Eitan Romanoff, Nathaniel Case

Mobile Robotics Programming Report

Strategy

The strategy we took used particle filtering for localization, a probabilistic roadmap for pathfinding on both a global and local level, and potential field for path execution and obstacle avoidance. The robot attempts to use a random wander technique until sufficiently localized, to which it will switch over to a potential field movement heuristic.

The following is a high level overview of the general structure of the execution of the robot:

```
for each goal in input file
    while the robots position is not at goal
        if the robot is not currently localized
            safeWander()
        else
            safeGoto(goal)
        attempt to localize current position
```

The idea is to wander around the building until enough data has been picked up by the robot to get an idea of where it might be. Once a good guess has been made, the robot can begin to plan and execute a plan towards the goal. As it moves, it continues to localize, and if the robot determines it made a bad localization guess, it returns to wandering until it has a better guess.

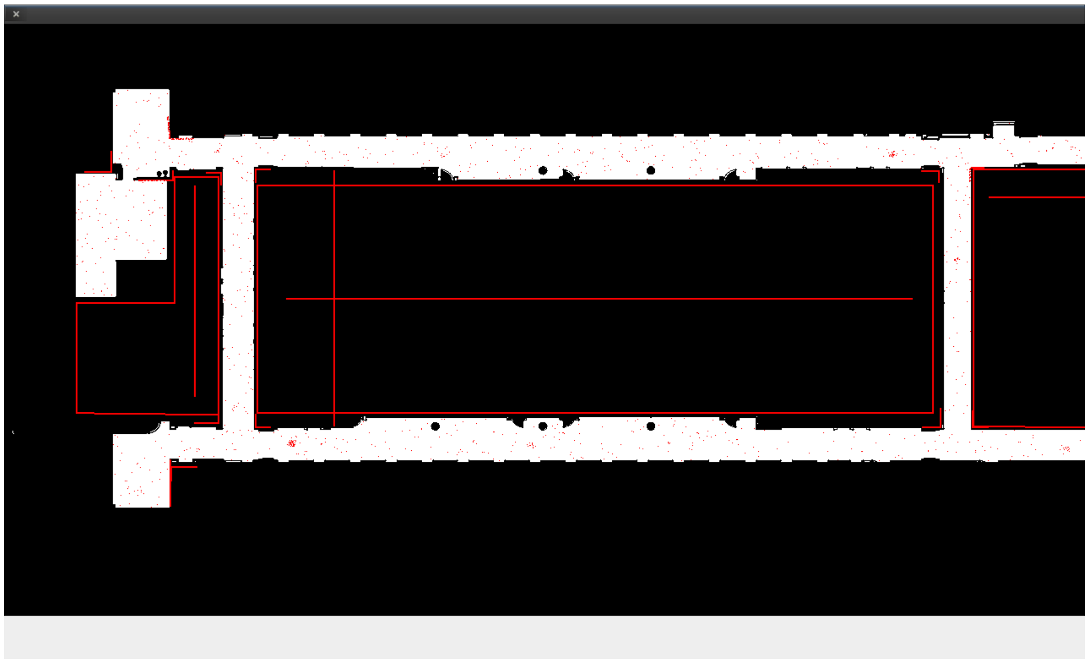
Localization

Particle filtering intrigued us with its relatively straight-forward approach, and with the opportunity for helpful data visualizations with particle clusters (which would help in the development and debugging of this method). Our implementation of this approach is as follows.

