Evaristo Koyama

ek4ks

Basic Path Finding (Certainty)

For the assignment, we had to use A*. A priority queue is necessary. It consists of possible paths and is ordered by the heuristic, which is the size of the path plus the minimum number of steps to reach from the last point to the end point assuming no obstacles. Since we can go diagonally, it is just the max between the x distance between the last point and the goal and the y distance. There is also a hash map containing the minimum distance to reach a point. This ensures that we always choose the shorter path to a target point. We pop the current best path and generate possible next steps. We remove the possibility of picking the current position, a position already in the path, and out of bound positions. Removing the out of bounds positions reduces the number of pings. If one of the next steps takes us directly to the goal, then we tell the robot to move along the path. Otherwise, we check the path. If the point is new, we ping the map. If the position is valid, then we add it to priority queue and keep track of the current minimum distance to the point. If the position is invalid, then we mark it as an invalid position. If it exhausts all possible paths, then we assume the destination cannot be reached.

The algorithm finds a path of size 38 with 204 pings given the input found in Collab. (input1)

The algorithm finds a path of size 190 with 378 pings given an input of a winding path. (input2)

The algorithm finds a path of size 19 with 112 pings given an input of open world. (input3)

The algorithm finds a path of size 30 with 355 pings given an input of an open world with a few obstacles close to the goal. (input4)

Based on number of pings and path size, we can see that the algorithm has a lot of trouble with input4. The issue is that it would first head for the goal, but when it encounter the obstacle, it has to go around. It finds paths to points around the obstacle until it finds an opening. Winding path worked out better, but this is because it pretty much checked the whole map and the path is large.

Adding Uncertainty

When uncertainty is added, I mainly added three things. One is a hash map that keeps tracks of points that are certainly valid or invalid. After finding a path, the robot will start moving. It will not necessarily reach the goal, and whether the robot moved to a certain position is valuable information for the next run. Second is a bound on the size of the path. The world is more uncertain the farther away we try to ping. We can try to find a path directly to the goal, but this would be highly unlikely. The initial length of the path is the average between the number of rows and columns the world has. Maybe we will get lucky or the uncertainty is low. We change the bounds based on the size of the paths we take. This gives an idea about the uncertainty of the map. We don't want to path to be super short, so there is a lower bound on the length of the path. If it is too short, then it could get stuck for a while. Third is a check to see a path contains any new points. This is to prevent endless backtracking. If the path is stale, find a longer path. The algorithm is pretty much the same. It keeps trying to find the best path until it reaches the end.

It would not be stuck forever since the robot gains more knowledge of the map, which would increase the length of the path of each subsequent run.

On input 1, the algorithm reaches the goal in about 75 moves. Pings are around 500.

Input 2 is the worst. The algorithm reaches the goal in about 2000 moves. Pings is usually around 25000. The algorithm has issues navigating the path. It needs to go away from the goal about half the times, which makes the robot back tracks.

Input 3 is the best. The algorithm reaches the goal in about 21 moves with about 120 pings. This is very close to the result when the map was certain.

On input 4, the algorithm reaches the goal in about 350 moves with about 3500 pings. If it is unlucky, then if finds the goal above 700 moves and 7000 pings.

The number of moves is reasonable except in the case of input 2, but the algorithm has a lot of pings. The main reason being that the only times it confirms an obstacle is when the robot hits the obstacle. Every time it tries to find a new path, it has to ping all the obstacles again. The number of moves becomes more unreasonable if the algorithm has to take a path that goes away from the goal. There is an additional cost that could be considered, the cost of collision. This algorithm does a good job minimizing the number of collisions. One is that it never collides with the same obstacle twice. We keep track of all the obstacles we collided with. Two is the algorithm is very cautious. It will usually avoid the obstacle close since the robot is more certain with detecting obstacles nearby. If an obstacle is detected when there is not, then the robot does not necessarily take that path. If an obstacle is not detected when there is, this usually means the robot is going to take a long path. Every run, there is at most one collision.

Removing the check for path inside the path finding algorithm slightly decreases the number of moves, but greatly increases the number of pings. This increases the chance for finding an alternate path since the priority queue is not emptied. This does increase the number of pings since with each reset, we need to ping the map all over again.