ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Bachelor Thesis

Path Planning for an Omnidirectional Underwater Robot

Autumn Term 2016

Supervised by:
Gregory Hitz
Philipp Krüsi

Author:
Johann Diep

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Your Project Title

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Johann	Diep

Student supervisor(s)

Gregory	Hitz
Philipp	Krüsi

Supervising lecturer

Roland	Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Preface

At this point I would like to thank everyone who supported me during this bachelor thesis. First of all, I would like to express my gratitude to Professor Dr. Roland Siegwart and the Autonomous Systems Lab (ASL) for their continuous support of the focus project Scubo. Special thanks to my supervisors Dr. Gregory Hitz and Philipp Krüsi for guiding me successfully throughout the whole project. Last but not least, I would like to thank the whole Scubo team. It was an amazing roller coaster ride and I am very grateful to be a part of it since the very beginning.

Zurich, June 2016

Johann Diep

Abstract

Scubo is a submersible ROV (Remotely Operated Vehicle) developed in the context of a focus project at ETH Zurich. Due to the arrangement of eight non-moving propulsion units and the superposition of its forces, it is able to move in any direction without prior change of its orientation. Besides, it is equipped with six cameras pointing in each direction and the necessary hardware for autonomy. Several bachelor theses are concerned with the software implementation of an autonomous path following system.

This bachelor thesis discusses the implementation of an offline global path planning algorithm for the Scubo robot. The algorithm uses a point cloud[2] of the environment to generate a set of collision-free waypoints between an initial and a goal state for the trajectory controller to follow. Since the robot is capable of omnidirectional movements, the generated path does not have to satisfy any kinematic or dynamic constraints of the system other than avoiding collision with obstacles.

At first, the basic idea behind available path planning methods are explained on a conceptual level. In a second step, the implementation of the path planning algorithm using a variant of the rapidly-exploring random tree (RRT) method is presented in more detail. Lastly, the algorithm is used in a practical application where the goal is to scan an object from all sides under different angles. In this case, the developed path planner in combination with a traveling salesman problem solver proves to be highly suitable to generate a collision-free and cost-efficient path around an object.

[2] A point cloud is a set of data points representing positions of features. It can be generated by post-processing recordings of a stereo camera and an IMU with a mapping software.

Symbols

Acronyms and Abbreviations

ASL	Autonomous Systems Lab
ETH	Eidgenössische Technische Hochschule
IMU	Inertial Measurement Unit
k-d	k-dimensional
LCA	Label Correcting Algorithm
LKH	Lin-Kernighan Heuristic
PRM	Probabilistic Roadmap
ROV	Remotely Operated Vehicle
RRT	Rapidly-Exploring Random Tree
RRT*	Rapidly-Exploring Random Tree Star
SLAM	Simultaneous Localization and Mapping
VIO	Visual-Inertial Odometry
VR	Virtual Reality

Chapter 1

Introduction

This section gives the reader a brief overview of the motivation, the context and the scope of this bachelor thesis.

1.1 Motivation

Scubo is a submersible robot (see Figure 1.1) developed in the scope of a focus project at ETH Zurich. During two semester, a team consisting of mechanical and electrical engineering students had the opportunity to build a functional prototype of a robot. The goal of the project was to design a manually controlled submersible robot which is capable for 3D reconstruction of single corals. By taking pictures from different sides and angles of the underwater object, a detailed model can be obtained in post-processing by feeding these pictures to a suitable third party software like Pix4D[1]. This way, changes of coral reefs can be well observed and documented. Furthermore, the robot should provide telepresence for exploration of underwater landscapes. With the help of six on-board cameras, the user has access to a visual feedback from all sides of the robot through a VR-headset. The overall goal for the future is a further development of Scubo into a fully autonomous system. This has the following benefits:

- Common image processing software is based on finding keypoints[2] between images, which will generate 3D points. By increasing the overlap between two images, the common area captured is larger and more keypoints can be detected and matched together. Consequently, a rule of thumb is to maintain high overlap between the images. This can be done by keeping a constant distance while slightly varying the angle to the object. Therefore, scanning corals via remote control is difficult since the user has to control the distance as well as the orientation of the robot to the object. This task can be simplified by generating an efficient trajectory consisting of position and orientation information around that object for the autonomous system to follow.

- One possible application of Scubo and its virtual reality feature is the deployment in large aquariums of entertainment parks. These are narrow environments containing static as well as dynamic obstacles such as glass walls and underwater creatures. Collision in such sensitive surroundings would endanger the safety of the audience and the animals. This problem can be prevented by an autonomous system that is able to follow a predefined collision-free path.

[1] https://pix4d.com/

[2] Characteristic points detected in an image.

In this way, the audience can observe the environments through a VR-headset without having to control the robot.