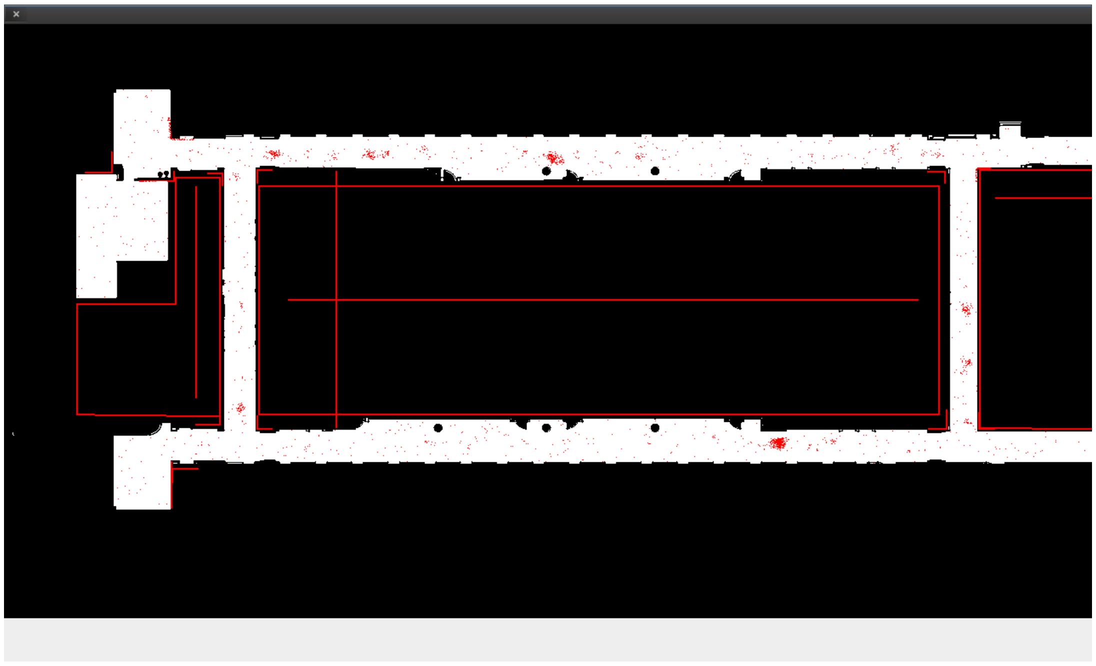
```
Take a snapshot of the robot’s sensor state
Take a distribution of random points
For each point
      Apply a random chance to kill the particle instantly
      Update the particle position based on robot movement
      Derive confidence by comparing robot sensor data to projected particle sensor data
            If the robot moved into an obstacle, or if confidence < threshold
                  Kill the particle
Get an average position of all particles with “very high” confidence
If the position is feasible, use this as estimate
Repopulate particles to get back to original quantity
```

The snapshot of the robot’s sensor state was simply taken by recording the data of the sonar modules, and using each as a feature for comparison. Particle states were estimated with the same angles and max ranges of the robot. The overall confidence was derived by taking the distance between the particle’s features and the robot’s features, and subtracting it from 40 (the maximum possible distance between a particle and robot based on the features is 5 meters * 8 sensors). The smaller the distance, the higher the confidence. Our threshold value for keeping particles was typically 35. These values were then normalized so that the overall probability between all particles was 1.

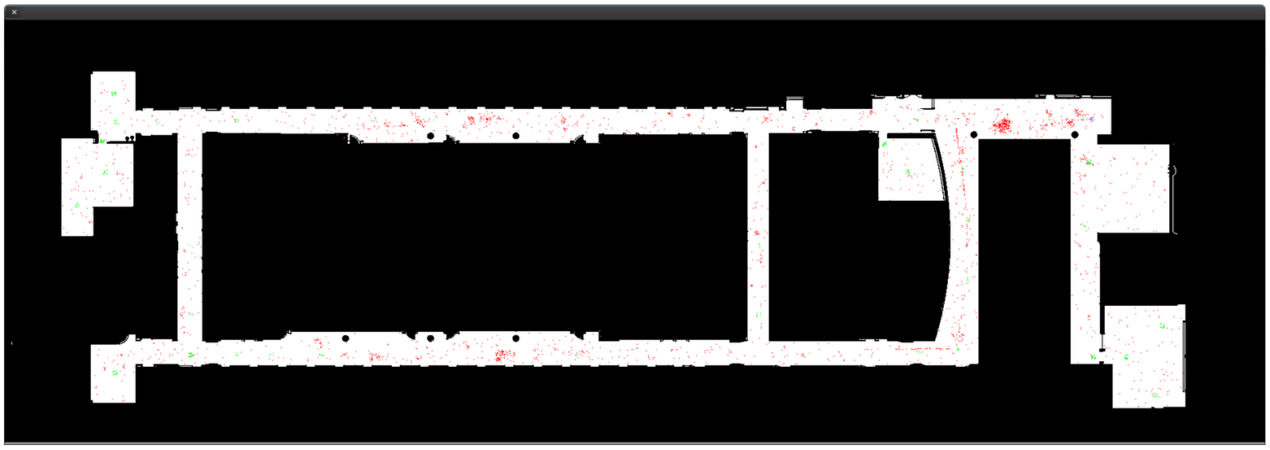
This method ended up working relatively well. While featureless hallways produced several smaller clusters that represented potential positions, and overly feature-rich positions resulted in particle genocide, the robot would settle on an accurate position after some time wandering.



