Karlsruhe Institute of Technology

Institute for Anthropomatics and Robotics (IAR)

High Performance Humanoid Technologies (H^2T)

Prof. Dr.-Ing. T. Asfour

Roboterpraktikum
Summer Term 2022

Exercise 7:

Path Planning for Navigation Tasks

Advisor:

Rainer Kartmann

Summary: In this exercise, several path planning algorithms have to be implemented for a 2D navigation scenario. The discussed algorithms cover the potential field approach, the cell decomposition algorithms Dijkstra's and A*, and randomized approaches like the Rapidly-exploring Random Trees (RRT). The exercises have to be implemented in C++, where the Simox robotic toolbox library is used for convenient access to robot models and collision detection.

Contents

1 Introduction

Path planning is an essential capability for autonomous robots like service robots, autonomous cars or humanoids. There exists a variety of algorithms covering different problem classes like 2D navigation, planning with non-holonomous vehicles, 3D/6D planning for complex robots, or planning while considering dynamic constraints. In this exercise, we will focus on a simple planning problem: 2D navigation planning for holonomic robots without considering dynamics. This means that the orientation of the robot can be ignored (the robot is approximated by a circle) and it is additionally assumed that the robot is able to execute any motions without limitations due to acceleration constraints.

Concerning the aforementioned assumptions, we will discuss several algorithms in this document which can be used to plan paths for autonomous navigation tasks from a start to an end position. The following standard algorithms will be presented: the potential field approach, Dijkstra's algorithm, A*, and Rapidly-exploring Random Trees (RRT).

2 Fundamentals

In this section, we will discuss fundamental concepts such as the configuration space and several path planning algorithms will be introduced.

2.1 Definitions

Configuration space $\boldsymbol{C}$ The configuration space of a robot is the space of all possible configurations that can be realized by the system. Depending on the application, this includes its *motion degrees of freedom* and/or its *joint positions*. For path planning, a configuration is usually (and mainly in this exercise) a 2D vector consisting of the robot's translational motion degrees of freedom in the plane (i.e., its (x, y) position) but can also include its orientation in the plane (resulting in a 3D vector) or its 3D position and orientation (resulting in 6D vectors). The computational costs of the planning problem drastically increase with the number of dimensions covered by the configuration space. The configuration space is the disjoint union of the free and obstacle spaces $\boldsymbol{C} = \boldsymbol{C}_{\text{free}} \cup \boldsymbol{C}_{\text{obst}}$.

Free space $\boldsymbol{C}_{\text{free}}$ The free space denotes the subset of the configuration space where the robot does not collide with objects in its environment or, in case of joint positions, with itself.

Obstacle space $\boldsymbol{C}_{\text{obst}}$ The obstacle space is the opposite of the free space and therefore denotes the set of configurations that are in collision.

2.2 Path Planning

Path planning refers to the search for a collision-free trajectory of a rigid object, such as a mobile robot like the ARMAR-III or an autonomous vehicle. In this exercise, we consider only the movement of a mobile robot in the two-dimensional Cartesian space. The concepts can, however, be transferred to three dimensions, for instance, for the path planning of satellites [7] considering all six possible degrees of freedom.

In the two-dimensional case, the *work space* of a robot typically is identical with the configuration space when considering only translational degrees of freedom, that is, the position of the robot (see [15]). If the orientation of the object is considered as an additional rotational degree of freedom, a configuration space with three dimension is spanned [13]. This is known as the *piano movers' problem* [17], where the objective is to determine a trajectory from an *initial configuration* to a *target configuration* on which a rigid object can move without collisions with the environment.

In the following paragraphs, a selection of typical algorithms for solving the path planning problem will be presented.

