Scott Crunkleton
Christopher Morgan
November 18, 2013

CS470_Grid_Filter

---

1. Scott Crunkleton – 20 hours
   Christopher Morgan – 25 hours

2. **A summary of what we learned:**
   - Grid filter Assumptions and Application
     - There were a few aspects of grid filters that this assignment really helped us to understand. One was how grid filters gets started. It starts with an initial view of the world, in other words, “Given no data at all, what do I believe the world to look like?” For example, in our project, what is the probability overall that any given space in the world is occupied? It made sense to use that the world is most likely more free space than it is obstacle-occupied.
     - Another aspect was what grid filter stores in it world representation. In our case, it was the idea that regardless of the observation we always stored the probability that a given space was occupied.
     - We also gained insight as to how and why grid filter differs from Bayes filtering. Given grid filters’ idea that the contents of a particular space never changes, the definition of bel bar with respect to a state at time t simplifies to just bel with respect to a state at time t-1. We used this by multiplying our current world probability by what the likelihood was of an occupied state given our current observation. In other words, we learned how to update a grid filter's cell's probability.
   - Potential Field Craziness
     - We learned that calculating potential fields around obstacles can be difficult when we aren’t sure of the true state of the world. Originally, our agents were given maps of the obstacles. The obstacles were square in nature and we made the simplification of treating the obstacle like a circle that encompassed the actual square. During this lab, we didn’t have access to such information. Now, we treated each obstacle pixel like an obstacle. This exploded the number of obstacles from a handful to hundreds of circles. We calculate tangential and repulsive fields every ‘tick’ for every tank. This is pretty expensive and caused some latency issues. In summary, we learned that how we represent the world is important.
   - Stacking by Design
     - We learned that sometimes one approach is not simply the best. Sometimes it is better to combine approaches. For example, don't just use a simple lawn mowing technique. Use a combination of lawn mowing, random search, and random unexplored search. This combination is quite exciting.

3. **A discussion of how you got your tanks to search the world (how many tanks your code could handle reasonably, whether or not you did anything tricky like looking at neighboring cells in the grid, how you moved your tanks around, what you did when tanks got stuck, etc.)**

Our search pattern of the world followed a simple overlapping lawn mowing style. We divided the world into slightly overlapping columns given the number of tanks that would search the world. Our tanks would then simply mow the world according to set of column waypoints. Once the end of the world was reached for a tank it would cycle back through its waypoints.
We did two tricky things in respect to obstacle finding. First a point was only ever given to obstacle consideration if 90% of the pixels within a 5x5 pixel box around it were obstacles. Then if a point passed this 90% rule it was only added as an obstacle point if none of its neighbors within a 7x7 pixel box were already obstacle points. This allowed us to treat individual pixels as circular obstacles and to reduce the shear number of obstacles.

With this configuration we were able to handle anywhere from 3 to 7 tanks, however, for more stability we settled on 3. Also, we did not do anything tricky when tanks got stuck, but proposed the solution of a waypoint time out. Meaning that if a tank failed to reach a waypoint in a certain amount of time, it would just move on to the next way point. Again, we did not implement this. Our crazy potential field madness was enough to keep them reasonably unstuck.

As mentioned before, the large number of pixel obstacles created latency when calculating the potential fields. If we were to change the calculation of the repulsive and tangential fields to occur only once for every time that a new pixel obstacle is added, the calculation time would drop drastically. After that, we would be able to send more tanks out.

4. **A discussion of what happens when the sensor has different parameters (ex. it has a bigger range, it returns noisier estimates, you have an incorrect model for the amount of noise it returns, it only detects moving objects, etc.) - some of this you can test by varying the parameters you pass to bzrflag, and some you may have to just speculate on (like the moving objects bit),**

Sensor Type	**What would happen...**
Bigger range	In this project, we would be able to discover more of the world quicker and in fewer passes. Our tanks would also run into fewer objects. With a 100x100 range our robots would often run head on into an object before it was classified as an object. It was only after a tank collided with the obstacle that the obstacle would then be classified as an obstacle and begin repelling our tanks. Therefore, with a large field we would be able to more quickly and appropriately act on newly

	discovered obstacles. The time that it took to process each 100x100 observation was insignificant. We should be able to increase the range without too much of an effect. However, if this range got really large, we could begin to see some calculation latency.
Noisier estimates	Our free space and obstacles would both become more speckled. We would become less sure of true free space and obscured space. It would also update the pixel probabilities slower and, therefore, it would require more observations to accurately classify a pixel as an obstacle or as free space. Since it takes longer to classify a pixel, our algorithm might need to use a smaller obstacle-neighbor percentage when classifying a pixel as an obstacle. This is because a pixel's neighbors would also take longer to be classified and therefore, there might be fewer neighbors classified as obstacles. Setting a smaller obstacle-neighbor percentage would help us identify obstacles faster, but we might get a few more false-positives.
Incorrect model	If we believe that our model is more accurate than it really is, we will end up with a very spotty and inaccurate guess of what the world looks like. We will more quickly mark things according to the sensor only to quilty try and reverse them if the reading changes. We would also have a very hard time recognizing obstacles, because we would quickly clear our occupied probability when a missed detection occurred. If we believe that our model was less accurate that it actually is, then we would have a very splotchy frontier, but each pass would consistently bring the occupied probability closer to its correct value. We would just require more passes to get a clear picture of the true world.
Moving objs only	Things would show up on our probability map as an obstacle only to be removed with an additional observation. It would work but it would not give us an accurate picture of what the current state of the world actually is, i.e., freeze the world and we would get an accurate picture. Grid filter is used in worlds with static state. We would not be able to get an accurate representation of the world because the state transition probability in grid filters assumes that state does not change. If obstacles moved in our world, we might label points as obstacles at time t which would not be obstacles at time t + 1. Furthermore, our algorithm 'burns-in' obstacles as we classify them. In other words, once we classify a pixel as an obstacle, it stays that way. So, for our algorithm, we would have 'shadows' of where obstacles had been even though they might not be there in actuality.

5. **A discussion of how you would apply the grid filter in the capture the flag tournament,**

especially given the fact that you might get shot if you spend too much time wandering around the opponent's territory without having a good map, and that asking for the occupancy grid costs you computation time.

We would send a few tanks out to explore the map. Once we have a decent enough picture of the map (i.e., we know where the enemies flags are) we would stop trying to explore the map and stop requesting occupancy grid observations. From that point, observations would only be asked for if we didn't have a good picture of a tank's current location or if we cannot locate an enemy's flag, we would then only explore and process observation to capture.