Olin College of Engineering

Computational Robotics: Mobile Robotics Report

Victoria Coleman
Mac-I Crowell
David Elkan

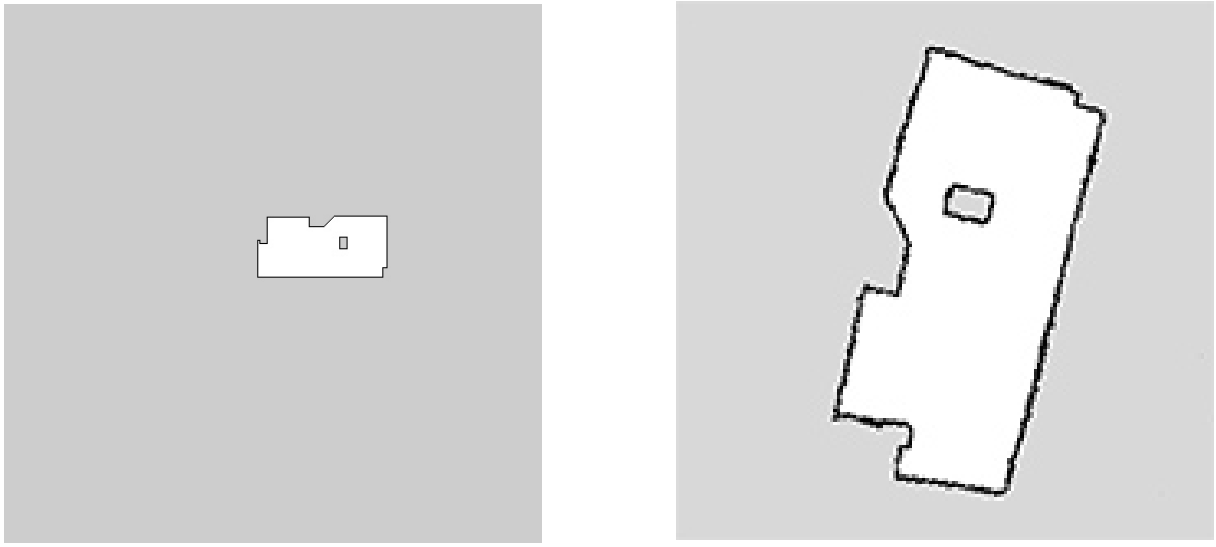
October 16, 2014

Goal

Our goal was to tackle the scaffolded project and implemented a particle filter for dynamic localization of a robot in a known environment. We wanted to create a solution to the "kidnapped robot" where a robot is randomly placed in an environment with a map and no initial guess as to the location. We wanted to ensure that the implementation was not too computationally intensive to prevent it from being practical on the time scale of the robot's movement. We also wanted to account for the cases when that if the prediction is thrown off by inaccurate sections in the map or unexpected dynamic obstacles. To account for that, we wanted the decrease the recovery time for finding the correct position again when the disruption clears.

Approach to Problem

The problem is how to probabilistically determine the location of a robot in a known environment. Our approach was to first implement the scaffolded portion of the project and then to determine adjustments to improve the algorithm. We got everything working with the simulator, and it seamlessly transitioned to real life without any bumps an only a little less accurate. We got the traditional particle filter working as described in the project prompt, but we adjusted it so that the predicted position is determined by the average of only the top third most probably points. Also, we made it so that one third of the points are always randomly distributed in the areas of the map known to be unoccupied. This would make it faster to recover from clustering in very erroneous locations due to disruptions.



(a) This map came from us measuring the room and building the map in Photoshop. This map does not take into account the curvature of the upper wall.

(b) This map was made using hector SLAM on the Neato.

The maps we tested the particle filter with.

Design Decision

We chose to embrace randomness and use it to improve the algorithm's ability to recover from multiple kidnappings and temporary unexpected changes in the environment. In the simple implementation of the particle filter all the points will cluster rather closely in the map. However if the robot is suddenly in a new spot there will be no position hypothesises in near that spot. This causes the robot to be unable to recover (or if it is able, it takes a really long time). In order to combat this issue, we only selected two thirds of the new particles based on the weights from the previous generation of particles. The remaining third were chosen randomly from the unoccupied spots on the map. In experimental testing this allowed the robot to recover quite quickly from being "kidnapped".

Code Structure

We made no major changes to the way the code was structured since we did the scaffolded project. The code is structured into a the classes: TransformHelpers, Particle, OccupancyField, and ParticleFilter. each class had methods specific to it and these were the only methods used.

Challenges

The largest technical challenge we faced was debugging with large amounts of data. We had an error when calculating how much the lidar scan should be rotated. Essentially the issue boiled down to dividing 2 integers and expecting them to return a float. It took us a long time to figure out this bug because there were so many data points from the lidar and so many hypothesized points. This particular bug was discovered by visualizing the rotated lidar scan for each point one at a time. This showed us that the scans were condensed into a single line. In the process of discovering this error we made another change that also led to catastrophic failure. This bug resulted in our weightings being way to similar to each other and therefore the points not clustering as expected. This bug was hard to deal with because there were so many weights, so many data points contributing to each weight and we didn't know what each weight should

actually be. This issue was diagnosed by us assuming that the weights all looked too similar and reverting to our old approach for calculating weights.

Improvements

If were to do more work on this project, the most important item to consider is making the algorithm more accurate to the true positon of the robot. The position is quite on point, but the angle seems to have an offset. We would also try to make the algorithm more computationally efficient. It would be interesting to explore whether it is useful make the number of hypotheses or resolution of scans analyzed dependent on how confident the current guessed position is (fewer points when more confident). It is also worth exploring improving the map as as the robot navigates through it or expands the unknown areas of the map after it strongly identifies its location.
Another improvement that may be worth pursuing is color-coding of particles based on weights. This would allow us to easily tell which areas of the map had high probability clusters and whether the particles were reasonable.

Lessons Learned

One of the key challenges of this project was the shear quantity of data, and the difficulty to interpret which data was reasonable. Printing out all of the particle weights to check for reasonable values is less useful when there's no way to know which particle on the map corresponds to which particle in the list. The code had the additional challenge of being difficult to test incrementally. Since each function depended on so many others, it was often difficult to tell which of component was broken when something went wrong. We also encountered challenges with error handling and identifying errors. To combat these challenges we would focus on writing code with debugging in mind. Making components more modular and paying close attention to error messages also helps. We also found scaling down certain components to reduce the data load and speed up testing was helpful.

Key assumptions should be considered carefully. One of our code's largest bugs turned out to caused by a simple error in an assumption we made early on in the project. Had we spent five more minutes early on questioning the assumption it would have been resolved easily. Because we did not do so, we ended up spending several hours debugging before we found the issue.

We also found that increasing uncertainty was occasionally beneficial. Code that is too specific is vulnerable to errors, and we found that always allowing some of our particles to be completely random increased our robot's ability to adapt to unexpected disturbances.