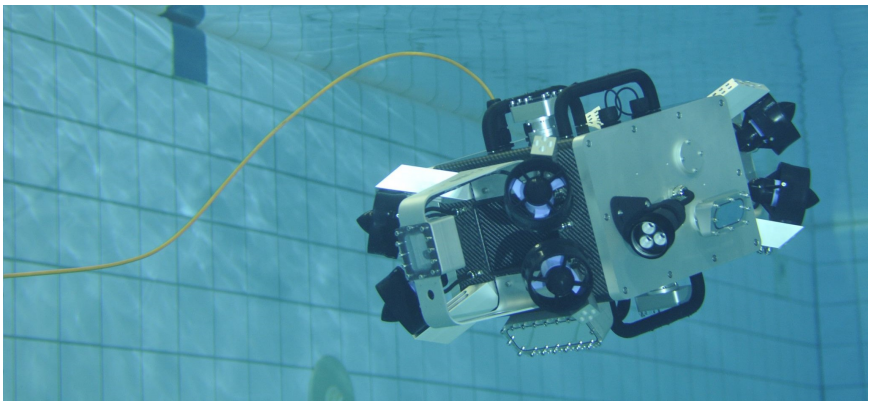

Figure 1.1: The SCUBO robot in action.

1.2 Context

This bachelor thesis discusses planning algorithm and aims to complement the work of the focus project SCUBO. Together with several other bachelor theses, its goal is to build the foundation for an autonomous system based on computer vision. In the following an overview of the theses and their associations is given:

- Underwater Camera Calibration: The process of determining all the necessary parameters for the image rectification in underwater application.
- Localisation and Mapping: State estimation and 3D reconstruction of the environment to a point cloud by post-processing recordings of the calibrated stereo camera and an IMU (VIO-sensor).
- Path Planning: Implementation of an algorithm which generates a collision-free trajectory from a starting to a destination point using the previous generated point cloud.
- Trajectory Controller: Design of controller needed to follow the generated set of waypoints.

1.3 Goal

The goal of this bachelor thesis is the development of an offline path planning algorithm as a basic for a semi-autonomous[3] system. The designed algorithm should provide a set of global waypoints of position informations from start to finish while avoiding collision with obstacles. Furthermore, a method should be found to generate a trajectory around an object for scanning purpose. The following framework conditions apply:

- The visible areas by the robot is represented by a certain space called the workspace. The robot itself is represented in the workspace by its centre of mass point.

[3]The planning algorithm of a semi-autonomous system requires the complete map in advance and does not calculate the set of waypoints in real-time.

- Obstacles in the form of a point cloud represents not accessible states.
- Starting and goal state are predefined input variables.

Chapter 2

Related Work

This chapter gives a brief overview of available planning methods. One distinguish between roadmap, cell decomposition, potential field, sampling-based and graph-search planning methods[1]. A detailed description can be found in (Siegwart, 2004, [2]), (Karaman, 2011, [1]) and (Bertsekas, 1995, [3]).

2.1 Roadmap Path Planning

This method is based on constructing a graph network consisting of lines in the obstacle-free configuration space[2]. The actual planning is then reduced to connecting the initial and goal state of the robot to the network and searching for feasible paths between them. In the following, two roadmap approaches are described.

2.1.1 Visibility Graph

The visibility graph is mostly suitable and highly efficient for sparse 2D configuration spaces with polygonal obstacles. The nodes of the graph include the vertices of the obstacles as well as the initial and final point. Straight-line segments connect every nodes that are visible from each other (i.e. sight is not blocked by an obstacle), which also includes the edges of the polygons. Thus, the optimal path in terms of length between start and goal state can be found (see Figure 2.1). The implementation of the visibility graph is straight forward, however, there are two major drawbacks of this method. First, the computational effort increases with the number of obstacles, which lowers the efficiency of the algorithm in dense environments. Second, the algorithm tends to generate a path as close as possible to nearby obstacles. However, the latter can be solved by expanding the size of the obstacles or using another roadmap method called the Voronoi diagram.

2.1.2 Voronoi Diagram

In contrast with the visibility graph method, a Voronoi diagram (see Figure 2.2) maximizes the distances between the robot and the obstacles in the configuration space (including the boundaries). It consists of equipotential line-segments constructed from points that are equidistant from two or more obstacles.

[1] A similiar classification can be found in the bachelor thesis Robotic Arm Motion Planning by Markus Brändle

[2] Vector space defined by the generalized coordinates of a system.

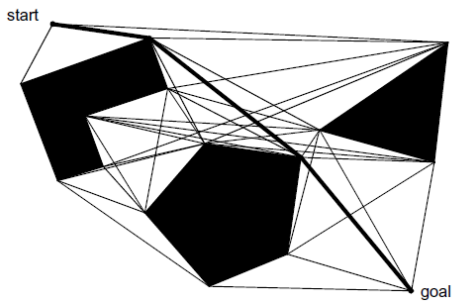

Figure 2.1: Visibility graph, 'this image is taken from [2]'.