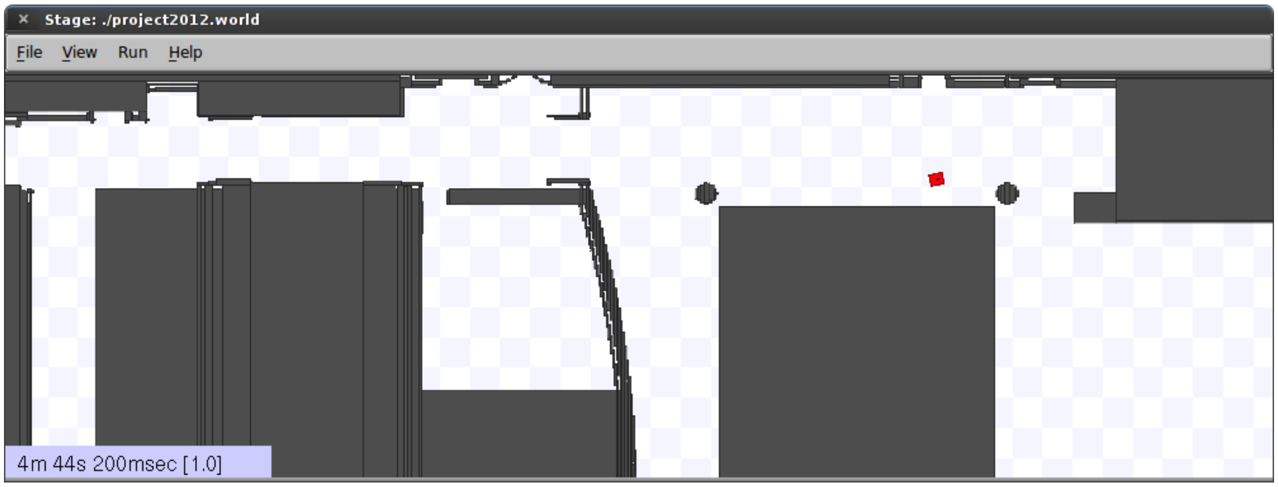
Particles Displayed Over Open Space (K = 4000, right after initialization)



Particles Displayed Over Open Space (K = 4000, after 15 seconds)



Particles Displayed Over Open Space (K = 4000, localization has converged)



Position of Robot in Player (shortly after the last picture)

Typically, the robot's position would be deduced in under a minute of wandering around in simulation. Due to the random nature of the spawning and despawning of particles, this process can take much more or much less time than average. In the worst case, the robot would eventually wander into one of the corners of the map and turn in circles until it has determined which corner it is most likely to be in. Once it has a good enough guess which is in the open space of the map, it moves on to the next part of the sequence, planning the path to the goal position.

Path Planning

Because we were offered a map ahead of time of the free space in the simulation, no dynamic mapping was necessary. The probabilistic roadmap method made the most sense because it was straight forward, easily implementable, and easily tailored to this particular environment. The initial method of mapping was taking a random gaussian distribution of points in free space, and performing a K-Nearest algorithm to create a connected graph of the open space. With a selection of 200 points, and using a K value of 8, the algorithm wormed as follows.

```
For each point A,
    While i < 8 && there exists points untried
        Find the nearest untried point B
        If the edge AB does not clip the world obstacles
            Add AB to the graph
            i++
```

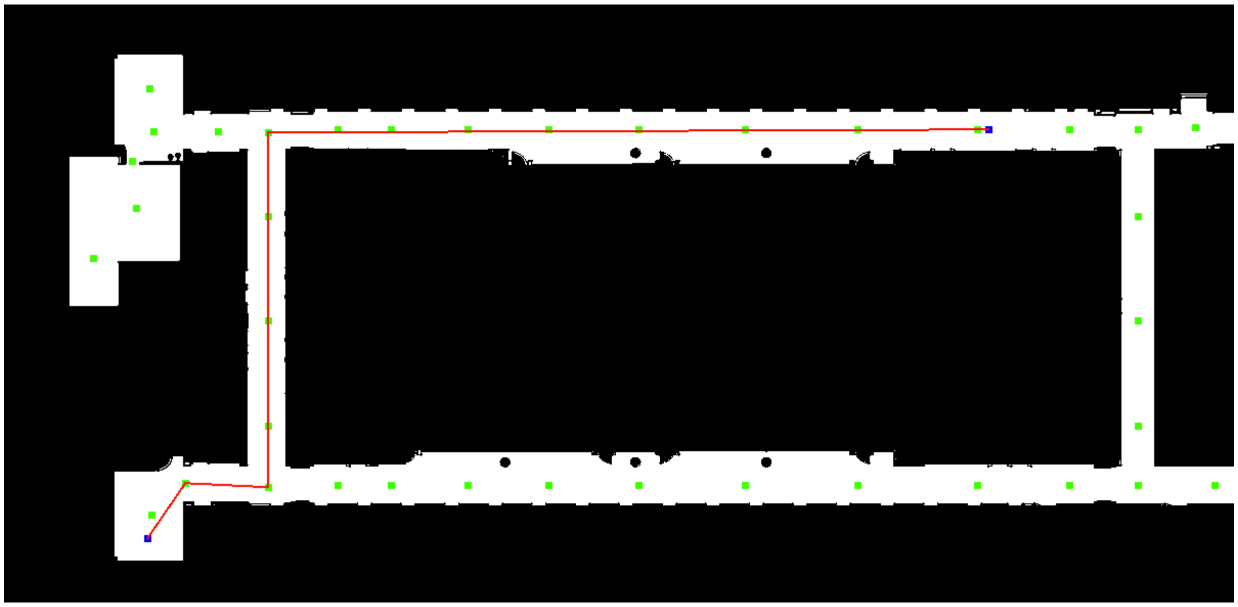
World obstacles were defined by outline edges of most “walls”, and simple a simple function was used to check for intersection. This worked relatively well, but in this scenario, we were left with a map with unconnected portions of the map around half of the time. While we could tailor this further by adding in specific points to the map, we found that there was no benefit by generating the map programmatically. Instead, we decided to make a relatively balanced map with specific points.

