NTNU

Kunnskap for en bedre verden

TTK4192 - Mission Planning for Autonomous Systems

Computer Assignment 1

Author:
Daniel Borger Larsen

Date: 27th February 2024

1 Voronoi diagrams

(a)

A Vornoi diagram is a partition of a plane into regions. Each region is defined by a seed, where all points in the diagram closer to a particular seed than any other is part of that seeds region.

(b)

A: Vertex

B: Voronoi region

C: Voronoi edge

D: Generator point / Site

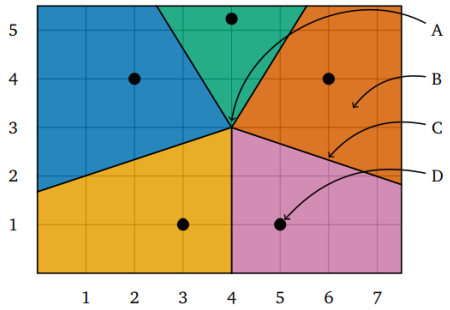


Figure 1: 1.b

(c)

Its not perfect, as its drawn by hand, but it should be approximately correct.

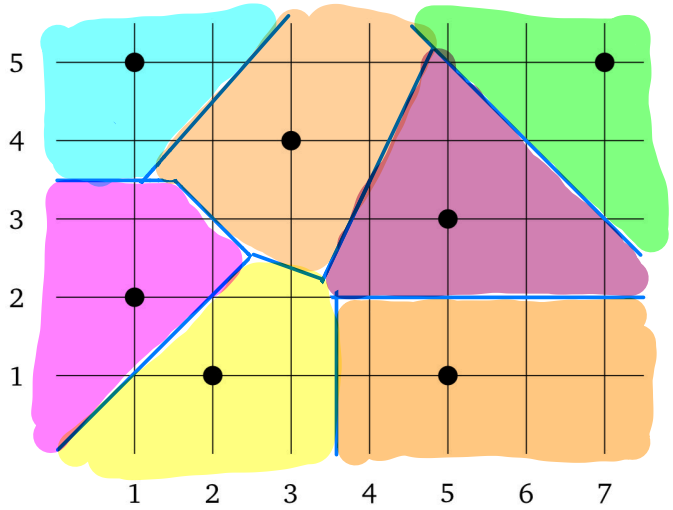


Figure 2: Points for 1.c

2 Fortune's algorithm

(a)

Fortune's algorithm is a fast an efficient algorithm for computing the Voronoi diagram given a set of generating points. It is a sweeping algorithm, where it uses a sweeping edge that travels over the space, and each point passed by the sweeping line generates a parabola where the distance from the generating point to any point on the parabola is equal to the shortest distance to the sweeping line. The intersections between the parabolas generated by the generating points creates the edges in the Voronoi diagram. The parabolas construct a line between the sites, and the sweeping line called a beach line.

Site events is when the sweeping edge has passed over a site, which add another parabola to the beach line. Edge intersection is when two parabolas intersect, adding another point to the edge between the sites associated with the intersecting parabolas. Circle events occur when three consecutive sites in the beach line form a circle which is empty of other sites. This event corresponds to a potential point where the Voronoi diagram edge may terminate.

(b)