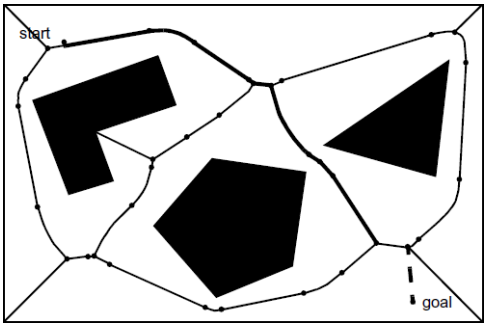

Figure 2.2: Voronoi diagram, 'this image is taken from [2]'.

2.2 Cell Decomposition Path Planning

The general idea of the cell decomposition method can be summarized as follows:

- splitting of the configuration space into smaller non-overlapping regions called *cells*,
- constructing a *connectivity graph* according to the cell-conjunctions,
- searching for a sequence in the connectivity graph that joins the initial with the goal cell,
- computation of a path passing through each cell of the sequence.

Depending on the cell division, one distinguish between exact and approximate cell decomposition.

- Exact cell decomposition: Boundaries of the cells corresponds to the structure of the environment (see Figure 2.3).
- Approximate cell decomposition: Division into cells results in an approximation of the map.

2.3 Potential Field Path Planning

The idea behind this method is to set up a gradient across the map, such that the robot, represented by a point, is attracted toward the goal, while being repulsed by the obstacles. As one can see in Figure 2.4, the resulting field (d) is thus the

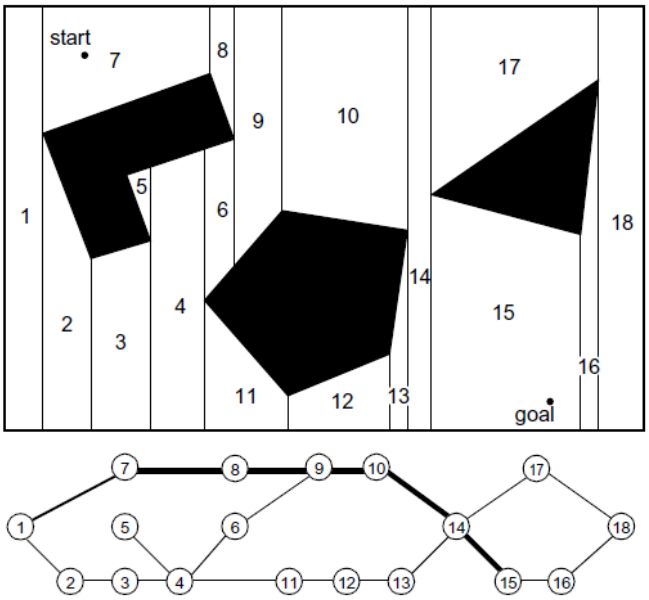

Figure 2.3: Exact cell decomposition, 'this image is taken from [2]'.

sum of a attractive (b) and a repulsive potential field (c). This method guides the robot smoothly towards its goal position while avoiding collision with obstacles (e). Potential field path planning can be used in high-dimensional configuration spaces. However, one major drawback of potential fields is the formation of local minima, in where the robot gets trapped.

2.4 Sampling-Based Algorithms for Path Planning

These methods are using some form of sampling-based algorithm to produces potential configurations, which are subsequently checked on their feasibility by a collision detector. They can be easily adapted to high-dimensional configuration spaces. One distinguish between multi- and single-query methods.

- Multi-query methods are mostly suitable for static environments and consists of a preprocessing phase, where a roadmap is constructed, and a query phase, where a path is planned to connect the start and goal configuration.

- In single-query methods, the robot constructs the roadmap while simultaneously searching for a path between initial and goal configuration. Since these methods works online, they are capable for usage in dynamic environments.

2.4.1 Rapidly-Exploring Random Tree

RRT (see Figure 2.5) is a single-query path planning algorithm. It builds a tree of possible configurations that expands from the initial state and tends to cover the whole obstacle-free workspace. At each iteration, a new random node is sampled and the necessary measures are taken to connect it feasible to the tree. A modified version of this algorithm is used for the path planner of this bachelor thesis.

