Time

I worked alone on this lab. I imagine I spent around 20-25 hours.

Summary

I learned that the grid filter is actually pretty simple in concept. All the grid filter does is apply Bayes' rule for every iteration, using the previous posterior as the prior. The observation likelihoods are given to us for this lab, but in a "real-life" scenario the observations are found by conducting tests and evaluating measurements, which still is fairly simple.

One thing I learned about is the importance of the normalizer in Bayes' rule. My tanks didn't seem to be finding the obstacles very well, so I did some debugging and discovered that my grid had some illegal values (less than 0 or greater than 1). Looking at my code I realized that I had the normalizers switched: the normalizer calculated given observation=hit was in the observation=miss branch, and vise-versa. Once I switched those my tanks seemed to be much better at finding the obstacles.

Another thing that I learned is that this program has a lot of constants that can make a big difference. There are the constants for the PD controller, constants used to determine if a tank is stuck, the initial belief that each cell is occupied, the observation likelihoods (truepositive and truenegative), the probability at which you consider a cell occupied, how strong potential fields are and what their radius of influence is, etc. With some you just have to make tweaks until it seems to work right, but it's infeasible to try all possible combinations of all the various constants, so you have to think about how the constants relate to each other and approximate a reasonable value for each. It's impossible to get it perfect.

Searching the World

Searching the world was the most difficult part for me to get down. I spent a lot of time refactoring code from the previous lab to work better for this lab. I added a lot of features, the most significant being the history tracker, allowing me to command a tank to repel its own history and allowing me to know when a tank gets stuck.

To search the world, I use 4 tanks. Each tank begins in a corner of the world. The tanks sweep the column they start in (top corners go down, bottom corners go up), move over a column, sweep that column, and continue until each tank has covered the entire world. To do this, I use moving attractive fields. I use the tank's history tracker to see if it has been near the target. If so, I advance the target so that it continues sweeping. In some cases, like if there is an obstacle in the location of the target, the tank won't be able to get near the target. So, I also advance the target if the grid filter determines that the location of the target is a probable obstacle.

While searching, tanks run into each other, enemy tanks, and obstacles. I can tell when a tank gets stuck by looking through the tank's history tracker to see how much it has moved over the last *n* measurements. When a tank gets stuck, I do several things to try and get it unstuck. First, I create a strong attractive field in one of the four cardinal directions from the tank's current position. If the tank is still stuck after a number of iterations, I move the attractive field to a different cardinal direction.

Second, I command the tank to repel its own history. It sets up repulsive fields for the past ten locations, the most recent location having the strongest repulsive field. Third, I command the tank to repel probable obstacles. This, of course, only really helps in the case where the tank is stuck against an obstacle. Fourth, I have the world boundaries repel tanks, just in case the tank is stuck against a boundary. Once the tank gets itself unstuck, all of these extra potential fields are deleted and it goes back to simply following the sweeping attractive target. This doesn't work perfectly, but it always seems to work eventually.

Sensor Parameters

I initially thought that giving the occupancy grid a larger range would help a lot because the world would be able to be covered that much faster. However, querying for the occupancy grid and updating the grid filter gets more expensive the larger the sensor range gets. In my implementation, making the range of the occupancy grid too large makes a single tank consume too many resources, leaving the rest of the tanks wandering idly for a bit. If I were to multithread the client I assume that increasing the size of the occupancy grid wouldn't have as much of a negative effect.

Making the estimates noisier definitely affects my implementation. I typically run it with true positive as 0.97 and true negative as 0.90. Taking them both down to 0.75 makes my estimates a lot worse. Maybe playing with some constants (such as the probability at which a grid cell is considered an obstacle or the initial prior that I use for each cell) would improve things, but really the only way to get a decent estimate is to sweep the board multiple times to get several sensor readings of each cell.

Having an incorrect model for the noise also causes my implementation to be much less accurate. To test this, I hard-coded 0.97 and 0.90 as the true positive and true negative that my grid filter would use, but I ran the server with both values equal to 0.80. My program thought that 97% of the hits it was getting were true, but really only 80% were, so cells that weren't really occupied were often considered occupied until the area had been swept over several times, and even then I ended up with a lot of false alarms.

If the sensor could only detect moving objects, the grid filter wouldn't work. It wouldn't be able to find stationary obstacles, and it wouldn't really help with moving obstacles either. It might correctly read a moving obstacle in a certain location, but it would probably consider that location an obstacle for a lot longer than the obstacle was actually there. When using the grid filter, the previous posterior belief that a cell is occupied is used as the prior belief for the next time you get a sensor reading of that cell. This makes sense for a stationary obstacle, but not for a moving object.

Capture the Flag Tournament

One of the main problems my implementation has is that the four tanks fight over different resources. There are times when three of the tanks are spinning idly because the fourth one is trying to ask for the occupancy grid while trying to get unstuck, which consumes resources on both the client and the server. Because querying for the occupancy grid is expensive, I would not want to have four tanks sweeping the board to the find all the probable obstacles. Instead, I would dedicate only one or two tanks to the task. Instead of starting at the corners and sweeping inward, I would probably have them start at the center and spiral outward. Opponents' bases are generally near the border, so starting in the center would hopefully help me to develop most of the map before getting near enemy territory and risk getting shot.