Shortest Distance Search

The problem of path planning can be transformed into the search for the shortest distance on an undirected graph. To this end, the configuration space and free space, respectively, have to be transformed into a graph structure. Several approaches for this purpose have been proposed in the literature including *Voronoi diagrams*, *visibility graphs* [14] and *cell decomposition*. The latter will be described in the following paragraph. Once a graph has been constructed, graph algorithms such as Dijkstra's and A* can be applied to find the shortest path. Such approaches for path planning are *global* in that they do not run the risk of running into local minima, at the price of higher complexity.

Cell Decomposition Cell decomposition (for instance, using *quad trees* as in [16]) hierarchically subdivides the configuration space into smaller cells. For each of the resulting cells, it can be determined whether the cell is a subset of the free space $\boldsymbol{C}_{\text{free}}$, the obstacle space $\boldsymbol{C}_{\text{obst}}$ or contains configurations of both. In the latter case, the cell can be either further subdivided or considered being part of the obstacle space. This way, the configuration space is discretized by the hierarchical representation. A collision-free path must traverse only cells that lie completely in $\boldsymbol{C}_{\text{free}}$. Fig. 1 shows an exemplary cell decomposition created with the *quad tree* algorithm (meaning that each cell is divided into four equally-sized non-overlapping cells, forming a tree where each node has four children). Alternatively, a *kd-tree* decomposition [3] can be applied.

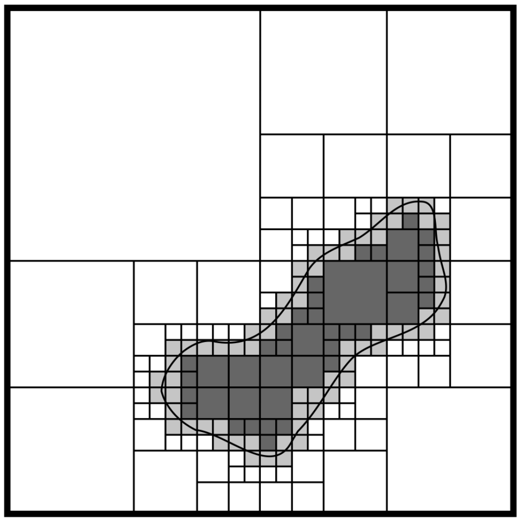

Figure 1: Cell decomposition hierarchically subdivides the (in this case, two-dimensional) configuration space into cells of decreasing size. White cells and dark gray cells completely lie in the free space $\boldsymbol{C}_{\text{free}}$ and obstacle space $\boldsymbol{C}_{\text{obst}}$, respectively. The remaining cells are further subdivided until a desired granularity has been reached.

Dijkstra's and A* algorithm Dijkstra's Algorithm [5] computes the shortest distance in an undirected graph. The distances can, for instance, be the distances between the centers of the cells of the previously described cell decomposition. Note that therefore, neighborhood relations between the cells must be known (and be retrievable from a tree). For each node of the graph, the algorithm stores the (minimal) distance to the starting node and its predecessor on the shortest path from the starting node and whether it has been visited by the algorithm.

At the beginning, the starting node is initialized with distance zero and marked as visited, and a list containing all nodes is created. In each following step, the node i with (currently) smallest distance d_i that has already been visited is chosen and removed from the list. For each neighbor j of i, its distance d_j is updated according to

$$d_j = \begin{cases} d_i + d_{i,j}, & d_j \text{ has not been visited before} \\ d_i + d_{i,j}, & d_j > d_i + d_{i,j} \\ d_j, & \text{else} \end{cases} \quad , \tag{1}$$

where $d_{i,j}$ denotes the distance between the two nodes. If d_j is updated, node i is stored as the predecessor of node j. The algorithm proceeds with choosing the node that has already been visited and has the smallest distance to the start node.

The A* algorithm [6] is an extension to Dijkstra's algorithm. It requires the availability of an *optimistic heuristics*, that is, an estimation for the distance to the target that is always lower or equal than the actual distance. In case of path planning, the Euclidean distance between the point in the free space and the target can be used (as there is no shorter way than the direct line connecting both points). That way, the shortest path can be detected faster in most cases.

