Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab

Exercise 6
Dijkstra's Algorithm and the Dynamic Window Approach for Motion Planning

1 Introduction

The goal of this exercise is to implement motion planning algorithms that enable a robot to navigate through a partially known environment. In order to achieve this task, two commonly used algorithms will be applied: Dijkstra's algorithm for computing a navigation function based on a static map of the environment, and the Dynamic Window Approach (DWA) for online planning, taking into account both static and dynamic obstacles. The algorithms are described in the book "Introduction to Autonomous Mobile Robots" (Siegwart et al., 2011) in Chapter 6.

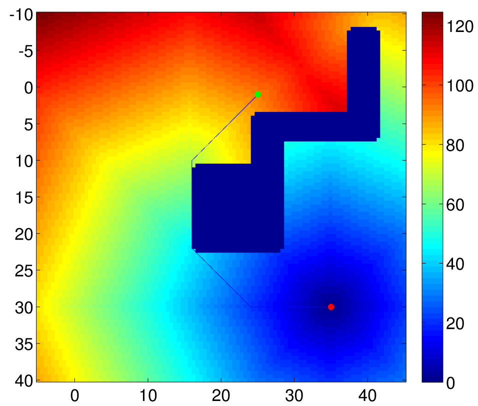

Figure 1: Dijkstra's algorithm for global motion planning in an 8-connected grid. The color of the map refers to the distance of the respective cell to the goal (blue: close, red: far). The blue line is the shortest path in the grid from the start (green dot) to the goal (red dot).

1.1 Dijkstra's Algorithm

Dijkstra's algorithm (Siegwart et al., 2011, p. 382) is a graph search algorithm that solves the shortest-path problem by applying the dynamic programming method. Starting from a source vertex in the graph, a discrete equidistant wavefront is expanded. Subsequently, all vertices are labeled with their lowest cost (distance) to the source vertex (see Fig. 1). The distance field created by Dijkstra's algorithm is free of local minima. Hence it can be used as a navigation function.

1.2 Dynamic Window Approach

The Dynamic Window Approach (Siegwart et al., 2011, p. 402), first proposed by Fox et al. (1997), is a robot navigation method that accounts for the kinodynamic constraints of the vehicle. Acceleration limits are incorporated by choosing motion commands (linear and angular velocity)

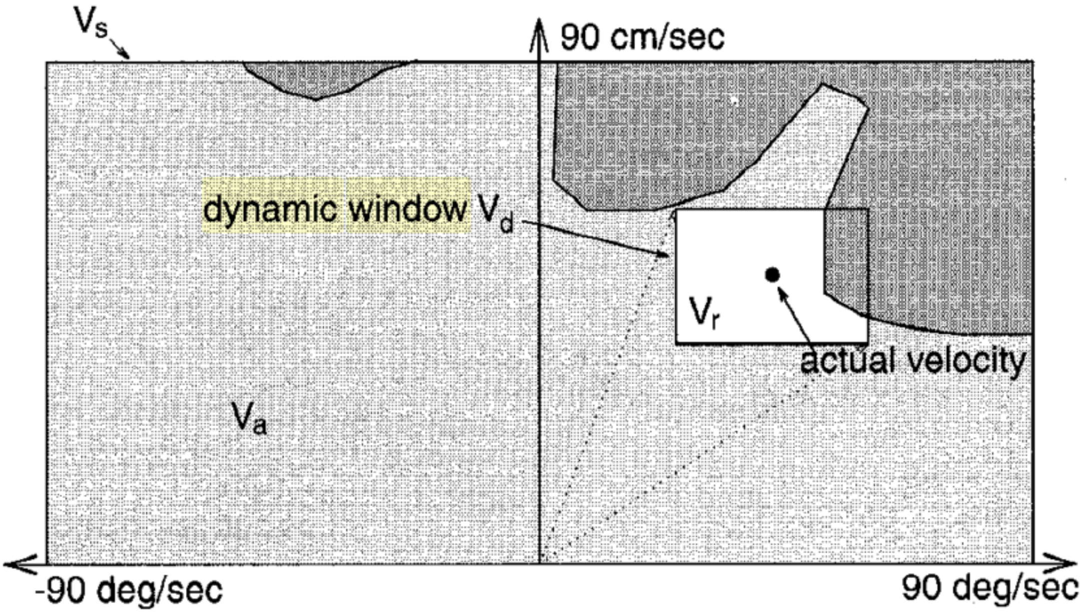

Figure 2: Dynamic Window Approach

from a finite window in the velocity space, centered at the current state. Velocities that are inevitably in collision are pruned, leaving a set of feasible velocities which form the search space. Velocities within the dynamic window are then sampled and rated based on a custom score function. In the original paper by Fox et al., the proposed score function

