TSFS12 Hand-in 2: Planning for Vehicles with Differential Motion Constraints

JOHN DOE
E-mail: email@example.com
LiU-ID: johndoe

SAMUEL ÅKESSON
E-mail: email@example.com
LiU-ID: sermuns

2025-11-24

1. Introduction

The object of this hand-in exercise was to investigate two different planning methods.

2. Results

Exercise 4.2

The output from next_state provides all the possible states that can be acted upon from current state. The expected number of states were 8, considering the interval in which theta could adopt. However, the output showed 11 states.

Depth first provides a long path with a lot of backtracking. Whereas best first, breadth first, Dijkstra and Astar all provide a much shorter path, with best first being the least optimal of the four. Breadth first, Dijkstra and Astar provide the optimal path in terms of length, which was expected of Dijkstra and Astar but not as much for breadth first. Figure 1 shows the planned path for all five planners.

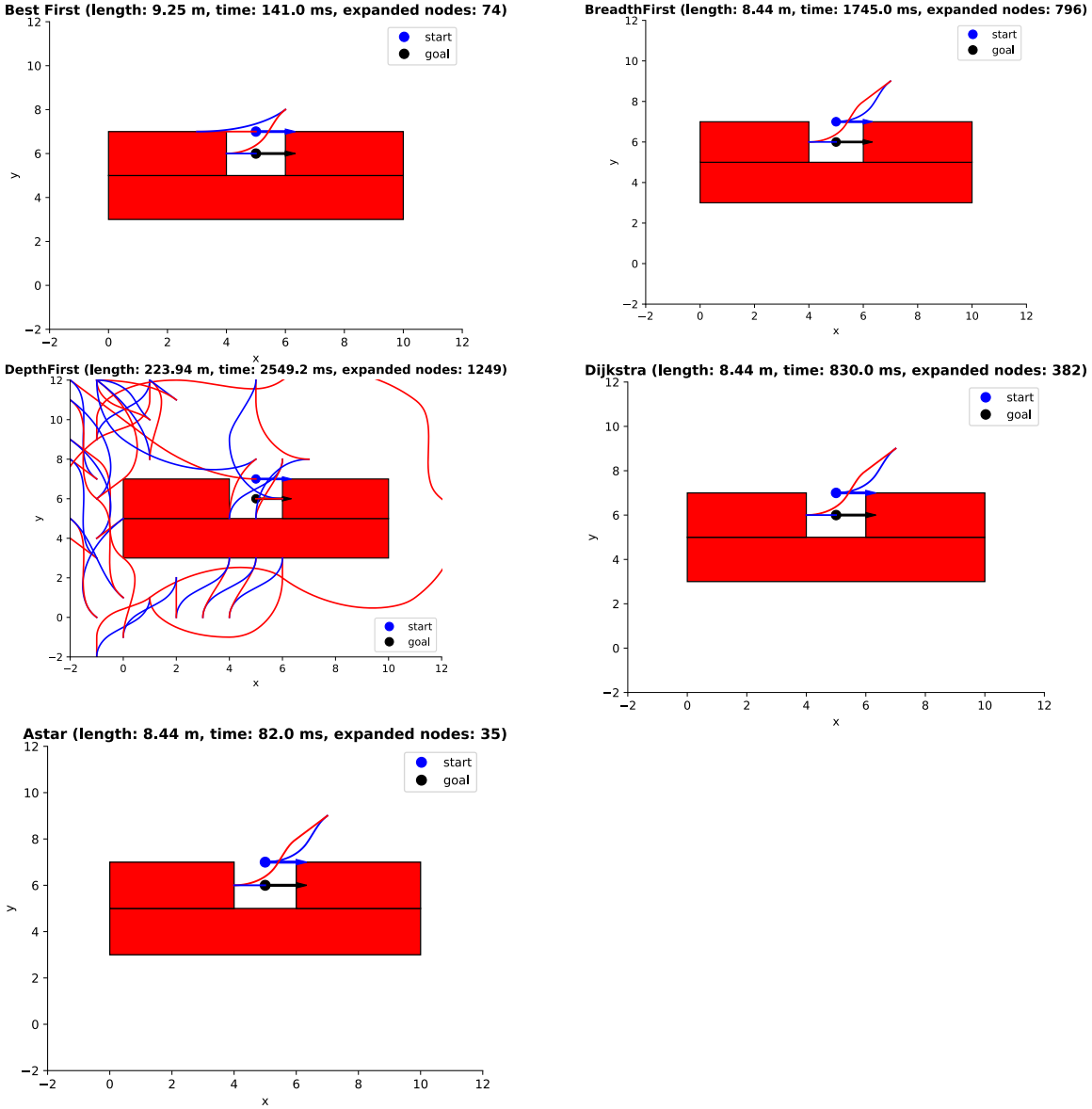

Figure 1: Motion plans generated by the search algorithms.