Potential Fields

The *potential field method* presented by Khatib [8, 9] is an approach for *locally* solving the path planning problem. Inspired by the electrical *Coulomb field* known in physics, the direction of motion of a rigid object is determined by calculating the potential of its position in the configuration space (i.e., its configuration). In order to avoid collisions, obstacles create a repelling potential while the target position attracts the object. For the resulting potential field, the *gradient*, that is, the resulting direction of motion, can be computed to find a collision-free route. As opposed to global methods, the gradient is computed locally for the current position of the object with the effect the agent can be trapped in a local minimum and consequently fail to reach its target configurations. Therefore, these methods are suited best for local planning tasks and robot systems in low-dimensional configuration spaces.

Assumption: The robot translates freely in $\mathbb{R}^2$ with a fixed orientation. The potential function U is used to *attract* the robot to the goal and to *repel* it from the obstacles.

- Attractive potential $U_a(\boldsymbol{q})$ associated with $\boldsymbol{q}_{\text{goal}}$.
- Repulsive potential $U_r(\boldsymbol{q})$ associated with $\boldsymbol{C}_{\text{obst}}$.
- The resulting potential $U(\boldsymbol{q}) = U_a(\boldsymbol{q}) + U_r(\boldsymbol{q})$ must be differentiable for every $\boldsymbol{q} \in \boldsymbol{C}_{\text{free}}$.

The field of artificial forces: $F(\boldsymbol{q}) = -\nabla U(\boldsymbol{q})$. $\nabla U(\boldsymbol{q})$ denotes gradient of U at $\boldsymbol{q}$, i.e., $\nabla U(\boldsymbol{q})$ is a vector that "points" in the direction of "steepest increase" of U at configuration $\boldsymbol{q}$. E.g., if $C = \mathbb{R}$, then

- $\boldsymbol{q} = (x, y)$,
- $\nabla U(\boldsymbol{q}) = \left[\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right]^\top$ points into direction of highest rate of change, and
- $|\nabla U(\boldsymbol{q})| = \sqrt{\frac{\partial U}{\partial x}^2 + \frac{\partial U}{\partial y}^2}$ is the magnitude of the rate of change.

The attractive potential:

$$\begin{aligned}
U_a(\boldsymbol{q}) &= \frac{1}{2} c_a \cdot \|\boldsymbol{q} - \boldsymbol{q}_{\text{goal}}\|^2 \\
\nabla U_a(\boldsymbol{q}) &= c_a \cdot (\boldsymbol{q} - \boldsymbol{q}_{\text{goal}}) \\
F_a(\boldsymbol{q}) &= -\nabla U_a(\boldsymbol{q}) = -c_a \cdot (\boldsymbol{q} - \boldsymbol{q}_{\text{goal}})
\end{aligned}$$

The attractive force $F_a(\boldsymbol{q})$ is a vector directed toward $\boldsymbol{q}_{\text{goal}}$ with magnitude linearly related to the distance from $\boldsymbol{q}$ to $\boldsymbol{q}_{\text{goal}}$. The factor $c_a \in \mathbb{R}$ can be used to adjust the attraction.

The repulsive potential: We assume that a method is present for computing the point $\boldsymbol{q}_i$ on the surface of each obstacle i with minimum distance $\|\boldsymbol{q} - \boldsymbol{q}_i\|$ to the robot.

$$U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{1}{2} \cdot c_r \cdot \left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_i\|} - \frac{1}{\hat{d}} \right)^2, & \|\boldsymbol{q} - \boldsymbol{q}_i\| \leq \hat{d}, \\ 0, & \text{else} \end{cases}$$

$$\nabla U_{r,i}(\boldsymbol{q}) = \begin{cases} c_r \cdot \left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_i\|} - \frac{1}{\hat{d}} \right) \cdot \frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_i\|^2} \cdot \frac{\boldsymbol{q}-\boldsymbol{q}_i}{\|\boldsymbol{q}-\boldsymbol{q}_i\|}, & \|\boldsymbol{q} - \boldsymbol{q}_i\| < \hat{d} \\ 0, & \text{else} \end{cases}$$

The factor c_r is used to adjust the effect of the repulsive potential. The value $\hat{d}$ can be used to adjust the distance of influence.

