Path Planning for Disk Holonomic Robots

Akash Desai and Deep Patel

September 2018

1 Question 1

The A* algorithm depends on Manhattan-diagonal distance for its heuristics, while FDA* uses Euclidean distance. Manhattan-diagonal distance allows movements in eight directions compared to the 4 directions allowed in the A* algorithm that relies on Manhattan distance. Further, the FDA* algorithm allows for free motion in all 360 degrees, assuming a collision free path.

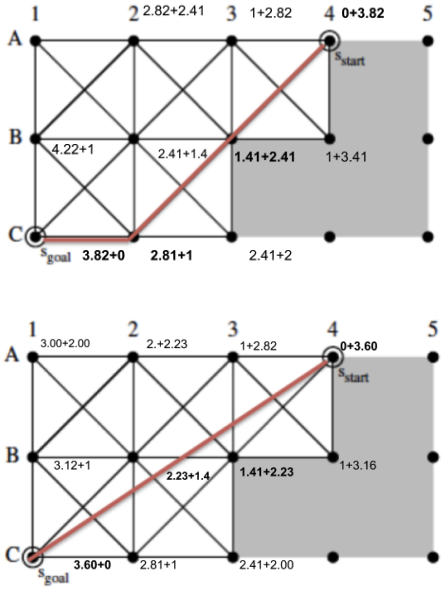

Figure 1: A* and FDA* trace with g and h values

Closed list	Open List
A4	B3, A3, B4
B3(p=A4), A4	C2, B3, A3, B4, B2, A2, C3
C2(p=B3), B3(p=A4), A4	C1, B3, A3, B4, B2, A2, C3
C1(p=C2), C2(p=B3), B3(p=A4), A4	B3, A3, B4, B2, A2, C3

Closed list	Open List
A4	B3, A3, B4
B3(p = A4), A4	B2, B3, A3, B4, C2, A2, C3
B2(p=A4), B3(p=A4), A4	C1, C3, A3, B4, B2, A2, C3
C1(p=A4), B2(p=A4), B3(p=A4), A4	B3, A3, B4, B2, A2, C3

Figure 2: Open and closed list

2 Map representation

A few consideration that we kept in mind while sketching the map representation were:

1. the size of the robot
2. finding the shortest path
3. finding the collision free path

As the robot navigates through the vertices of the graph, we decided to start our tests with the block sizes as the radius of the robot. As the diameter of the robot lies from 0.34 - 0.35 meters, we figured .40 meters is suffice for a conservative estimate for the robot diameter, modelled as a sphere. In order to map the polygon we started exploring the point in polygon method, which relies on the Ray casting algorithm, however, although it provides a good estimate for a point inside a polygon, it excluded the points on the polygon. To account for this, we later diverted to using a "Intersection of two paths method," where one path is the polygon and the other is a single cell in the grid map. Using this we were successfully able to map blocked and unblocked cells to create our binary grid. After representing the obstacles, we grew the boundary of the obstacles by approximately the radius of our modelled TurtleBot, 0.2 m, so that we can now treat the robot as a point. Then later we transformed this cell map into a

(n+1) by (m+1) vertex map of nodes which keep track of many things like x,y position, heuristic, g value, parent, and unblocked neighbors.

The grid based representation allows us to have a short and obstacle-free path because we do not allow the path to intersect blocked cells. This discretization of the map allows us to make binary decisions, and by keeping the resolution high, we are able to get as close to the optimal shortest path as computationally possible with this approach. In essence, we are able to have a short path due to the resolution of the map painting a precise representation of the polygons and as a result, the path vertices are as close to the obstacle as possible.

