Exercise 2

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

Due date: March 9, 2018
Submit by Brightspace before midnight.

Course: SC42090 Robot Motion Planning and Control, TU Delft, DCSC
Contact: Linda van der Spaa, email: L.F.vanderspaa@tudelft.nl
Office hours: Wiener hall C-3-190, Thursdays 10:00 – 11:00

Acknowledgements

This exercise is derived from the identically named exercise by the Autonomous Systems Lab of ETH Zürich.

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 books "Planning Algorithms" [LaValle, 2006, Ch. 2] and "Introduction to Autonomous Mobile Robots" [Siegwart and Nourbakhsh, 2004, Ch. 6].

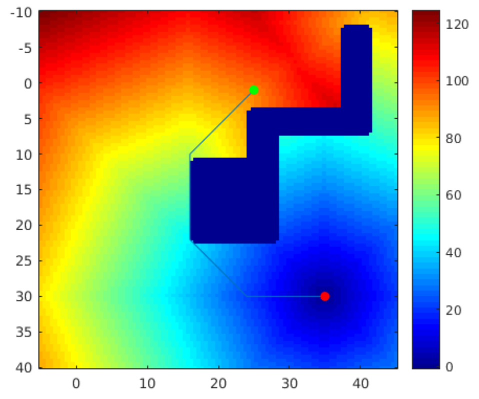


Figure 1: Dijkstra's algorithm for global motion planning in an 8-connected grid. The color of the map refers to the distance of the respective cell to the goal (blue: close, red: far). The blue line is the shortest path in the grid from the start (green dot) to the goal (red dot).

The exercise consists of tasks that you have to do, but do not require a written answer, and questions that do require a written answer. For each task, a method of validation is provided, so you will be able to correct you solutions to the tasks yourself. The questions to be answered are indicated by the number of points a correct answer will yield between parentheses. A maximum number of 20 points can be earned by completing this exercise, 10 points extra for the bonus part. **To complete this exercise, hand in your answers to the questions using at most 2 pages (A4, 11pt). For the bonus exercise you are allowed one page extra.**

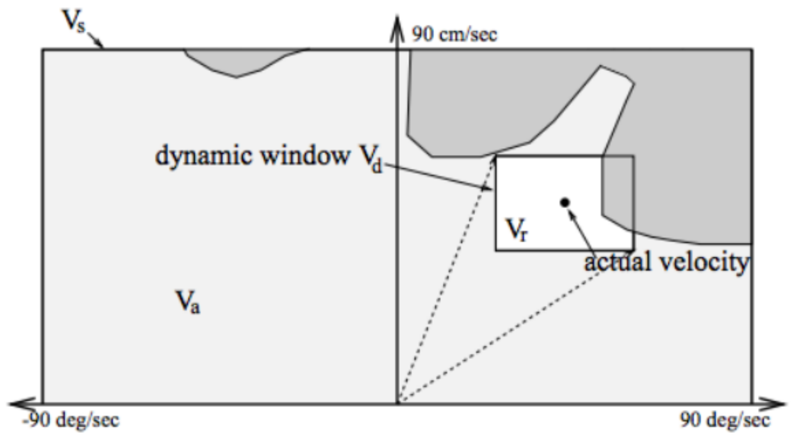


Figure 2: Dynamic Window Approach [Fox et al., 1997, Siegwart and Nourbakhsh, 2004, Fig. 6.14]

1.1 Dijkstra's Algorithm

Dijkstra's algorithm [LaValle, 2006, Section 2.2.2, p.36] 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 and Nourbakhsh, 2004, Section 6.2.2.5, p. 282], 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) 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

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

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

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

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.

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 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 V-REP with the script `vrep/vrepSimulation.m`.

Exercise 2.1: 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. 2.1)`:

a) Define the dynamic window in lines `270` to `273`.

b) Define the heading function in line `359` and `373`.

c) Define the cost function in line `192`.

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.

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.

Validation: Once your implementation is correct, you can run the simple navigation simulation with the script `test/testNavigationLocalDwa.m`.

(4pt) Vary the weighting factors of the DWA score function and the start and goal locations, and explain the changes in the behavior of the planner.

Exercise 2.2: 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 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:

a) In the main `dijkstra` function, initialize goal cost in line `37` and define a suitable loop termination criterion in line `50`.

b) Expand the nodes in lines `154` to `157` and `162` to `165`.

c) Define the cost for traveling from node to node in line `176`.