Exercise 4.3

Figure 2 shows the number of expanded nodes with respect to the planning time. The graph shows the values for four different mission lengths. Neither of the graphs seem to follow a linear development, where the planning time increase with the number of expanded nodes.

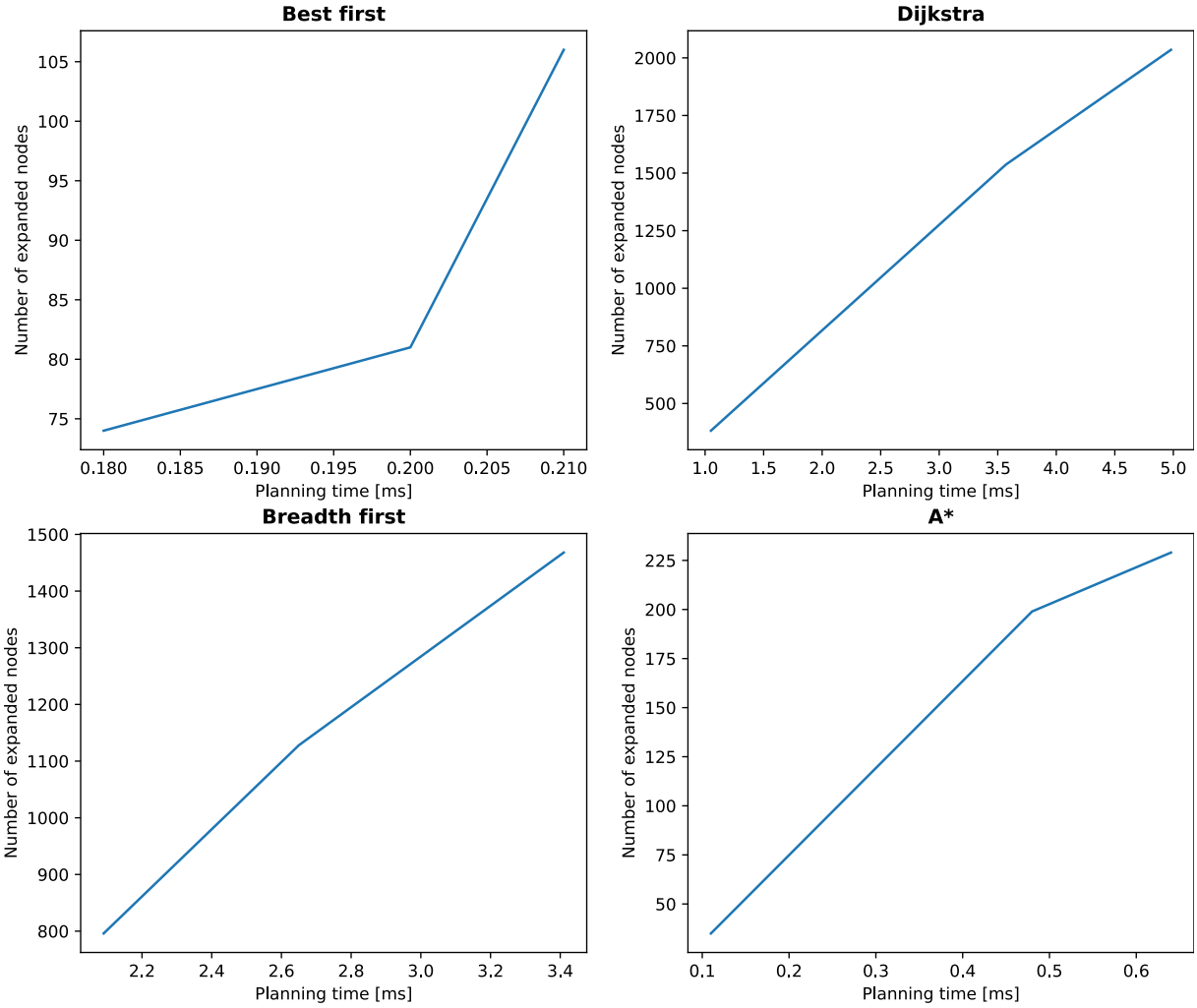

Figure 2: Expanded nodes vs. planning time for the graph-search algorithms.