The resulting potential and artificial force:

$$U(\boldsymbol{q}) = U_a(\boldsymbol{q}) + \sum_i U_{r,i}(\boldsymbol{q})$$
$$F(\boldsymbol{q}) = -\nabla U(\boldsymbol{q})$$

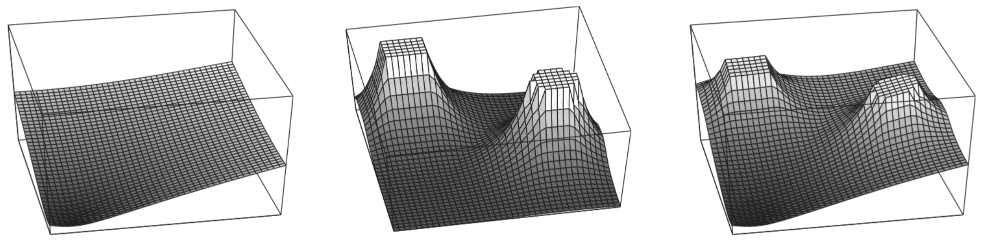

Figure 2: In the potential field method, the target region and the obstacles define repelling and attracting potentials which are used for navigation.

Local Minima

Since the potential field algorithm only considers local solutions, there might occur situations where no further movements are possible, i.e. the gradient is zero, but the goal has not been reached. Such local minima can be addressed with the random motion approach introduced by Barraquand and Latombe [1, 2]:

- Start at $\boldsymbol{q}_{\text{init}}$ and use best-first search to follow the gradient of $U(\boldsymbol{q})$ until you reach a local minimum $\boldsymbol{q}_{\text{loc}}$ (called gradient motion).
- Execute a random walk starting at $\boldsymbol{q}_{\text{loc}}$ and ending at some new configuration $\boldsymbol{q}_{\text{new}}$ (random walk approximates a Brownian Motion).
- Execute another gradient motion starting from $\boldsymbol{q}_{\text{new}}$ (end of random walk).
- Repeat until $\boldsymbol{q}_{\text{goal}}$ is reached.

Note, that the random walk should be performed in small steps while each intermediate configuration has to be checked for collisions.

2.3 Rapidly-Exploring Random Trees (RRT)

The *Rapidly-exploring Random Trees (RRT)* belong to the class of *sampling-based* planning algorithms. In most cases, an explicit modeling of the obstacle space is not tractable and therefore, the free space is approximated by a tree structure. RRTs have been introduced by LaValle and Kuffner in [11, 10] and evolved into a powerful tool for path and motion planning since then. The two basic algorithms, *RRT-Extend* (see Alg. 1) and *RRT-Connect* (see Alg. 2), create trees covering collision-free configurations, thereby generating an approximation of $\boldsymbol{C}_{\text{free}}$. The *sampling parameter* ε regulates the granularity of the configurations lying on path segments.

RRTs can be constructed in a unidirectional or bidirectional way, with the latter leading to significantly more efficient behavior [10]. Alg. 3 depicts a bidirectional variant implementing two individual RRTs which are alternatingly extended. This algorithm is analogous to the approach in [10].

The RRT approach can be further extended, for instance, by multiple search trees [4] and interpolation between configurations (*Rapidly-exploring Dense Trees* in [12]). An exemplary illustration of a two-dimensional planning task is given in Fig. 3.