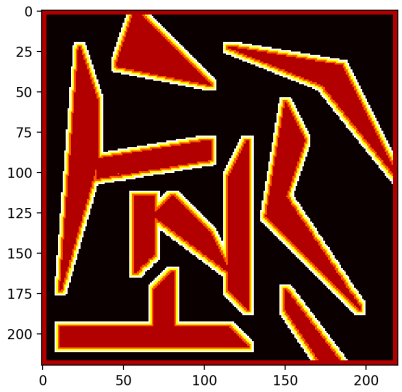

Figure 3: Grid-Based Map Representation

3 A* Algorithm

Why does A* find the true shortest path?
We also know that the heuristic of A* is consistent and admissible and so we can further say that the path found to the goal state is optimal, because for optimality to hold heuristic must be consistent considering that the cost of our path are always positive. We can see that the A* algorithm holds to consistently constraint $g(s, sprime) + h(sprime) => h(s)$.This can be seen by traveling from A4 to B3 and the rest of the paths. For our heuristic function, we used the maximum possible diagonal distance + the Manhattan distance for our path; the formula is shown in the function below. A* finds the shortest distance on the map you provide, meaning if the resolution is not fine enough, then the shortest path on this map and the true shortest path may differ more than just slightly. This combination of Greedy Best-First-Search (favoring the vertices closer to the goal) and Dijkstra's Algorithm (favoring vertices closer to the start) makes A* yield a shortest path in a time efficient manner. A* algorithm builds on Dijkstra's algorithm for path planning. In Dijkstra's algorithm we

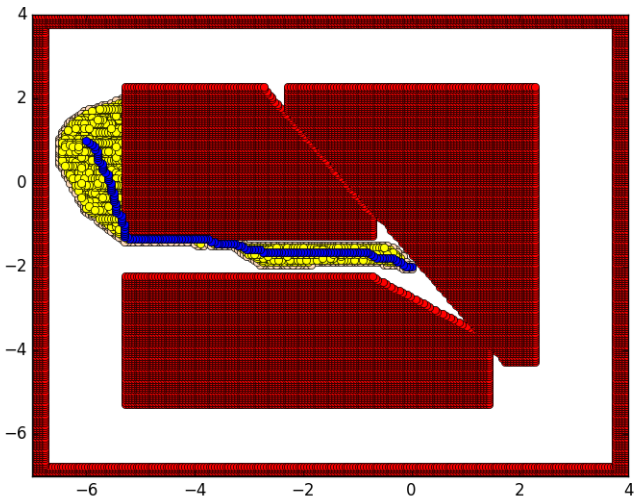

Figure 4: A* running on map 1, 373 nodes discovered

use the evaluation function f(n) = g(n) which gives priority to nodes that have traveled the least distance so far from our open-list and adding to the closed-list. One of the disadvantages of having the Dijkstra's algorithm for path planning is we explore the whole map in symmetric layers before getting to the goal node. Thus, it is not the most efficient algorithm for our use case but it does find the shortest path. This is where A* shines the most. By adding a heuristic function, A* narrows the exploration region from being uniform/symmetric to making it in the rough direction of the goal. So, for A*, f(n) = g(n) + h(n). If h(n) = 0 it turns into Dijkstra's Algorithm. If h(n) is lower that or equal to the cost of moving from n to the goal, A* is guaranteed to find the shortest path. The lower h(n) is, the more nodes get expanded, making the algorithm slower. Although h(n) has to be admissible, meaning it cannot overestimate the distance to the goal, but it should not be so conservative that it turns into Dijkstra's Algorithm, where there is no notion of narrowing your search to the goal.

$h(s)=\sqrt{2}\Delta min(|sxsxgoal|,|sy-sygoal|)+max(|sx-sxgoal|,|sy-sygoal|)min(|sx-sxgoal|,|sy-sygoal|)$

4 FDA* Algorithm

FDA* builds on top of the A* algorithm, but allows paths to go in any direction, unlike A* which is limited to 8 directions (up,down,left,right, and the four diagonals). Through this, FDA* can shorten the distance between a vertex's parent and a subsequent vertex by creating a direct link between that vertex