Figure 3 shows the planning time with respect to the mission length.

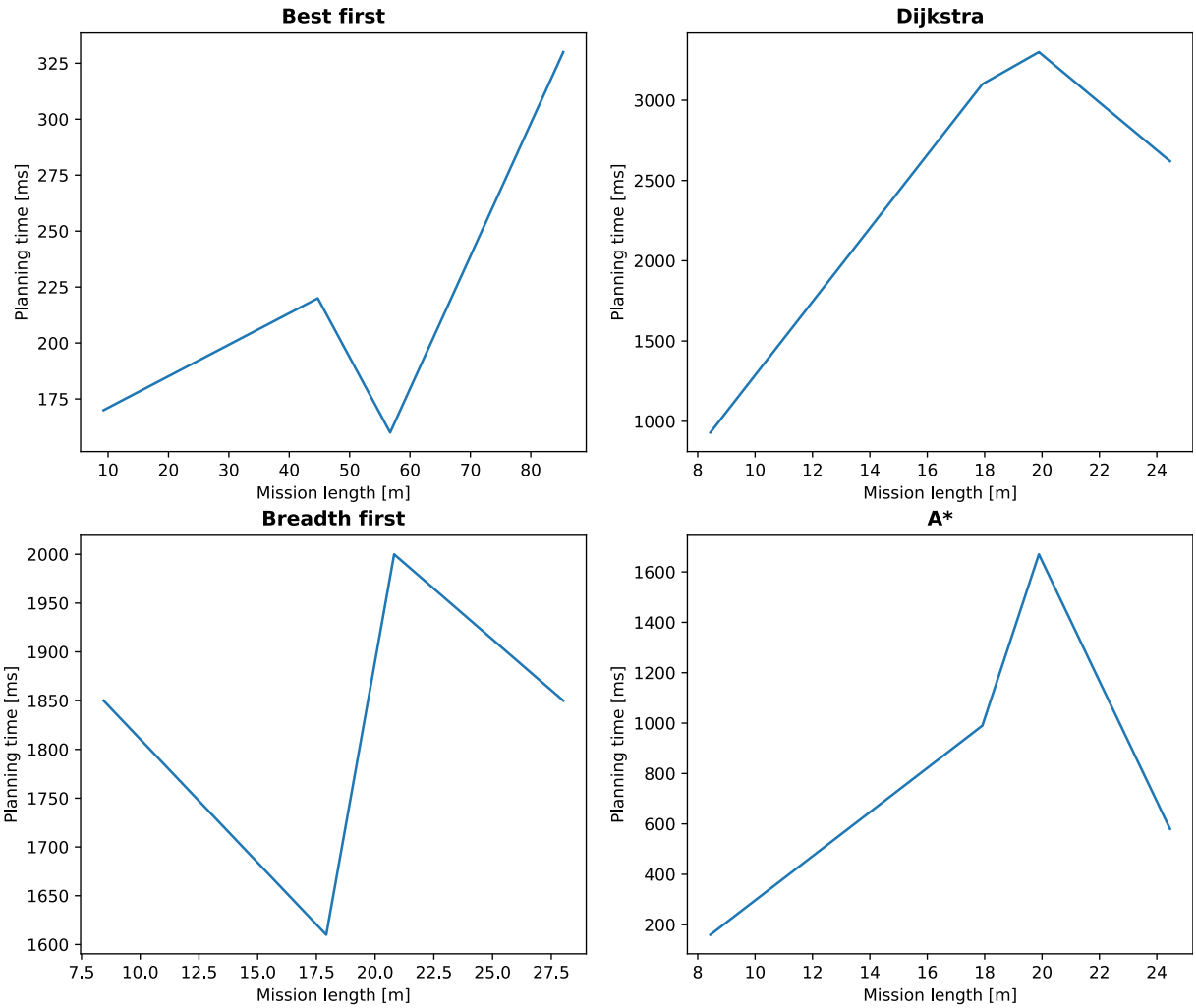

Figure 3: Mission length vs. planning time for the graph-search algorithms.