Algorithm 1: The *RRT-Extend(RRT, c, ε)* algorithm.

```
1  $c_{nn} \leftarrow NearestNeighbor(RRT, c)$
2  $l \leftarrow |c_{nn} - c|$
3  if $l = 0$ then
4      return $c$
5  if $l > \varepsilon$ then
6      $c' \leftarrow c_{nn} + \frac{\varepsilon}{l}(c - c_{nn})$
7  else
8      $c' \leftarrow c$
9  if $PathCollisionFree(c_{nn}, c')$ then
10     $AddConfig(RRT, c')$
11     return $c'$
12 return $NULL$
```

Algorithm 2: The *RRT-Connect(RRT, c, ε)* algorithm.

```
1  repeat
2      $c' \leftarrow$ RRT-Extend$(RRT, c, \varepsilon)$
3      if $|c - c'| = 0$ then
4          return $true$
5  until $\neg c'$;
6  return $false$
```

Algorithm 3: BiRRT$(c_{start}, c_{goal}, \varepsilon)$

```
1  $AddConfig(RRT_1, c_{start})$
2  $AddConfig(RRT_2, c_{goal})$
3  while $\neg TimeOut()$ do
4      $c \leftarrow CreateRandomConfig()$
5      if $RRT\text{-}Extend(RRT_1, c, \varepsilon) \ \wedge \ RRT\text{-}Connect(RRT_2, c, \varepsilon)$ then
6          return $BuildSolution(RRT_1, RRT_2, c)$
7      $SwapTrees(RRT_1, RRT_2)$
8  return $NULL$
```

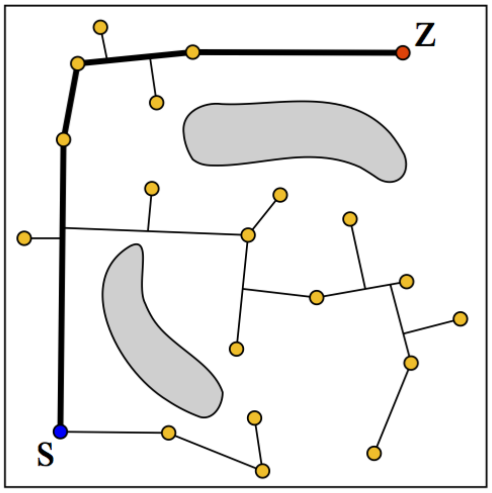

Figure 3: Exemplary illustration of the *Rapidly-exploring Random Tree* approach in a two-dimensional configuration space. The start and target configurations as well as the found solution are highlighted in the image.

3 Description of Exercises

```
Versuch 7/......................................Base directory
 ├─ CMakeLists.txt ..........................Project description
 ├─ PathPlanning.cpp..............................main() function
 ├─ PathPlanning.ui .............................. GUI definition
 ├─ PathPlanningWindow.cpp ............................ GUI logic
 ├─ PathPlanningWindow.h .............................. GUI logic
 ├─ Planner/
 │   ├─ Node.cpp
 │   ├─ Node.h
 │   ├─ PFPlaner.cpp..................Potential field implementation
 │   ├─ PFPlanner.h ........................ Potential field declaration
 │   ├─ Planner2D.cpp .... Implementation of the base class for planners
 │   ├─ Planner2D.h...........Declaration of the base class for planners
 │   ├─ RRTPlanner.cpp ................. RRT planner implementation
 │   └─ RRTPlanner.h........................RRT planner declaration
 ├─ build/ ..........................Directory the project is build in
 └─ data/
     ├─ *.xml .................................... Scene descriptions
     └─ *.iv.....................3D models (OpenInventor file format)
```

Figure 4: The directory structure for the exercise.

Preparation Before you start with the exercises (i.e., at home), you should do some preparations:

1. In this experiment, you will use the simulation toolbox *Simox*. To get an idea of how Simox works, read the documentation of the *VirtualRobot* library: `https://gitlab.com/Simox/simox/-/wikis/VirtualRobot`.
2. Study the fundamentals and read this document carefully.
3. Do the exercises 1 – 3 at home and write down the results.
4. Optionally: Install Simox from *http://gitlab.com/Simox/simox* and have a look at the examples.