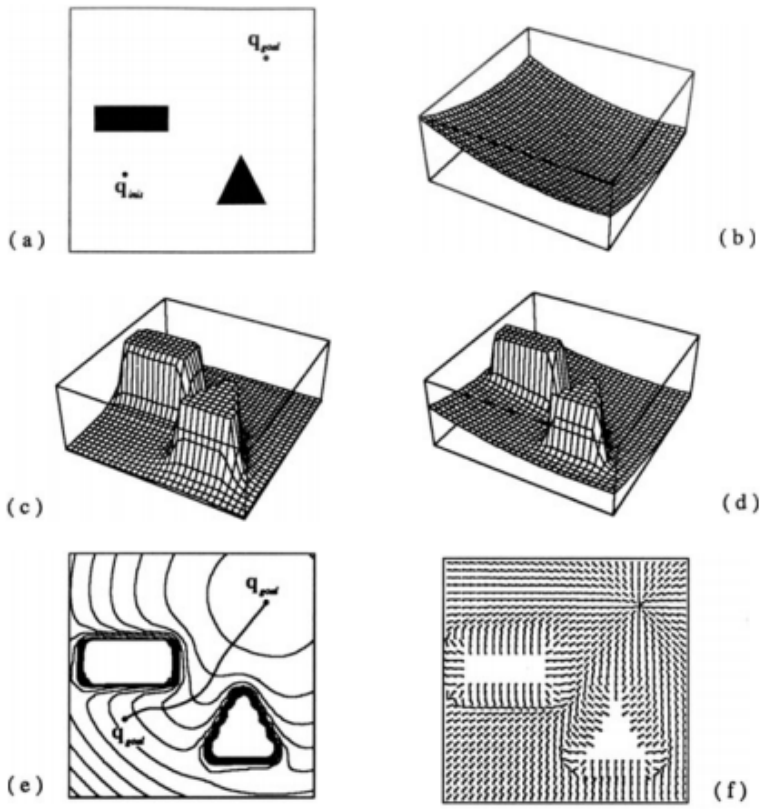

Figure 2.4: Different potential fields, 'this image is taken from [4]'.

2.4.2 Probabilistic Roadmap

Probabilistic roadmap (PRM) (see Figure 2.6) is a multi-query path planning method. In a first step, random samples from the configuration space of the robot are taken. Second, these samples are being checked whether they are located in the free space. In case of being in the obstacle free space, a local planner attempts to connect them to other existing nearby configurations. After the initial and goal states are connected to the constructed roadmap, a graph search algorithm (see also Section 2.5) is applied to determine a path between both nodes.

2.5 Graph-Search Algorithms

This section gives a brief overview of the most common common graph-search algorithms.

2.5.1 Lin-Kernighan Heuristic

The Lin-Kernighan heuristic is an algorithm which deletes existing connections between configurations and reconnects them in other possible ways to generate a more optimal solution. It is commonly used for tasks related to traveling salesman problem[3].

[3]Searching the shortest route through a list of states.

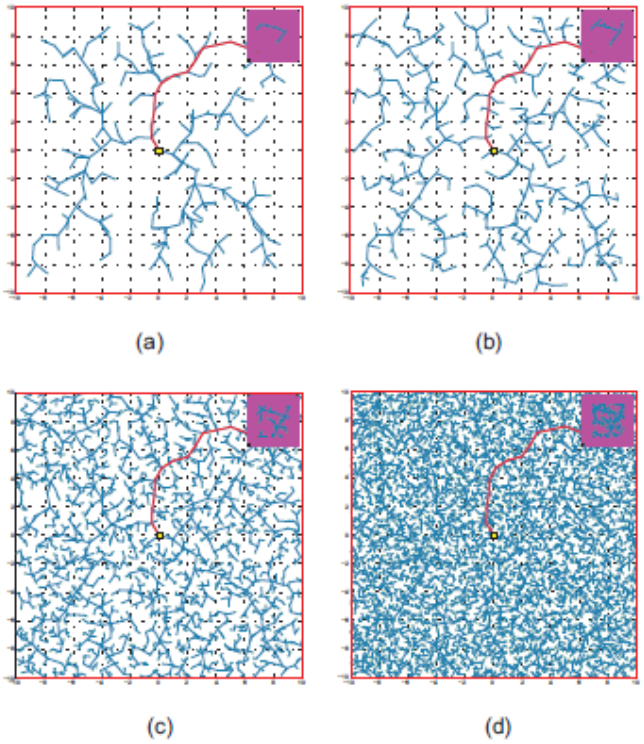

Figure 2.5: RRT tree after (a) 250, (b) 500, (c) 2500 and (d) 10'000 iterations, 'this image is taken from [1]'.

2.5.2 Greedy Algorithm

The Greedy algorithm makes the locally optimal choice at each node. In general, this method does not generate a global optimal sequence of configurations.

2.5.3 Label Correcting Algorithm

With the label correcting algorithm (LCA), the shortest path in a graph network between an initial and a terminal node can be found, assuming the transition cost a_{ij} between state i and j is positive. The algorithm can be summarized in 4 steps (detailed description of the variables and the steps can be found in [3]):

1. Place the initial node S in the OPEN BIN and set: $d_S = 0$ and $d_j = \infty \; \forall j$ (d_q: cost to move from state S to state q)
2. Remove a node i from the OPEN BIN and execute step 3 for all children j of i.
3. If $d_i + a_{ij} < min(d_j, d_T)$ (d_T is the final node), set $d_j = d_i + a_{ij}$ and i to be the parent of j. If $j \neq T$, place j in the OPEN BIN if it is not already there.
4. If the OPEN BIN is empty, the algorithm is done, else go back to step 1.

There are multiple options in selecting a node to be removed from the OPEN BIN in step 2, which results in different algorithms:

- Depth-first search: Last in, first out.