$$G\left(v,\omega\right) = \alpha\,\text{heading}\left(v,\omega\right) + \beta\,\text{dist}\left(v,\omega\right) + \gamma\,\text{velocity}\left(v,\omega\right) \tag{1}$$

computes a weighted sum of a heading-term (enforcing alignment of the robot towards the goal position), a distance term (penalizing samples resulting in motions towards obstacles) and a velocity term (rewarding higher velocities in order to make progress towards the goal). The optimal control input for the robot is given by the velocity sample $[v, \omega]$ that maximizes the score function G.

Local Dynamic Window Approach. The original Dynamic Window Approach is a *local* planning method that does not take into account the topology of the environment. It is vulnerable to cul-de-sac environments, as progress towards the goal is enforced only via the definition of the *heading* term

$$\text{heading}\left(v,\omega\right) = |\pi - |\Delta\theta||, \tag{2}$$

where $\Delta\theta$ represents the heading offset to the goal (i.e., the angle between the robot's heading and the straight line to the goal).

Global Dynamic Window Approach. Brock & Khatib (1999) addressed the shortcomings of the Dynamic Window Approach by including a global navigation function. The *heading* term is modified to capture the alignment of the robot with the gradient of the navigation function. This enables to guide the robot to the goal even in the presence of dead ends in the environment.

2 Matlab Implementation

The above mentioned algorithms will be utilized in this exercise to navigate a robot. This section explains their implementation.

2.1 Introduction

In order to complete the implementation, you will have to fill out blanks (indicated by the `TODO` symbol) in the provided Matlab code. There are two files at the top level folder of exercise 6 that require modification:

- **`dynamicWindowApproach.m`**
  - Implementation for the Local and Global Dynamic Window Approach
- **`dijkstra.m`**
  - Implementation for Dijkstra's algorithm

Note that all file paths in this document are given relative to the exercise's root folder. The comments in the code will provide you more information about what has to be implemented. Each of them can be tested with the following scripts:

- **`test/testLocalDwa.m`**
- **`test/testGlobalDwa.m`**
- **`test/testDijkstra.m`**

The scripts

- **`test/testNavigationLocalDwa.m`** and
- **`test/testNavigationGlobalDwa.m`**

will allow you to test the navigation performance in a very simple simulation environment, before moving on to navigating a simulated robot in CoppeliaSim with the script **`vrep/vrepSimulation.m`**.

2.2 Exercise 6.1: Local Dynamic Window Approach

The first task of the exercise is to implement the Local Dynamic Window Approach. All necessary modifications will take place in the file **`dynamicWindowApproach.m`**. The function

```
[ vSolution, omegaSolution, debug ] = dynamicWindowApproach( robotState,
goalPosition, localMap, parameters, globalGradientMap)
```

takes the current robot state `robotState`, the goal position `goalPosition`, a local map `localMap` containing obstacle information from a local surround sensor and a parameter struct `parameters` providing amongst others a description of the robotic platform. Optionally, the gradient of the global navigation function `globalGradientMap` can be provided (which is not relevant for the implementation of the local dynamic window approach).
The outputs are given by the optimal translational velocity `vSolution` in m/s and rotational velocity `omegaSolution` in rad/s. The struct `debug` will contain information that allows us to

check for the correctness of the algorithm's internals.

Task: Your task is to fill out the blanks indicated by `TODO (Ex.6.1)`. After finishing this task, you can check the correctness of your implementation with the script **`test/testLocalDwa.m`**. This script will load precomputed solution values for predefined inputs from the file **`test/localDwaSolution.mat`** and will compare your output values against them.