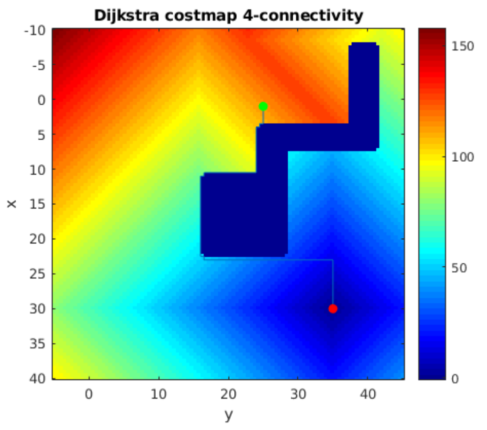
d) Compute the cost gradient in line `200`.

e) To extract the optimal path with `extractBestCostPath`, define a suitable loop termination criterion in line `217`, find the minimum cost and the corresponding index in lines `229` and `230` and return the path in line `243`.

Validation: You can check the correctness of your implementation with the script `test/testDijkstra.m`. The output should look like displayed in Figure 3.

Exercise 2.3: Global Dynamic Window Approach

In this exercise, we will use the result of the global planner developed in Exercise 2.2 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 2.2).



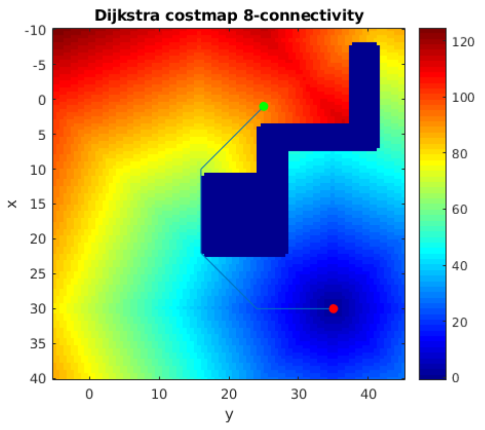


Figure 3: Resulting distance maps for the 4-connected (left) and 8-connected (right) Dijkstra's algorithm.

Task: To do so, add the missing code indicated by `TODO (Ex. 2.3)` in the file `dynamicWindowApproach.m`.
Validation: Test your implementation with the script `test/testGlobalDwa.m`. Similar to Exercise 2.1, you can run a simple navigation simulation to observe the behavior of the planner: `test/testNavigationGlobalDwa.m`.

(4pt) Again, vary the weighting factors of the DWA score function and the start and goal locations, and explain the changes in the behavior of the planner.

(4pt) Use your insights to improve the escaping behavior from initial robot position $(-0.5, 3.5)$ to the default goal position. Provide the weighting factors you found and show in what way the behavior improved.

Exercise 2.4: Simulated Navigation in V-REP

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 V-REP. To do so, start V-REP, 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 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 resources 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.

(4pt) Observe that in `vrep/vrepSimulation.m` the default weighting factors of the DWA score function have different values compared to the previous exercises. Again, play with the weighting factors of the DWA score function and the start and goal locations and also try the weighting factors used by default in the previous exercises (`parameters.headingScoring` = 0.3, `parameters.velocityScoring` = 0.45, `parameters.obstacleDistanceScoring` = 0.25) Explain why the scene in V-REP requires such different weight factors.

(4pt) You will have observed that the success of the planner may rely very closely on the starting position of the robot, especially when starting from within the dead end. Yet Dijkstra's algorithm provides a gradient to follow without local minima. Explain why it is still so easy to get stuck.

Bonus Exercise 2.5

(4pt) Explain how the planner could be made more reliable for any set of start and goal positions, and less sensitive to the choice of weighting factors of the DWA score function.

(6pt) Implement your suggested changes, test your new method and discuss your results.

To complete this exercise, hand in your answers to the questions using at most 2 pages (A4, 11pt). For the bonus exercise you are allowed one page extra.

References

[Brock and Khatib, 1999] Brock, O. and Khatib, O. (1999). High-speed navigation using the global dynamic window approach. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 341–346. IEEE.

[Fox et al., 1997] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33.

[LaValle, 2006] LaValle, S. M. (2006). *Planning algorithms.* Cambridge university press.

[Siegwart and Nourbakhsh, 2004] Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots.* Bradford Book.