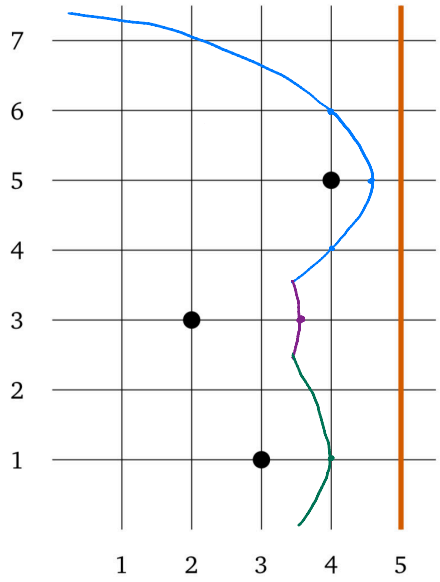


Figure 3: Points for 2.b

Figure 2: Approximate drawing of the beachline

3 Connecting waypoints

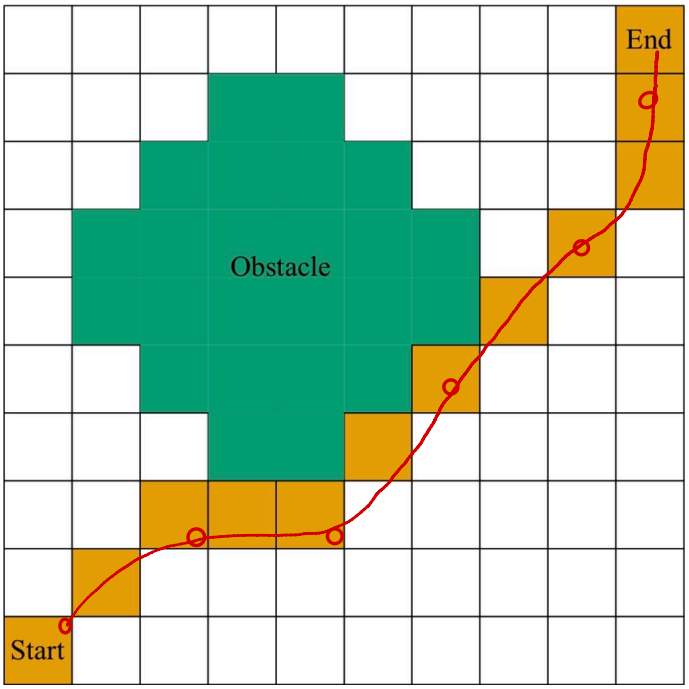


Figure 3: Connecting waypoints using splines

4 Graph creation

The weights are found by using google maps to measure, and nodes are placed in intersections that provide new paths. The script finds the path marked in green, which had a cost of 1685 and is the shortest path. The script is called `graph_creation.py` and is found in the `graph_search` folder.

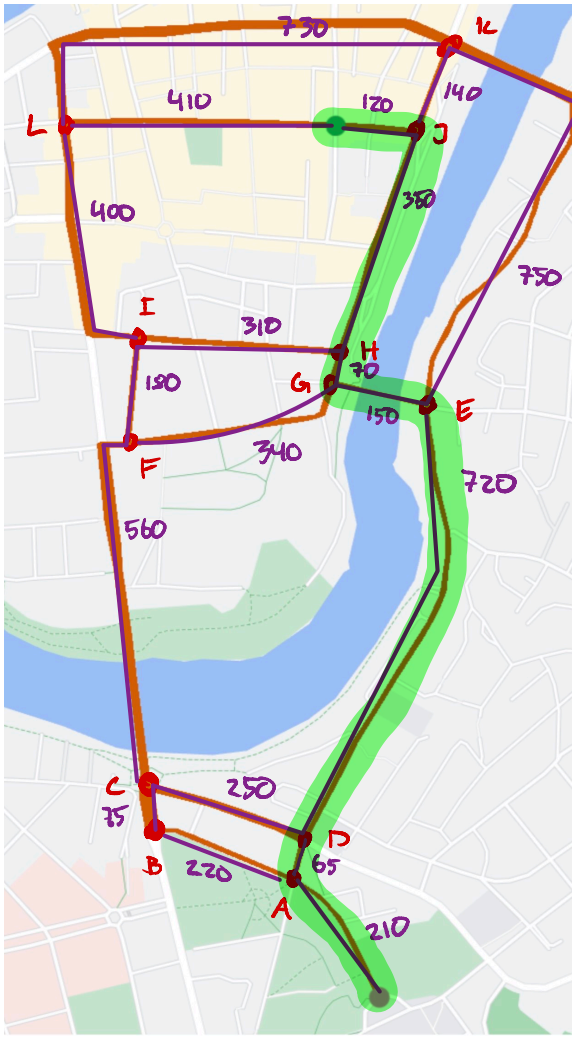


Figure 5: Map for 4.a

Figure 4: Illustration of the distances

5 Dijkstra's algorithm

(a)

Dijkstra's algorithm is a graph search algorithm that finds the shortest path between all the nodes in a weighted graph. The algorithm guarantees to find the shortest path in a graph with non-negative edges.

Dijkstra's algorithm assumes that there are non-negative edge weights. It also assumes that the graph is connected, i.e. that there is a path between every pair of nodes in the graph.

(b)

