robot to slow down when it approaches each way point to accurately land on each waypoint without overshoot. The way we started to approach localization was using simple odometry which gave us very fast updates but also tended to drift and vary over longer distances. This only allowed us to accurately follow short paths. We switched to using particle filtering to more accurately determine our position and orientation while not accumulating compounded error. Paul’s my\_pf script and amcl behaved similarly in terms of speed and accuracy. We went with amcl due to the ease of resetting its particle cloud with a pose guess. The main issue with our particle filter was how slow it updated our pose. In order to precisely control the neato we needed constant pose feedback. We added a dead reckoning component to our position estimation to “interpolate” our position between amcl updates.

**Describe a design decision you had to make when working on your project and what you ultimately did (and why)? These design decisions could be particular choices for how you implemented some part of an algorithm or perhaps a decision regarding which of two external packages to use in your project.**

Localizing the neato within the map proved to be our most difficult challenge. Our choices were odometry or particle filtering. We started with extensive testing on both solutions. As mentioned above odometry was great for our feedback control system because it updated quickly, as for overall path following the slight drift compounded over time making it almost useless for any sizable length path. We could see the drift in rviz by using the fixed coordinate of the map and watching our laser scans move from being lined up with the map to being very far off. We switched to particle filtering in order to improve accuracy and hopefully remove any accumulated error. However, the slow response time meant that our feedback system would not be updated frequently enough often causing overshoot, and stable oscillations (not converging on our goal). After a large amount of testing neither system seemed that it would achieve our goal, and this is where our dead reckoning interpolation came in. With our amcl particle filtering we combated the slow update timing by simply updating our position based on the actual angular and linear velocities we were sending to the neato. This solution was quite effective as we saw that the dead reckoning interpolated accurately between amcl updates, meaning that the amcl updates were not far off our predicted position.

**How did you structure your code?**

We split our task into two modules path\_finder and path\_follower. Path\_finder contains two important classes, firstly the Map class contains our representation of a map and has a function to perform dijkstra's algorithm between any two points on the map. The Path\_finder class uses the map and does all necessary subscribing and publishing to allow our path\_follower to operate. Path\_follower contains a single class that controls the robots waypoint navigation.

I will walk you through the main flow of control when navigating from point to point on our map. We start with amcl, path\_finder, and path\_follower nodes running, path\_follower has an empty queue of waypoints. Path\_finder’s starting position is updated by amcl pose messages. In rviz when a point location is published path\_finder calculates the path from the robots position to the new point. This path is published and updates path\_follower’s waypoint queue. Path\_follower uses amcl input to navigate through each waypoint till its queue is empty. The map, path, amcl pose, interpolated pose, and current waypoint are all picked up by rviz for display.

**What if any challenges did you face along the way?**

Dijkstra:

When trying to implement Dijkstra’s Algorithm at first we ran into problems in figuring out how to queue our nodes. We ran into an issue with our test of the algorithm because it would decide to run through walls and take very inefficient paths. We realized that we were ordering our nodes in the wrong way, we had to make sure to always take the minimum cost node first and we used the heap function to make sure that happens, and pull out the minimum.

Bug:

Due to a gross misunderstanding of quaternions we were taking the z value times pi to represent our Yaw. This was quite an interesting bug to find because over some ranges of angles this calculation seems to be fairly accurate. Due to this fact our waypoint navigation actually functioned, but would only approach the waypoint from a particular angle, an extremely interesting behavior indeed. Once discovered we used the Quaternion to Euler formula to calculate Yaw from the quaternions.

Transforms:

Lining up each coordinate frame when integrating the various ros nodes was a very important issue. We had to make sure the map, amcl, and dijkstra path all had the same coordinate frame. The solution was simpler than expected, we used the map as our universal fixed frame and just made sure that any message that needed to be lined up with the map was stamped with a header that stated the frame\_id as “map”. One interesting issue was that somehow our path finding node had its x and y dimensions flipped, so we have to remember to flip all x and y coordinates when moving in or out of the path finding node.

Real world:

We also ran into an issue with navigating through our path in an imperfect world. The buffer zones needed to be tweaked several times to make sure we didn’t clip cabinets on our path, and we needed to further improve localization because our waypoint navigation would at times push small angular velocities in addition to linear velocities but the angular velocities wouldn’t be reflected because of the wheels on carpet.

Our localization challenge was covered in the previous question.

**What would you do to improve your project if you had more time?**

Our first foremost goal would be to improve localization by eliminating any odometric drift. Even the particle filtering solutions become less accurate with the odometry.

Our second goal would be to extend our project to path planning in an unknown environment. We would use SLAM to build our map and localize simultaneously while continually updating our optimal path. It would be interesting to play with known vs unknown occupancy cost weighting to see how it would affect how the robot attempted to navigate in its unknown environment.

**Did you learn any interesting lessons for future robotic programming projects? These could relate to working on robotics projects in teams, working on more open-ended (and longer term) problems, or any other relevant topic**

Working together was very effective and it helped us think organically and work off of each other to solve problems. It would have been smarter to set a finite goal much earlier on in the project as our project was ever evolving throughout our attempts to move towards completion.

In terms of technical lessons, we found that rviz along with standard messages were extremely useful for inter-node communication. It was so easy to publish any information and immediately visualize it alongside our map and all other data in rviz. It was also awesome that all of our nodes could listen to points and poses being specified by mouse clicks in rviz. ROS has proved to be a elegant and robust system so adhering to its standards is almost always the way to go.