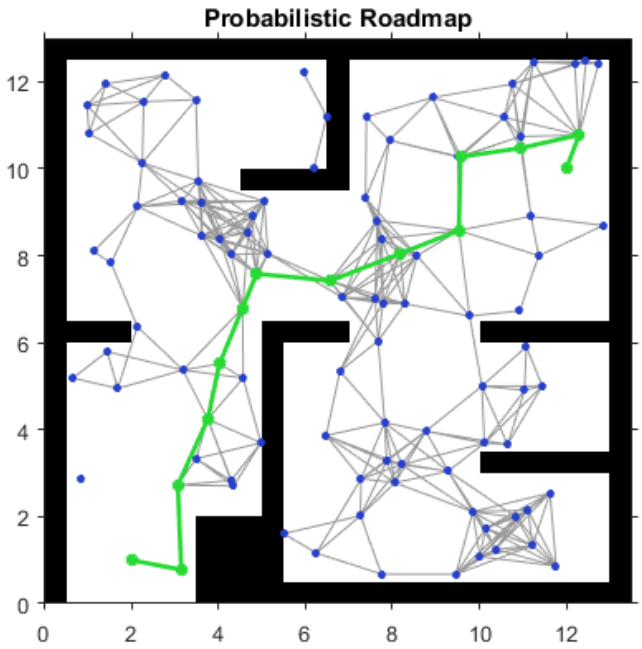

Figure 2.6: An example of a probabilistic roadmap with the path from an initial to a goal configuration marked in green, 'this image is taken from *Mathworks* under the topic Probabilistic Roadmaps'.

- Best-first search: Remove best label first (Dijkstra's method). A generalization of this algorithm is A*. By adding a lower bound to the cost function, non-optimal nodes can be excluded.
- Breadth-first search: First in, first out (Bellman-Ford).

Chapter 3

Algorithm Concepts

This chapter explains the concepts of the chosen algorithms for the offline global path planner and the path generation around an object in greater detail.

3.1 Offline Global Path Planning Algorithm

RRT* is a variation of the basic rapidly-exploring random tree algorithm as described in 1. Similiar to its predecessor, the algorithm is able to search high-dimensional configuration spaces by randomly building a tree starting from an initial state. With two further rewiring steps, the algorithm is asymptotically optimal. The following section will consider this offline global path planning algorithm step by step.

3.1.1 RRT* Algorithm

In Figure 3.1, a partially constructed tree is spreading from a starting configuration into the obstacle-free workspace. Furthermore, a goal state within a specific threshold area is defined. The red circles represent states, which consists of informations on their positions.

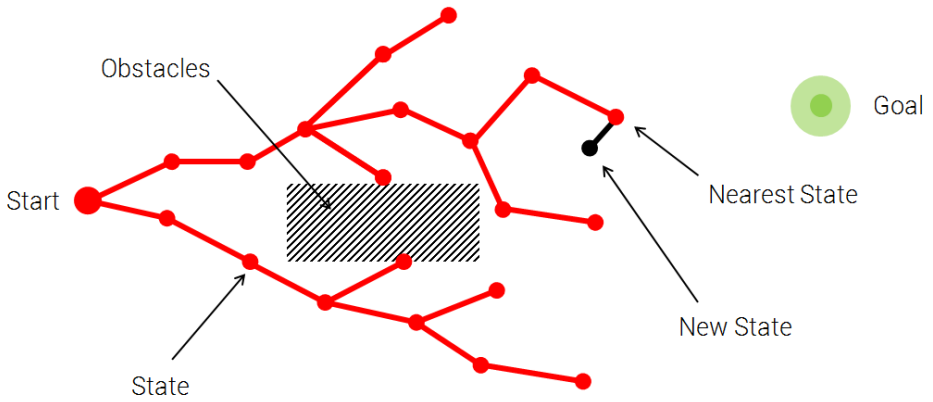

Figure 3.1: A new node (black) is being added to the tree.

In a first step, a random state on the map gets sampled. Control is applied from the inherent nearest state to get closer to the sample. In this case, the stepsize is added in the direction of the sample in order to generate a new state. If this connection does not intersect an obstacle, the new state is added to the tree. Otherwise, the current iteration terminates and the process starts from the beginning again by sampling a new configuration. In a further step, new possible parents within a defined neighbourhood radius are considered for the new state. A rewiring is performed, if the new connection is collision-free and results in a path with minimum cost. Thus the current edge gets replaced in order to still maintain a tree structure. Figure 3.2 shows an example of this process. In Figure 3.3, the final rewiring step is taking place. Edges are created from the new state to its possible children vertices, if the path through the new states results in a lower cost than the path through the current parent of the children. At this point, the current iteration is completed and the process starts over again. The procedure is repeated, until a feasible path from the initial to the goal configuration is found (see Figure 3.4). Since RRT* is an asymptotically optimal path planning algorithm, this solution gets updated with further iterations.