3.1 Exercise One: Potential Fields

- Write down a pseudo code implementation of the potential field approach.
- What are the benefits and disadvantages of the potential field approach?
- What can be done to avoid the local minima problem, which problems may arise? Write down some pseudo code.

3.2 Exercise Two: Dijkstra's Algorithm

- Pseudo Code: Write down a pseudo code implementation of the Dijkstra's algorithm. Consider a preprocessing step.
- What are the benefits and disadvantages of this approach?
- What problems can arise and how can they be solved?

3.3 Exercise Three: Rapidly-Exploring Random Trees

- Pseudo Code: Write down a pseudo code implementation of the unidirectional RRT algorithm.
- What are the benefits and disadvantages of this approach?
- Which problems can arise and how can they be solved?

3.4 Exercise Four: Implement one of the path planning algorithms at the lab.

- Implement the RRT approach using the provided code skeleton.
- Verify and evaluate (success rate, length of path, runtime) your implementation by randomizing start and goal position.

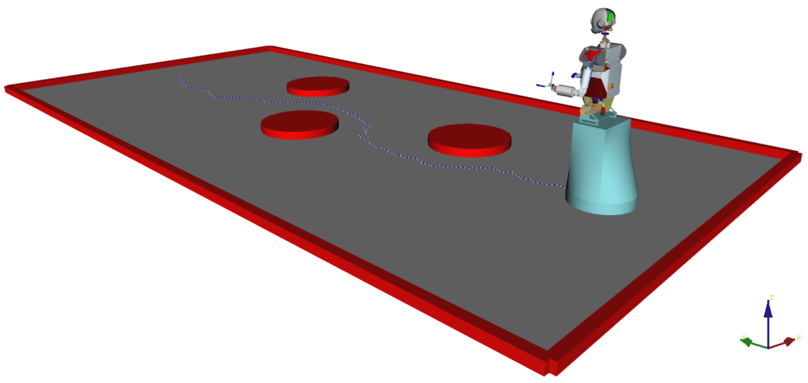

Figure 5: Screenshot of the final planned route around the obstacle.

References

[1] J. Barraquand and J.-C. Latombe. A monte-carlo algorithm for path planning with many degrees of freedom. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1712–1717 vol.3, May 1990.

[2] Jérîme Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *Int. J. Rob. Res.*, 10(6):628–649, December 1991.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] Matthew Clifton, Gavin Paul, Ngai Kwok, Dikai Liu, and D-L Wang. Evaluating performance of multiple rrts. In *Mechtronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME International Conference on*, pages 564–569. IEEE, 2008.

[5] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC*, 4(2):100–107, 1968.

[7] Takeo Kanade and Yangsheng Xu. *Space robotics : dynamics and control.* Boston : Kluwer Academic Publishers, 1993.

[8] O Khatib and J. F. Le Maitre. Dynamic control of manipulators operating in a complex environment. In *In Proceedings of the 3rd CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, pages 267–282, 1978.

[9] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[10] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[11] Steven M LaValle. Rapidly-exploring random trees a new tool for path planning. Technical report, Computer Science Dept., Iowa State University, 1998.

[12] Steven Michael LaValle. *Planning algorithms.* Cambridge university press, 2006.

[13] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[14] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, October 1979.

[15] Nils J Nilsson. Shakey the robot. Technical report, DTIC Document, 1984.

[16] Brad Paden, A Mess, and Mike Fisher. Path planning using a jacobian-based freespace generation algorithm. In *Robotics and Automation,*

1989. Proceedings., 1989 IEEE International Conference on, pages 1732–1737. IEEE, 1989.

[17] Jacob T Schwartz and Micha Sharir. On the "piano movers'" problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, 36(3):345–398, 1983.