Alternating the options for the heuristic function showed that changing from euclidean distance to manhattan distance did not improve the planner, in terms of number of expanded nodes or path. However, when using an inflated euclidean distance the planning time decreased as well as number of expanded nodes.

Exercise 4.5

When running the planner with the same variables produced different solutions which means that the random tree is non-deterministic. Some runs were unlucky, where the planner explored different paths unnecessarily before finding the best path. Figure 4 shows the constructed RRT for the particle model.

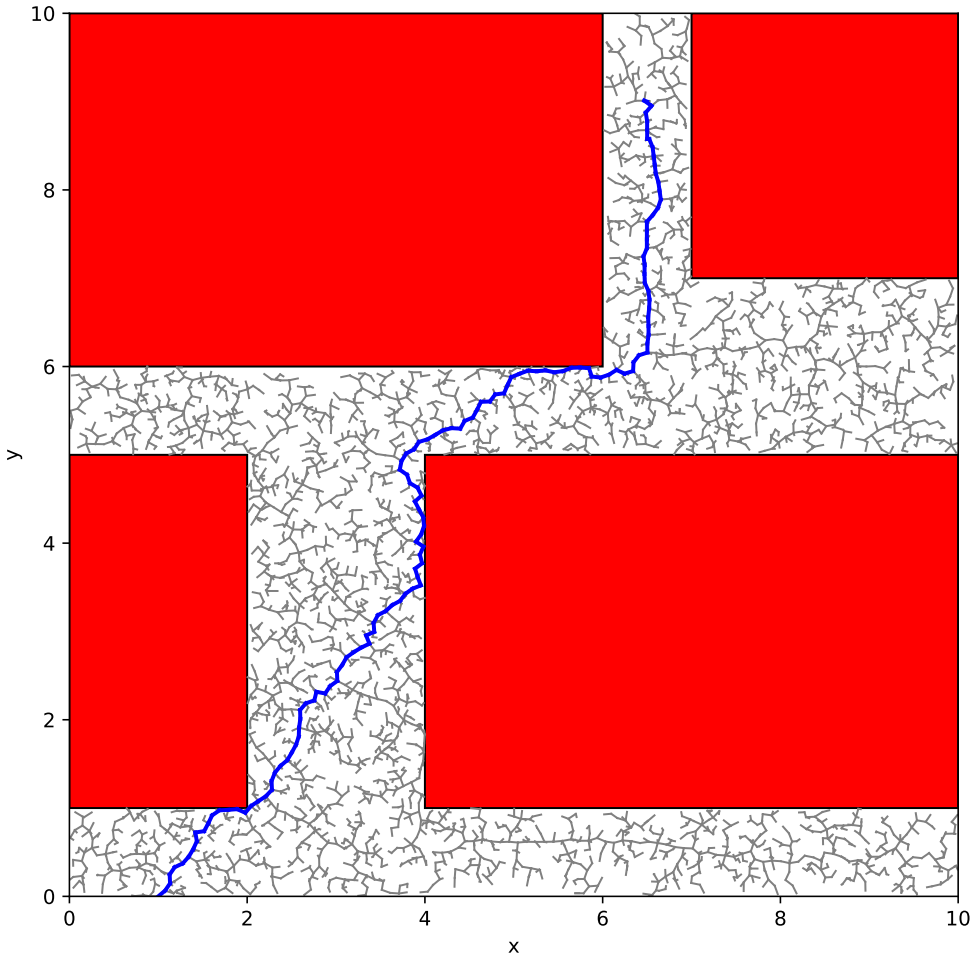

Figure 4: Constructed RRT for the particle model. Explored paths are gray and the solution is blue.

Excercise 4.6

By varying the variable opts.lambda it showed how a higher value resulted in less number of nodes visited in the solution. However it did not have much impact on the number of total nodes. In terms of distance, a higher value for lambda increased the solution distance.

Having a positive epsilon value instead of a negative one meant the solution became faster to compute and visited less nodes however, it might not reach its end goal.

When Beta was equal to 1 it could not traverse obstacles. Higher beta leads to stricter path toward goal less total number of nodes. For a lower beta value the resulted path is more jagged.

Exercise 4.8

Figure 5 shows the constructed tree with the blue line being the start to end path. The grey lines in the figure are the explored trajectories.