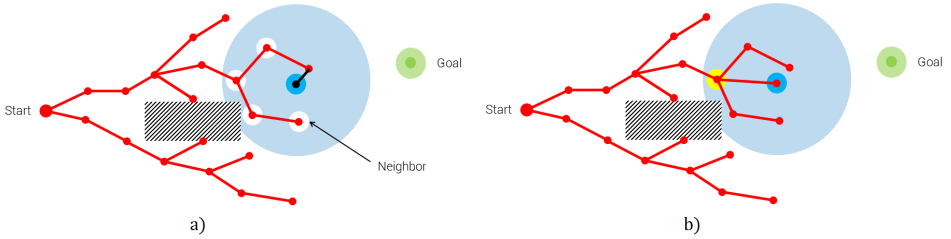

Figure 3.2: Searching for new parent nodes: a) potential states and b) rewiring

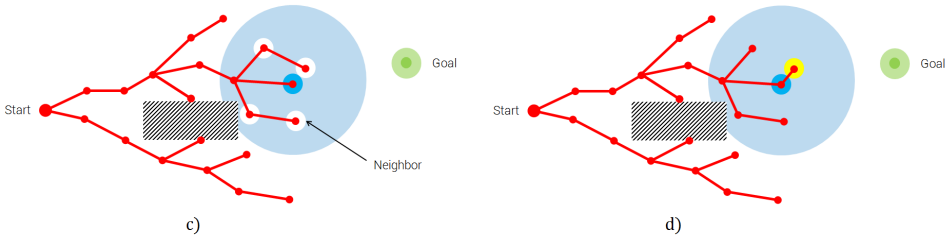

Figure 3.3: Searching for new children nodes: c) potential states and d) rewiring

Search Radius

The search radius r_{Search} is a non-constant variable and is calculated with the following formula (Formula 3.1):

$$r_{Search} = min(\gamma_{RRT^*}(log(card(V))/card(V)^{1/d}, \eta) \tag{3.1}$$

$$\gamma_{RRT^*} > (2(1+1/d))^{1/d}(\mu(X_{free})/\xi)^{1/d} \tag{3.2}$$

- γ_{RRT^*} is a constant and is chosen such the inequality (Formula 3.2) is met. On the basis of a theorem, the fulfilment of this requirement results in the asymptotic optimality of the RRT* algorithm,

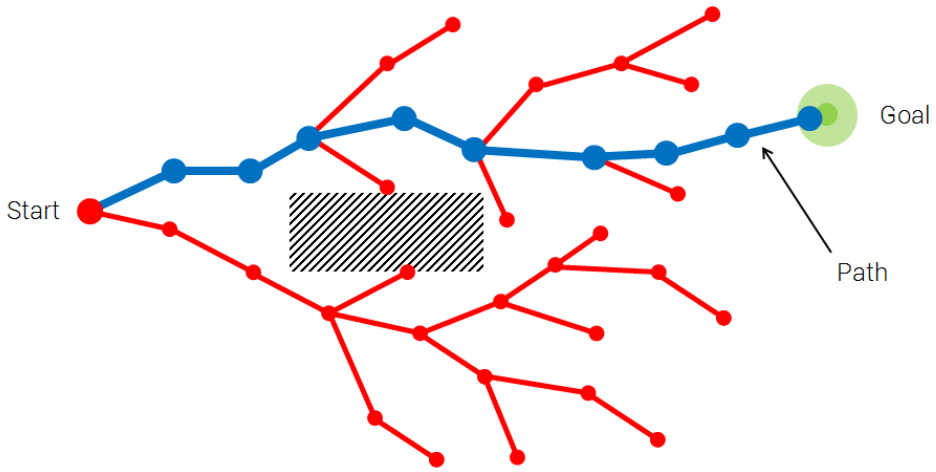

Figure 3.4: A feasible path between start and goal state.

- $\mu(X_{free})$ defines the Lebesgue measure[1] of the obstacle-free space,
- ξ is the volume of the unit sphere[2],
- d is the dimension of the workspace,
- η corresponds to the stepsize,
- $card(V)$ defines the current number of configurations available in the workspace.

The search radius (see Formula 3.1) decreases logarithmically (see Figure 3.5) with the number of available configuration in the workspace. This choice is logical, since there are less required rewiring in a highly populated map needed. Furthermore, the computational effort of the collision-check function as well as the search of potential neighbours for the rewiring step gets released with a decrease of the search radius, which shortens the overall runtime of the algorithm.

3.2 Path Generation around an Object

The second goal of the thesis is the generation of a feasible path around an underwater object. Summarized, the process works as follows.

- First N viewpoints around the object are manually calculated,
- all possible paths between them are generated,
- the order of visiting the viewpoints is figured out.

3.2.1 Viewpoints Calculation

The N viewpoints are set, such that the front camera of *Scubo* is able to cover the whole object in one frame. Thus, the distance to the object was estimated with the horizontal and vertical angle of view of the camera.