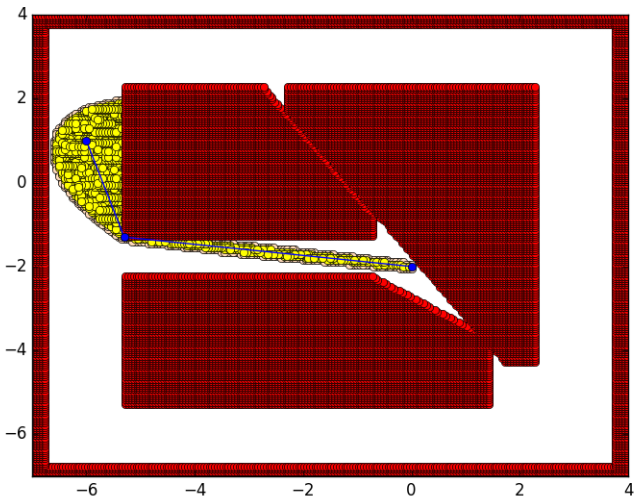

Figure 5: FDA* running on map 1, 355 nodes discovered

and the successor. FDA* relies on two paths: (1) is the free direction path between two points and (2) the path considered by the A* Algorithm in straight lines (in 8 directions). If the free direction path between the two points is not blocked by an obstacle, then due to the triangle inequality property, we know that the distance yielded by the free direction path will be shorter than the A* line path used for travelling from the parent to the subsequent vertex. We also know that the heuristic of FDA* is consistent and so we can further say that the path found to the goal state is optimal, because for optimality to hold, the heuristic must be consistent considering that the cost of our path are always positive. We can see that the FDA* algorithm holds for consistency constraint $g(s, s') + h(s') \leq h(s)$. This can be seen by traveling from A4 to B3 and the rest of the paths. FDA* relies on a path shorter than A* and because of the triangle inequality property we can say that FDA* finds the shortest path for a given map. This is further seen in the data that we collected, even though the difference in distance is small, we can see that FDA* gives a shorter path as seen in Figure 5.

5 Optimization

In order to optimize the algorithm we used a heap (as a priority queue) for the open-list data structure. Due to its O(log(n)) complexity for insertion and constant time O(1) for obtaining the smallest f(n) value, we dramatically decreased the number of comparisons for the addition and removal of nodes to the

open-list. This compares to the linear time O(n) addition and removal of nodes that any other data structure would give.

Another optimization we made was during the grid-based map representation, where we configured the resolution for the map in order to yield the shortest path but also be able to compute it in a reasonable amount of time. We did this by making it dependant on the input map boundary size, but we also set an upper limit. Through experimentation we figured out that a resolution over 200 x 200 and under 300 x 300 gave the best balance between fast run time and accurate shortest path. Experimenting results over 300 resolution took longer to compute, thus we settled on having a resolution of between 200 and 250 based on the boundaries of our environment.

6 H-values for A* and FDA*

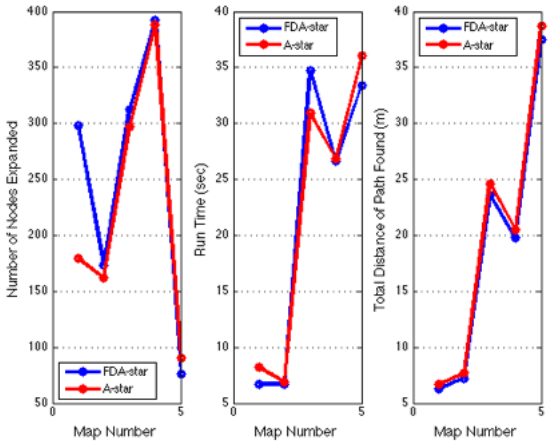

Figure 6: A* and FDA* comparison and observation

H(n) or heuristic of a node is the cost of of moving from n to the goal, if we have an accurate measure of h(n) the our we will only follow the best path and never expand any other nodes. However, we generally do not know the distance for the unexplored nodes and so h(n) is generally an estimate. On the other hand an h(n) = 0 is not any useful to us because of the algorithm expanding every node possible.The heuristic of A* used the Manhattan diagonal distance and the FDA* used the euclidean distance. The heuristic of both algorithm are admissible, because they both heuristic estimates a lower bound for the distance from s to goal. As the heuristic of A* and FDA* is lower than or equal to the cost of moving from S to goal, we are guaranteed to find the shortest path with some time expense. Following that, we also that the heuristic of both are algorithm are consistent as proven earlier. Both admissibility and consistency

are good marker for having good heuristic function. However, lets say our goal was not to necessarily find the shortest path, then we can use Adaptive A* which updates the h-values from previous searches h(s) = g(goal) - g(s), thus transforming consistent heuristic values into informed consistent heuristic values. This method again focuses on speed rather than finding the shortest path. So based on our use case, we should update the heuristic accordingly. For now, FDA* and A* give us the shortest path .