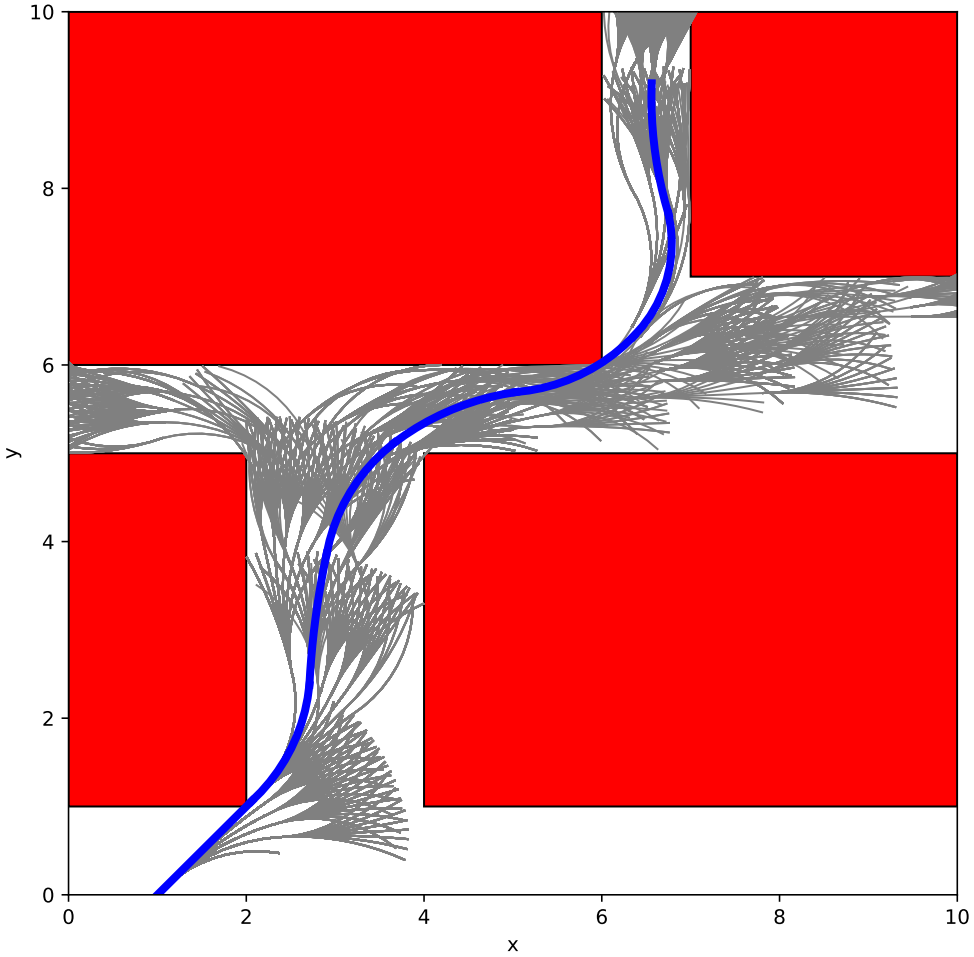

Figure 5: Constructed RRT for the kinematic model. Explored trajectories are shown in gray and the solution is blue.

Exercise 4.9

Some missions are impossible, such as having the car start against a wall and oriented into the wall. The car cannot turn fast enough to escape the wall, resulting in no valid states escaping from the starting state.

Increasing opts.lambda can result in the car not being able to traverse tight corridors, since it doesn't have the "reaction time" for it. Decreasing it, on the other hand, makes the car more reactionary, but makes the tree much more complex. The solution path also seems more curvy. See Figure 6 for an example where opts.lambda was decreased from 0.1 to 0.05.