[1] Measure that associate objects in the euclidean space with their content (length, area, volume)
[2] Sphere with unit radius in d-dimensional space (length, circle, sphere)

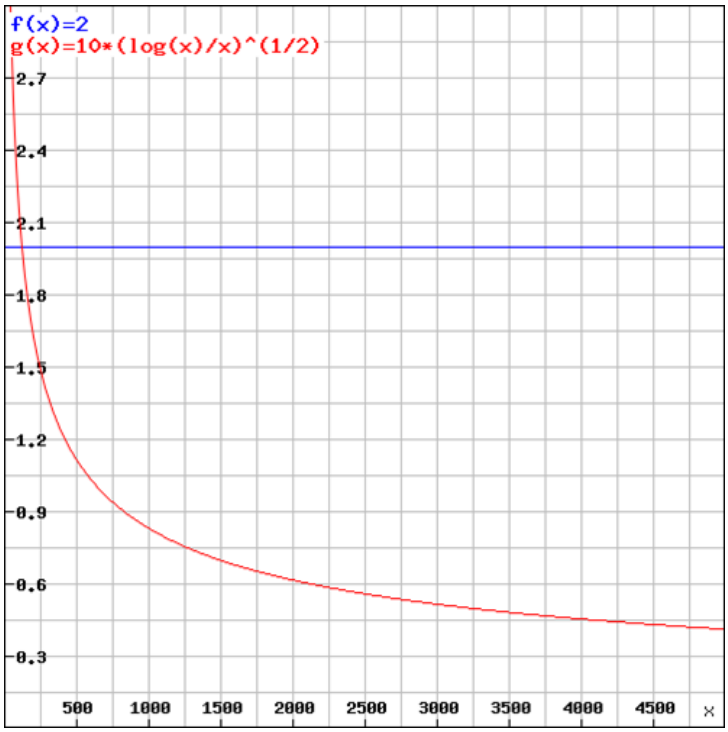

Figure 3.5: Plot of the search radius ($\eta = 2, \gamma_{RRT^*} = 10, d = 2$, cardinality on the x- and radius on the y-axis)

3.2.2 Paths and Cost Matrix Generation

In Figure 3.6, the viewpoint 1 is considered first. Feasible paths to other viewpoints are calculated using the RRT* algorithm (see Section 3.1.1). The costs, which are defined by the length from one viewpoint to another are stored in a $N \times N$ matrix (see Figure 3.6).

3.2.3 Traveling Salesman Problem

With the information about the viewpoints and the costs between them, the task of object scanning is reduced to a traveling salesman problem. The goal is to calculate the shortest possible route that visits each viewpoint exactly once and returns to the starting position. Large amount of heuristic and exact methods are able to approximate a solution of the problem. In the following, two of them are shortly introduced:

- Greedy Algorithm: Makes the locally optimal choice at each stage. Thus, tends to find a local rather than a global optimum.
- Lin-Kernighan Heuristic (LKH): Randomly swapping pairs of sub-tours in order to generate a new more cost-optimal tour (see figure 3.7).

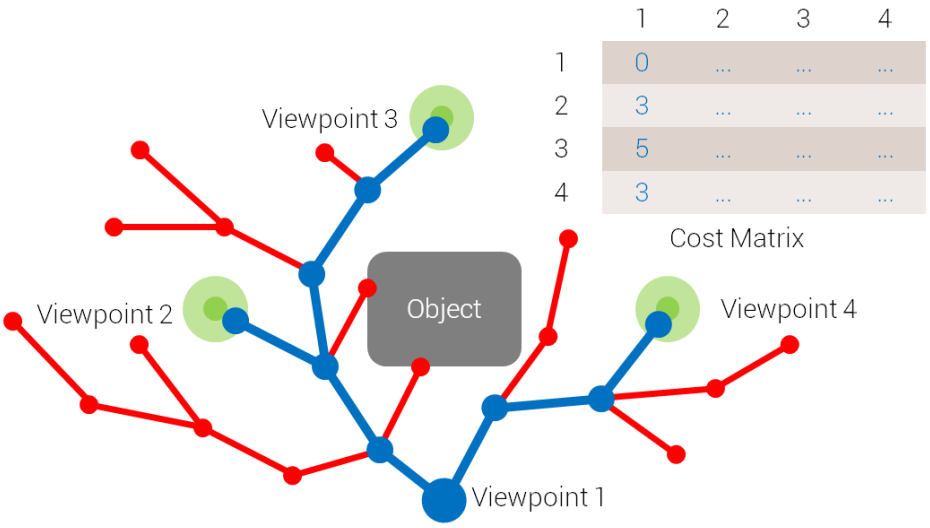

Figure 3.6: Feasible paths to each viewpoint are calculated with the RRT* algorithm. The length of the paths are stored in a cost matrix.

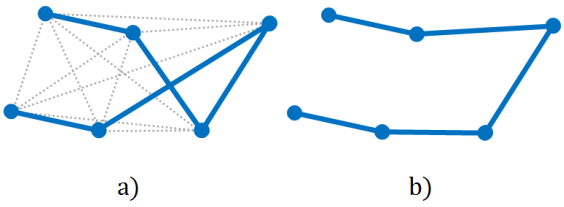

Figure 3.7: Lin-Kernighan Heuristic: a) possible sub-tours in grey, b) optimal tour through eliminating cross-tours

Chapter 4

Implementation Keypoints

This chapter gives an overview of the keypoints regarding the implementation of the methods illustrated in Chapter 3.