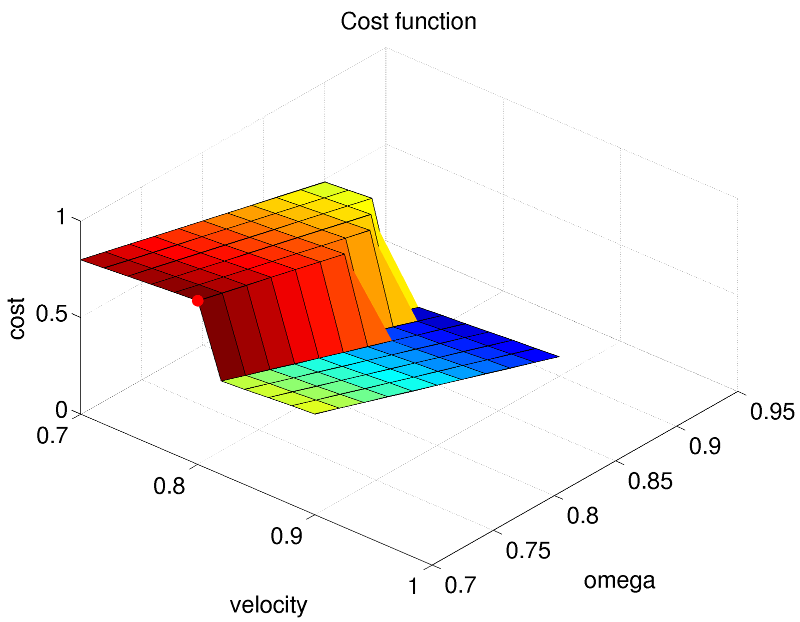

Figure 3: Score function evaluated by the Dynamic Window Approach.

For easier debugging, you might want to enable plotting of internal data via the field `plot` in the `parameters` struct. This will display the generated trajectory set as well as a plot of the score function (see figure 3).

Validation: Once your implementation is correct, you can run the simple navigation simulation with the script **`test/testNavigationLocalDwa.m`**. Play with the weighting factors of the DWA score function and the start and goal locations, and observe the changes in the behavior of the planner.

2.3 Exercise 6.2: Dijkstra's Algorithm

You will soon notice that the Local Dynamic Window Approach runs into problems in cul-de-sack environments. To overcome this problem, we will now implement Dijkstra's Algorithm to compute a minima-free navigation function later to be used in the Global Dynamic Window Approach. All necessary modifications will take place in the file **dijkstra.m**. The corresponding function `dijkstra`

```
function [ costs, costGradientDirection, path ] = dijkstra( map, goalIdx,
parameters, startIdx )
```

takes a global description of the environment `map`, a 2D index of the goal vertex `goalIdx` and a parameter struct `parameters`. The start vertex `startIdx` (in most situations the current robot location in the map) is only required if an explicit path from start to goal is requested via the

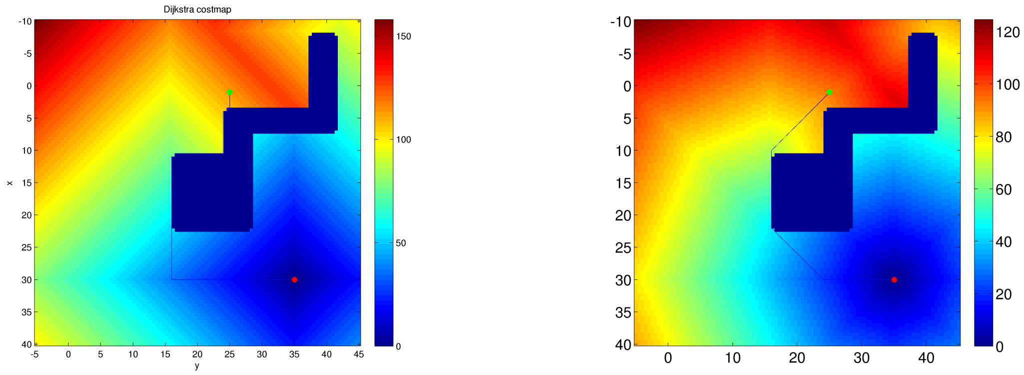

Figure 4: Resulting distance maps for the 4-connected (left) and 8-connected (right) Dijkstra's algorithm.

output argument `path`. If no path is requested, the algorithm will return the cost map `costs`, encoding the geodesic distances for every cell to the goal vertex and the gradient direction `costGradientDirection`.

Task: Your task is again to fill out the blanks indicated by the `TODO` symbol.

Validation: You can check the correctness of your implementation with the script **`test/testDijkstra.m`**. The output should look like displayed in figure 4.

2.4 Exercise 6.3: Global Dynamic Window Approach

In this exercise, we will use the result of the global planner developed in Exercise 6.2 to enhance the Dynamic Window Approach with global guidance. This will finally yield a motion planning framework that enables autonomous navigation even in the presence of difficult obstacle configurations (e.g. dead ends), while still taking into account the robot's kinematic and dynamic constraints. This is achieved by adapting the *heading* term in the score function of the DWA: instead of looking at the orientation with respect to the straight line to the goal, we now wish to maximize the alignment of the robot with the *direction of the gradient* of the geodesic distance to the goal (the result of Exercise 6.2).

Task: To do so, add the missing code indicated by `TODO (Ex.6.3)` in the file **`dynamicWindowApproach.m`**.

