Grid Filter Lab

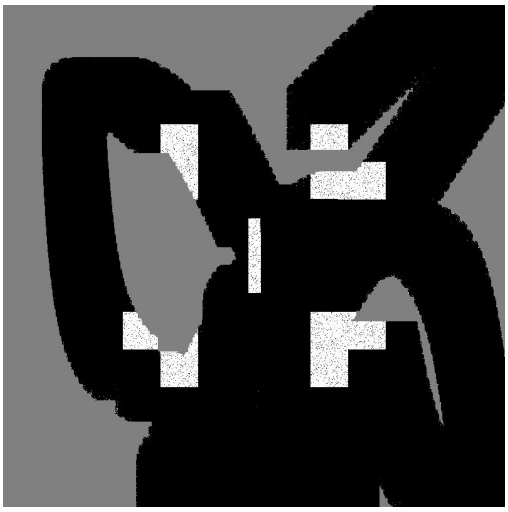
For this lab, I worked without a partner. I spent around six hours programming and testing the grid filter and waiting for my robot to explore a world.

In this lab, I learned how to use probabilities to build a reliable model of a robot's surroundings. I found the concept of Bayes Rule in this context particularly insightful, because it shows how to compensate for noisy sensors in static environments. As a hobby, I've played with environmental sensors (GPS, barometer) and a Raspberry Pi, and now I'm inspired to try applying Bayes' Rule to see how much more accuracy I can get from my sensors (when they're not moving, obviously)!

World Exploration

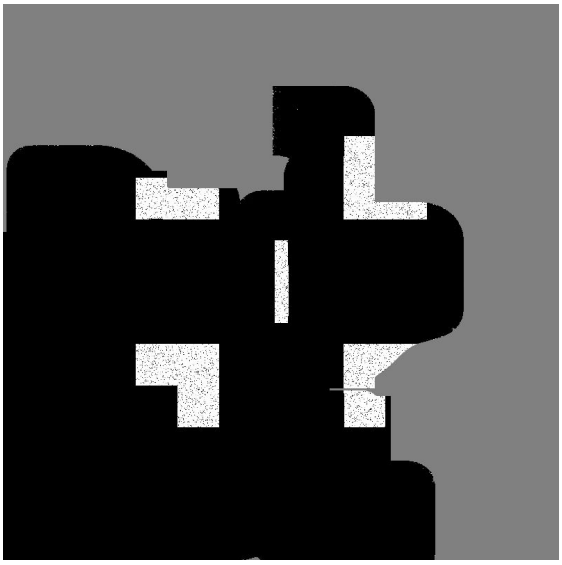
After implementing the Grid Filter, I tried two different algorithms for exploring the world. Originally, I used the concept of "waypoints" to encourage the robot to explore. Waypoints were generated by selecting a random point from the world that had not yet been seen through the occupancy grid. The robot continued to choose waypoints until all points in the map had been seen at least once. This approach offered correctness, but was really slow to explore for two reasons:

1. Regions near the edges of the map were unlikely to be selected until they were the only unexplored space left.
2. The robot often selected very distant waypoints and wasted a lot of time traversing through the center of the map, ignoring closer unexplored space.



The original random-path algorithm.
Movement is observably sporadic and non-optimal due to missed exploration opportunities.

I improved upon this algorithm by switching to a grid-based model, where the map was split into a 16x16 grid, with a waypoint at the center of each cell. Similar to the original approach, the robot explored by iteratively selecting waypoints. However, in this case it probabilistically favored the closest unvisited waypoint rather than a randomly-selected unseen point. This algorithm encouraged the robot to explore its immediate neighborhood before distant lands.



The improved, grid-based algorithm.
Exploration is done in contiguous blocks, leading to more thorough results.

In both cases, the tank selected a new waypoint when it spent more than 10 iterations in the same location. This helped the robot to “give up” on waypoints that it was blocked from reaching, but allowed it to return to those points later when it might have a better path available. One limitation of this approach is that it cannot finish if only one waypoint remains and some obstacle exists in the way.

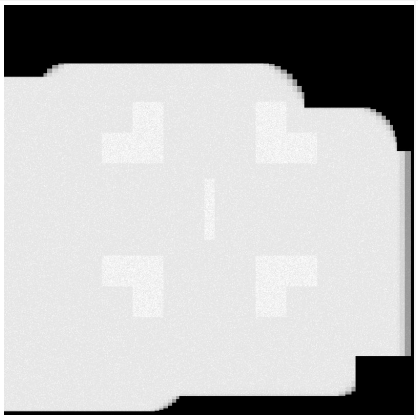
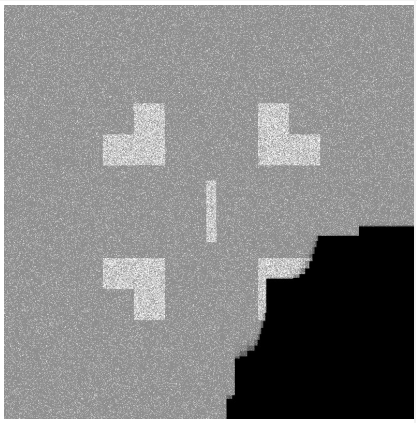
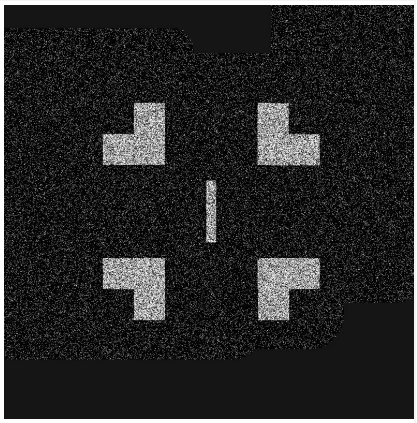
Parameter Variation

Waypoint Grid Size. When the grid size was reduced (i.e., using an 8x8 grid instead of 16x16, or one waypoint per 100px square) I found that the robot more easily became fixated on particular waypoints because the “gravity well” of a waypoint was much larger. This was fine except for when the nearest waypoint was unreachable, which led to situations where the robot got stuck. Additionally, the smaller grid size led to small unexplored chunks that slipped through the cracks.