This method was expanded upon to make this roadmap a local and global planner. The function could take in a starting point (the robots estimated position) and a goal (the given destination point) and plan a path using the roadmap for the intermediary area. The only difficulty in this was that these two points had to be programmatically included into the map. Each call to the function would perform a copy on the roadmap, and perform the following function on both the start and goal points.

```
    For each point p in the roadmap
        If start connects to p without clipping world obstacles
            Add edge to the roadmap for this call
```

Furthermore, we performed a similar check on the goal and start, so that if their path was unobstructed, an edge could be created. This made it so that the roadmap “waypoints” were useful for rounding corners, getting through doors, and getting around local minima defined by the environment obstacles and boundaries, but the robot would take the path of a straight line if offered. Actual path planning worked by running **A*** search over the connected graph, though without the use of any distance heuristic, given that the roadmap itself was complete for the given free space. The map generation and planning was wrapped in one function, `getNextLocalWaypoint(start, goal)`, which could return the full path, or just the next destination in the path to make travelling more manageable with numerous smaller paths.

There was some difficulty in defining the world edges as not all obstacles are rectangular in shape. However, most of these boundaries and obstacles could be approximated with rectangular shapes, allowing for straight edges that could be used for path clipping detection. Missed corners (or underestimated corners) could be handled by potential field movement.



Roadmap Generated (blue points are examples of origin and goal)

Path Execution

Path execution and obstacle avoidance was handled using the same potential field technique explored in lab 2. The approach was modified to take in waypoints derived with the PRM A* pathfinding algorithm. Because the distance between these waypoints is relatively small compared to the overall free space, there was not expected to be any difficulty in encountering local minima. In the case of no localization, the robot returns to a safe wander movement pattern while collecting data until it is confident of its position again.

Real World Results

All of the previous results are in simulation. Given the fuzzy results of the simulation, not much was expected of the robot attempting to localize with noise and unaccounted obstacles. Taking one of the Pioneer robots around to see where it would go, it performed about as poorly as expected. With the exception of the area around the door leading to the stairs next to the CS lounge, the robot was entirely lost for the entirety of its trip outside. Despite attempting to increase the reliability of the algorithm by doubling the number of particles and modifying the threshold of when particles are culled, the robot could not build a cohesive representation of its area. This could have to do with the fact that the robot preferred to be in the corridors going across the building, which are harder to differentiate than other areas.

Issues Encountered

Identifiability of Hallways - Many hallways look the same in simulation. Even when the robot is moving forward through a hallway, localization may cluster in on a different hallway, or have the cluster move in the opposite direction. Programmatically figuring this out is difficult until new features in the sensor data are seen, so when the robot switches to directed movement, rather than wander, under a false localization, the result is either a counter-productive direction, or general uncertainty after moving towards the local goal.

Particle Confidence Threshold - The threshold value for our filter algorithm is a bit trivialized. Particles are easily removed and replaced on any particular iteration because there is no stored confidence that grows or diminishes over time. This makes incorrect clusters very easy to wipe out, but also makes good clusters susceptible to being wiped out due to misreadings or noise.

Map Backtracking - A more dense map means more manageable waypoints for navigation. However, connecting the start or goal node to too many nodes in the roadmap means that the A* search will often try to minimize a path by going past redundant waypoints.

Potential Field Inaccuracy - The potential field algorithm used for this project was not sufficient to apply our chosen path, primarily due to the fact unforeseen objects have a big impact on the obstacle vectors represented in the algorithm. In order to represent these things accurately, one should map at the same time using SLAM. These difficulties resulted in us reverting to a random walk for many of our localization tests.