Validation: Test your implementation with the script **`test/testGlobalDwa.m`**. Similar to Exercise 6.1, you can run a simple navigation simulation to observe the behavior of the planner: **`test/testNavigationGlobalDwa.m`**

2.5 Exercise 6.4: Simulated Navigation in CoppeliaSim

Now that everything is put together and the correctness of your implementation is validated, let's use our planners to move a simulated robot in CoppeliaSim. To do so, start CoppeliaSim, load the scene file **`scene/mooc_exercises.ttt`**, and press **run**.

After that, run the Matlab file **`vrep/vrepSimulation.m`**. The script will first get a global map from the simulator and run Dijkstra's algorithm to compute the global navigation function. On

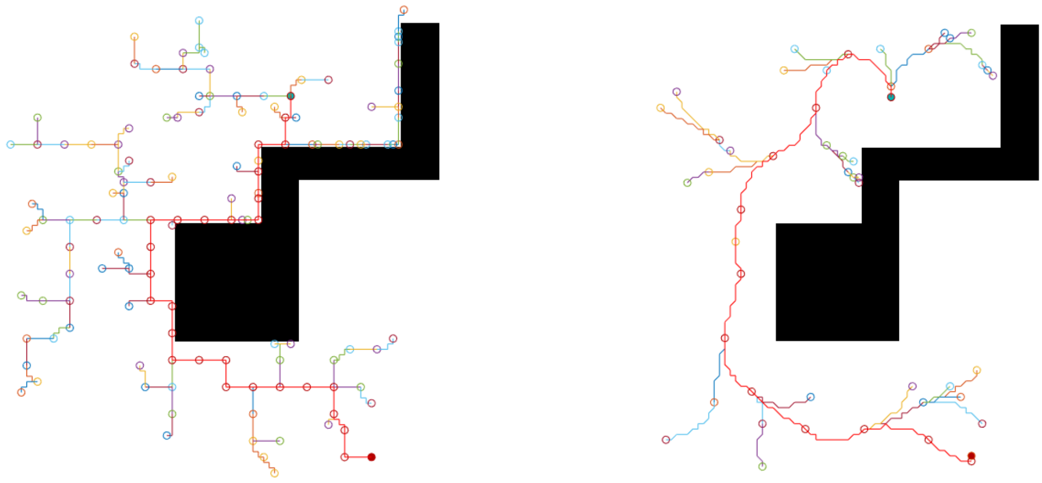

Figure 5: Example solution of pixel-wise (left) and kinetically constrained (right) RRT algorithm.

success, it will subsequently call the Dynamic Window Approach to retrieve suitable motion commands for the robot, that will lead the robot to the goal location.

Depending on the computational ressources of your computer, the implemented algorithms might run too slowly in order to have a decent real-time control. In this case, consider using the stepped simulation by setting the parameter `parameters.vrepSteppedSimulation = true`. This will trigger a simulation step after each call to the Dynamic Window Approach algorithm.

2.6 Optional - Exercise 6.5: Rapidly exploring Random Trees

This section is not relevant to the exam. It is added as an extra to introduce the curious student to rapidly exploring random trees, a sampling-based algorithm for path-finding.

Rapidly-exploring Random Trees, proposed by LaValle (1998) is an algorithm for searching n-dimensional space efficiently, by building a tree graph which progressively fills the space in a random manner.

The files **`rrt.m`** **`dynamic_rrt.m`** and **`test/showcaseRRT.m`** have been added to the exercise code folder. Once the placeholders in **`rrt.m`** have been filled in, run **`test/showcaseRRT.m`** to see an example solution of RRT in the same test scenario as Exercise 6.2. Note that in the case of RRT, there is no single correct solution. The algorithm may yield different solutions as it is not deterministic in nature, and the solution is not guaranteed to be the shortest possible path.

In addition, many flavors of RRT exist and are commonly used, taking into account several constraints (for example the kinetic constraints of the robot), or modifying details of the algorithm depending on scenario-based priorities (memory/complexity limitations, recomputing paths to remove zig-zags - named RRT*, and so on).

The solution to the exercise will also include a version of RRT constrained to robot kinetics, implemented in **`dynamic_rrt.m`**.

References

Brock, O., & Khatib, O. 1999. High-speed Navigation Using the Global Dynamic Window Approach. *Pages 341–346 of: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Fox, D., Burgard, W., & Thrun, S. 1997. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics & Automation Magazine*, **4**(1), 23–33.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning.

Siegwart, R., Nourbakhsh, I., & Scaramuzza, D. 2011. *Introduction to Autonomous Mobile Robots*. 2nd edn. MIT Press.