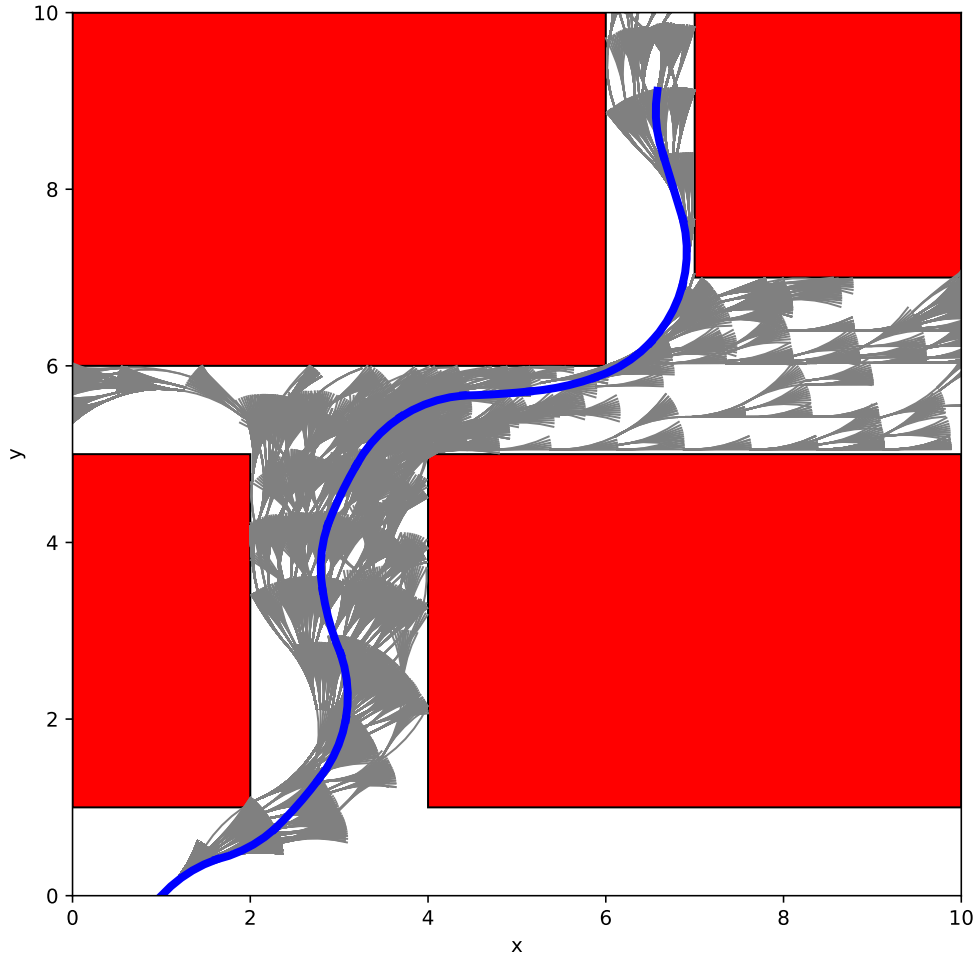

Figure 6: RRT constructed with opts.lambda decreased from 0.1 to 0.05.

Exercise 4.10

Currently, distance_fcn is performing $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (Listing 1), calculating the euclidean distance of $(\Delta x, \Delta y)$. It is not at all considering the difference in θ.

Listing 1: Original distance_fcn.

```
1  def distance_fcn(x1, x2):
2    d2 = x1 - x2
3    return np.sqrt(d2[0] ** 2 + d2[1] ** 2)
```

This means that the solution will (in most cases) not have the correct goal orientation.

Changing the distance_fcn naively to computing the euclidean distance of $(\Delta x, \Delta y, \Delta\theta)$ is probably a bad choice, since when the current state is far away from the goal state, the orientation is not important. A better idea is to include $\Delta\theta$ in the distance calculation, but prioritize it down- i.e. scale it down.

Listing 2: Proposed distance_fcn that takes $\Delta\theta$ into account.

```
1  def distance_fcn(x1, x2):
2    d2 = x1 - x2
3    return np.sqrt(d2[0] ** 2 + d2[1] ** 2) + 0.2 * d2[2]
```

Exercise 4.11

The original control set $\mathcal{U}$ is generated by

```
u_c = np.linspace(-np.pi / 4, np.pi / 4, 11)
```

giving us 11 distinct angles. Changing to a lower number of angles, makes solutions still possible, but noticeably more jagged, since it cannot make small adjustments.

In some missions however, it is required to decrease opts.lambda to compensate for the bad precision resulting from the few control with more often recurring decisions.

3. Discussion

Among the graph search motion planners Breadth First, Dijkstra and Astar provided the most optimal paths. Best first also provided a better path than depth first but less optimal than the other three. Using an inflated version of the euclidean distance as the heuristic function proved to reduce planning time and number of visited nodes. The RRT planner was non-deterministic, however tuning the parameters lambda, epsilon and beta had impact on the solution. Higher lambda reduced number of nodes but led to longer paths and a positive epsilon reduced planning time but might not finish the path.