7 A* vs FDA*

A* does guarantee the sequence of f-values to be monotonically non decreasing because f(s) $\leq$ f(s'). We can see this in the A* trace we did for question 1. The f-value from the start was 3.82 and any following F(s') was either equal to that and greater than f(s start) due to an obstacle. This is visible in the first question; during the FDA* trace, we noticed that the F(s') of B2 would have less than F(s) of B3 if the goal node was not equidistant (B2 lying in the middle of start and end node). So if the goal node was -2 units in the x direction further than the C1 vertex, we would have gotten an increase in the h(s prime) = 2.23 and h(s) = 3.16, giving us f(s') = 4.47, which is less then f(s) = 4.57. Thus the f-value of FDA* can decrease over time in contrast to that of A*. Additionally, our heuristic is admissible and consistent. It is admissible because it is never greater than the actual cost. It is consistent because if we check any node we can agree that: $g(s, s') + h(s') \leq h(s)$. For example, A4(s) to B3(s') gives us 1+1.14 = 2.14, another example is B3(s) to B2(s') gives us 1.41+2.23 = 3.64.

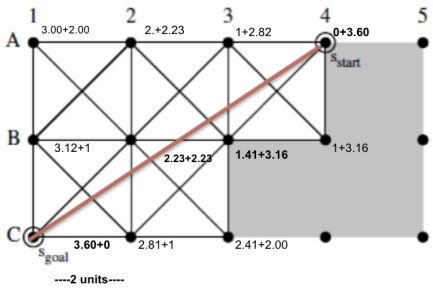

Figure 7: A* and FDA* trace with g and h values

8 Visibility Graph

The visibility graph technique is used to build a graph by adding the start and goal vertices along with all the polygon vertices and by joining any two vertices

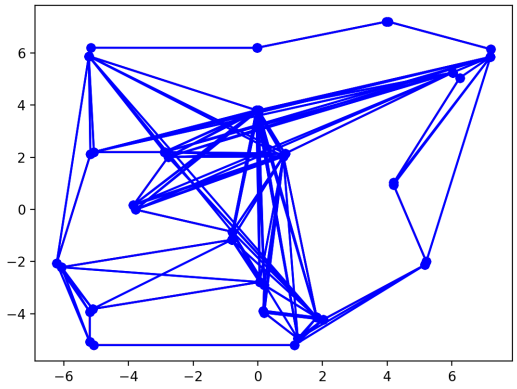

Figure 8: Visibility Graph

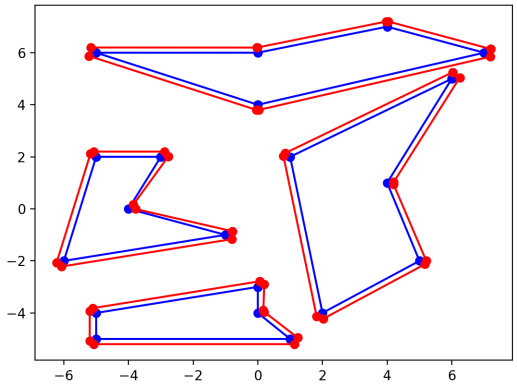

Figure 9: Obstacle Growth

that can be seen from each other. I.e. if an obstacle (polygon) is not blocking the straight line between two points, then we can successfully make a connection between it. Doing that for every point on a map creates a graph of connections that can be used to travel from one point to another. We decided to build a reduced visibility graph by detecting all concave vertices and ignoring them when creating our edge list. Two examples can be seen in Figures 10 and 11. To build our reduced visibility graph, we used a naive approach and checked if our line segment was intersected any other line segment of a polygon on the map. If it does not then a connection was added, otherwise it was not. We expanded the polygon by the radius of the turtle-bot (.2) towards free space to

have a collision free path for the TurtleBot. One of the advantages of Visibility graph over grid based representation is its faster run-time for algorithms such as A*, because there are much fewer vertices and edges to consider. One of the disadvantages are its complexity; it takes O(N^2) to check if one possible edge intersects with any of the polygon edges (which is directly proportional to N), taking O(N^3) to check all possible edges. This complexity can be reduced with the rotational sweep algorithm, which takes $n^2 * log(n)$ complexity for building the visibility graph.