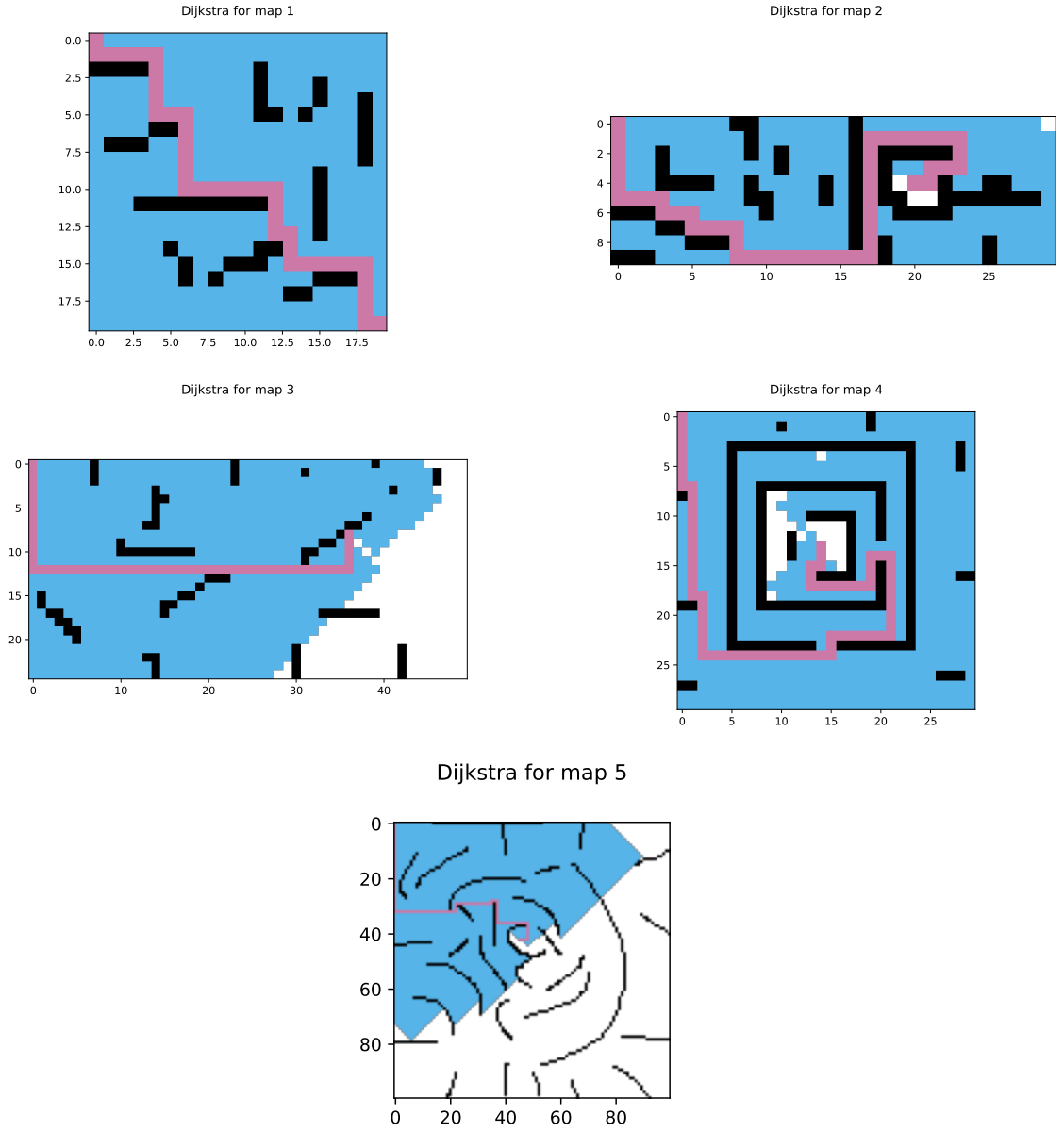


Figure 5: Solutions for the Dijkstra algorithm

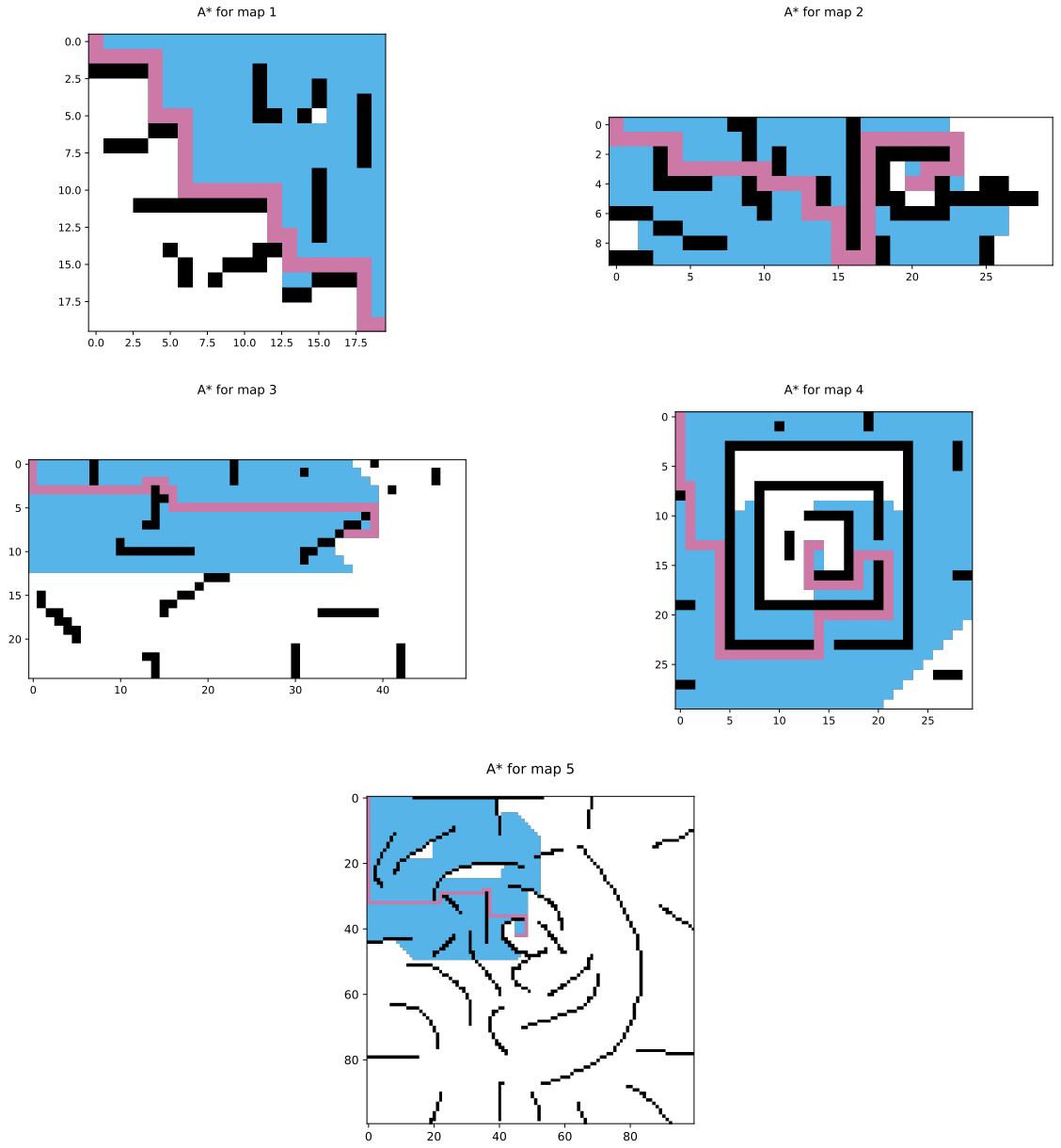
6 A*

(a)

A* is a path finding algorithm that finds the shortest path from a starting node to a goal node in a weighted graph. The essential part of the algorithm is using a heuristic, like a f.ex. a distance measure, to construct its priority queue. The algorithm combines the advantages of Dijkstra's algorithm with a heuristic function that guides the search to the goal more efficiently.

The algorithm makes the same assumptions that Dijksta's algorithm does, like non-negative edge weights, connected graph, and that there is a single source, and a single destination. It also makes some assumptions about the heuristc, namely that it is admissable, i.e. that it never overestimates the cost to reach the goal.

(b)



(c)

The implementation of A* is better because it performs better. This will always be the case, assuming that the heuristic is reasonable. This is obvious from the plots, as the A* algorithm need to cover less of the space in order to find the destination. This makes perfect senses, as A* is essentially the same has Dijkstra's algorithm with the added benefits of heuristics.

(d)

The original implementation used Manhattan distance, but one could use Euclidean distance as well. This does not work as well because the space is discrete, which means that we cannot make diagonal steps, which the Euclidean measures assumes possible. This means that for points $p = (0, 0)$ and $q = (1, 1)$, the actual cost of reaching q from p is 2, as it is necessary to take a

step up, and a step to the side, while the Euclidean distance gives a cost of $\sqrt{2}$, which is a worse estimate than the Manhattan distance.

7 Hybrid A*

(a)

Hybrid A* is a variant of A* that is specifically designed for path planning in continuous spaces, like for autonomous vehicle navigation and robotics. A* is designed for discrete state spaces, and hybrid A* is designed for continuous state spaces. Hybrid A* incorporates vehicle dynamics into the planning process, so vehicle constraints are accounted for.

(b)

Hybrid A* is great for autonomous vehicle navigation, and robotics, since it operates in a continuous state space and it takes vehicle constraints into account.

(c)

The computation time was 40.635s, with a total of 197 iterations where the shortest path was 28.43. The path is quite smooth, but it does generate a path closer to the obstacle than necessary, causing needless sharp turns around the obstacle. It could have generated a straighter path, further from the obstacle, giving it a better angle for turning around the obstacle.