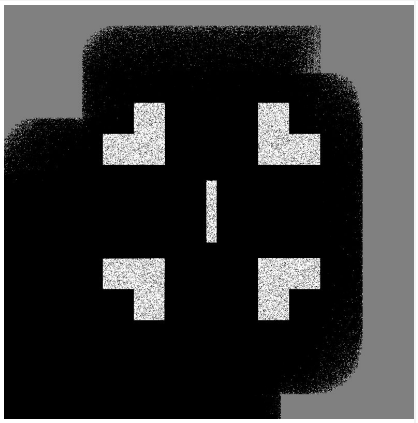
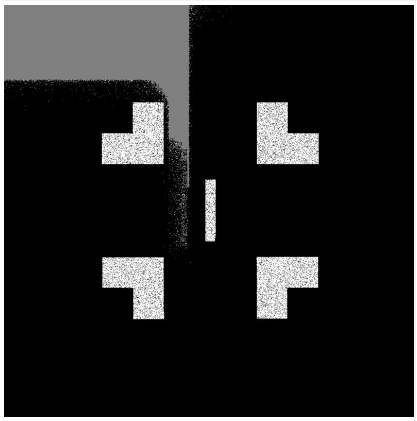
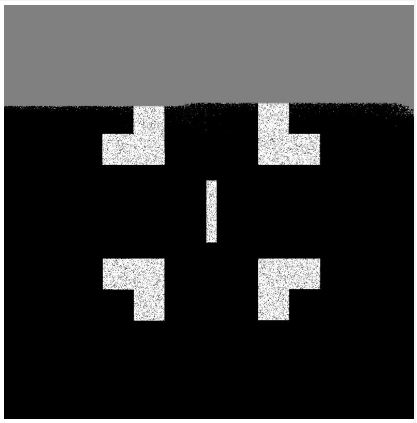
Occupancy Grid Size. Increasing the occupancy grid visibility to a 400px square eliminated the visibility problems associated with a small waypoint grid. Additionally, it led to far more thorough exploration and much higher confidences, as each individual pixel was seen far more often.

Furthermore, the robot fully explored the map an order of magnitude more quickly, in around 400 iterations instead of 4000. While ideal, it is not always feasible or physically possible to use such a large visibility grid.

Low True Positive Rate. The results above are for a true positive rate of 0.97 and a true negative rate of 0.9. With a true positive rate of just 0.5, the robot thinks that everything is an obstacle because the sensor essentially tells us nothing at all about the world—we might as well just randomly guess. Interestingly, with a rate of 0.51 the obstacles become discernable, though probability still suggested that everything was an obstacle. A rate of 0.55 was even better—potentially even usable, with some signal filtering—and 0.65 was downright helpful despite its high noise level.

		
$P(occ \mid hit) = 0.51$ $P(\neg occ \mid \neg hit) = 0.9$	$P(occ \mid hit) = 0.55$ $P(\neg occ \mid \neg hit) = 0.9$	$P(occ \mid hit) = 0.65$ $P(\neg occ \mid \neg hit) = 0.9$

Low True Negative Rate. When the robot isn't confident in sensor misses, there's a ton of additional noise. However, Bayes Rule takes care of this effectively, so varying this value doesn't matter very much if there are sufficient samples taken of the occupancy grid.

		
$P(occ \mid hit) = 0.97$ $P(\neg occ \mid \neg hit) = 0.51$	$P(occ \mid hit) = 0.97$ $P(\neg occ \mid \neg hit) = 0.55$	$P(occ \mid hit) = 0.97$ $P(\neg occ \mid \neg hit) = 0.65$

Moving Objects. Since the Grid Filter assumes in principle that its environment is stationary, moving objects probably present an insurmountable challenge. Likely, moving objects would get smeared across the model, similar to a fast-moving object in a still photograph. If many additional sensor readings were taken, as is the case in my implementation, then Bayes Rule would likely almost eliminate the moving object from the model. In other words, it would probably be treated much the same way as false-positive random noise.

Tournament Plans

The Grid Filter would be most helpful if used passively by tanks with other responsibilities occasionally taking sensor readings and collating results together from all available tanks. This would ensure that scout tanks don't become sitting ducks. Without the potential for evasive maneuvers, having scout tanks would be dangerous as the probability of being shot by enemy tanks is very high.

Computing the occupancy grid isn't nearly as expensive as I expected it might be, so if each tank took a sensor reading once a second or so we could quickly build up a map of the surroundings without impacting the tanks' ability to perform their other jobs while still collecting enough data for Bayes' Rule to eliminate most of the noise.

Once a valid obstacle map had been generated, one might consider disabling the Grid Filter calculations to save computation time and focus on more important jobs. Since obstacles are all stationary and the Filter doesn't help us with mobile objects anyway, there isn't a reason to spend time developing it once a usable map has been created.