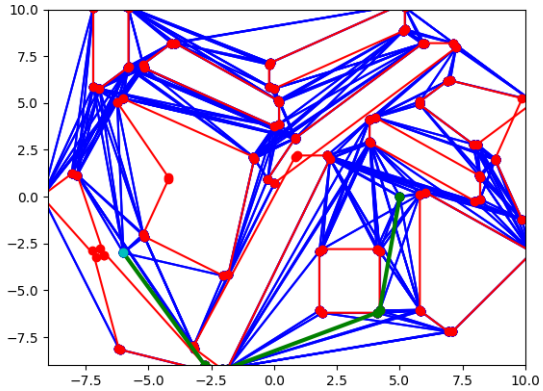

Figure 10: Reduced visibility graph for map 5, start/goal pair 6

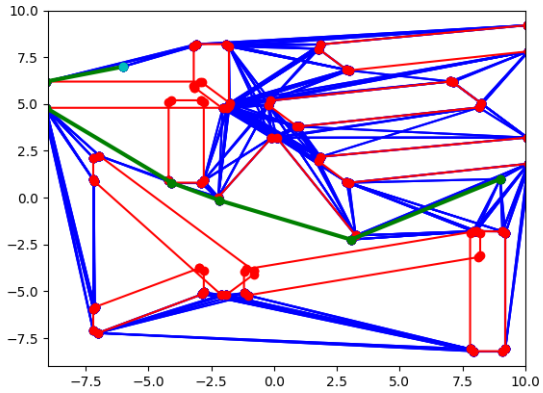

Figure 11: Reduced visibility graph for map 3, start/goal pair 10

9 FDA* vs A* on Visibility graph

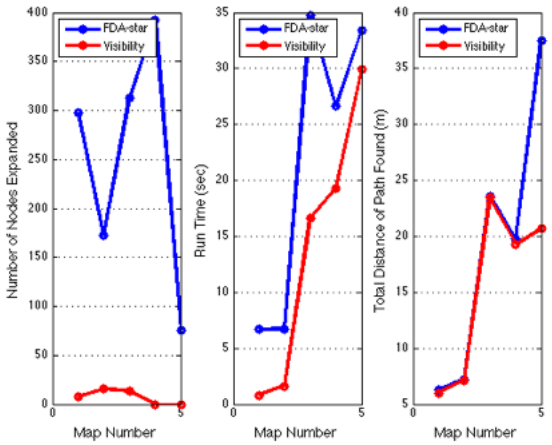

Figure 12: Data analysis of A* on visibility graph vs FDA* on cell decomposition

The FDA* algorithm is similar to running A* on visibility graph because it yields a similar route for our start and goal node, namely, there is "free direction" in both cases. Nevertheless, the FDA* algorithm gives us a slightly longer paths than A* on the visibility graph, as shown in Figure 10. The only drastic difference in path length can be seen for map 5. After analyzing the maps, we found that in the grid based FDA* approach, a very narrow potential path between two obstacles became blocked after the obstacles were grown. In the visibility graph, however, this path remained open because the grown of the obstacle is less discrete. This allowed the visibility graph approach to find a much shorter path that FDA* did not see as available.

Aside from this edge case, the visibility graph consistently gave slightly shorter paths. This is due to the fact that the resolution of the grid map cannot be infinitesimally small because of computational limits. If this were the case, both methods would yield the same length paths (given the assumption that in both approaches the obstacles are grown in the same manner). The time complexity of running grid-based FDA* is $O((|V| + |E|)log|V|)$ or $O(b^d)$. The time complexity of grid map is o(x dimension * y dimension) - $O(n^2)$. The time complexity of A* is $O((|V| + |E|)log|V|)$ or $O(b^d)$ The time complexity of visibility graph is $O(points^3)$. An algorithm that is dependent on h(n) to be really accurate, solves for the best path in $O(E)$ time. Basically, the closer h(n) is to h*(n), the more the algorithm is biased to take the optimal path. As the map reduction technique makes a grid of m times n which is dependent on the size of the world instead of number of polygon you have, it takes more number of comparison to make the grid than a visibility graph with a limited number of obstacles. As in our case, our map sizes were limited, but we saw the

dimensionality reduction really affect the time for algorithm's to run, because the size of the vertices and edges became so high. However, with the same obstacles, in the visibility graph approach, we saw that we had a lot fewer number of comparison with naive approach and even less with rotational sweep. Then further running A* on visibility graph we get a clear insight on where we are going. So we see A* doing well starting at start node and going to the goal node by jumping from node to node without expanding any other navigation points. This is much faster than looking for a path in the grid even with FDA*.