4.1 2D-RRT*

The RRT* path planner was firstly implemented on a 2D plane.

4.1.1 Runtime Improvement

The main challenge here was the runtime, which increases exponentially with the number of iterations. As a reference, one of the first implemented versions took approximately 30 minutes to compute 10'000 iterations. This is due to the following reasons:

- The collision-check function takes two states as an input and scans single points along the connection on collision with a specific check resolution. As one can see in Figure 4.1, the smaller the resolution gets, the higher the probability of detecting a potential threat is. However, a small resolution results in long runtime for the collision-check function. Since this function gets called at least once per iteration, it has strong influence on the overall calculation time of the path planner.
- At high amount of configurations in the workspace, the search of the nearest neighbour as well as the search of potential neighbours for the rewiring step becomes time-consuming, since the algorithm has to check the distance to each available state one by one.

Both problems can be kept within a limit by the implementation of the variable search radius described in Chapter 3.1.1. Regarding the collision-check function, the runtime could be improved by increasing the check resolution. However, this would take place at the expense of safety and is thus not recommendable.

k-d-tree

A method called k-dimensional-tree (k-d-tree) can be used to reduce the searching work. It is a data-structure for organizing k-dimensional points by their attributes. As in the example of Figure 4.2, the idea is to divide the configuration space into smaller subgroups. This is done by repeatedly splitting the space at the median of different attributes, which can be represented by a tree. By arrival of a new point,

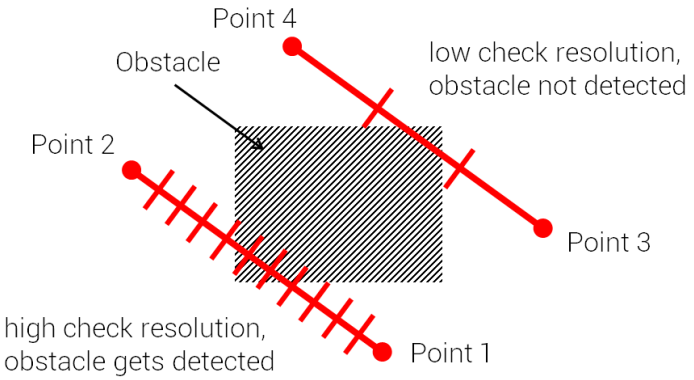

Figure 4.1: Influence of the check resolution on obstacle detection.

it only needs to be sorted based on its attributes into the corresponding subgroup. This way, the search can be restricted and the runtime improved. This method is an approximating technique and in some cases, errors in the search of the nearest neighbour can occur. In *MATLAB*, k-d-tree is already implemented in the Statistics and Machine Learning Toolbox[1].

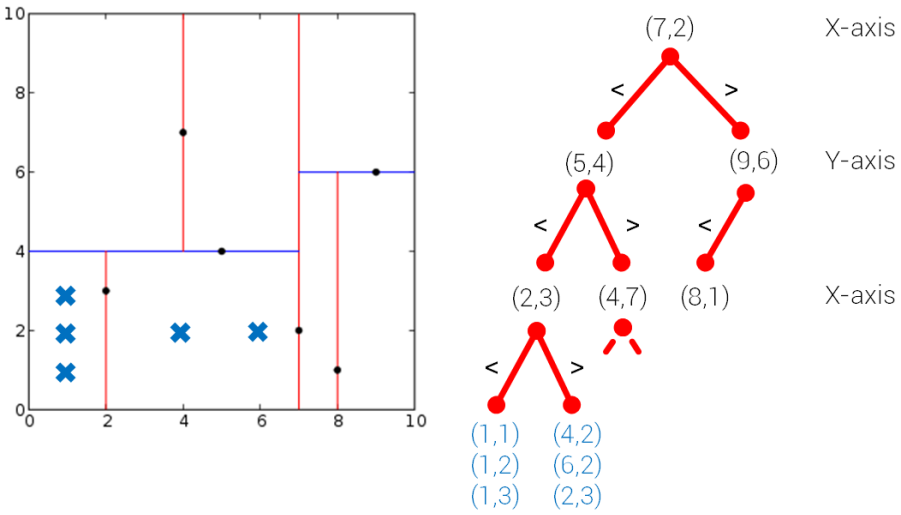

Figure 4.2: Example of a 2-d-tree.

Matrix Operation

Since *MATLAB* is designed for numerical calculation using matrices, the run time can be minimized by using matrix operations instead of for-loops. In the following, two code parts are shown which shows the search for the nearest state (see Code 4.1) and the neighbours (see Code 4.2). Furthermore, a runtime test showed that matrix

[1] http://ch.mathworks.com/help/stats/index.html

operations is slightly more time-efficient at high iteration number than k-d-tree (see figure 4.3).

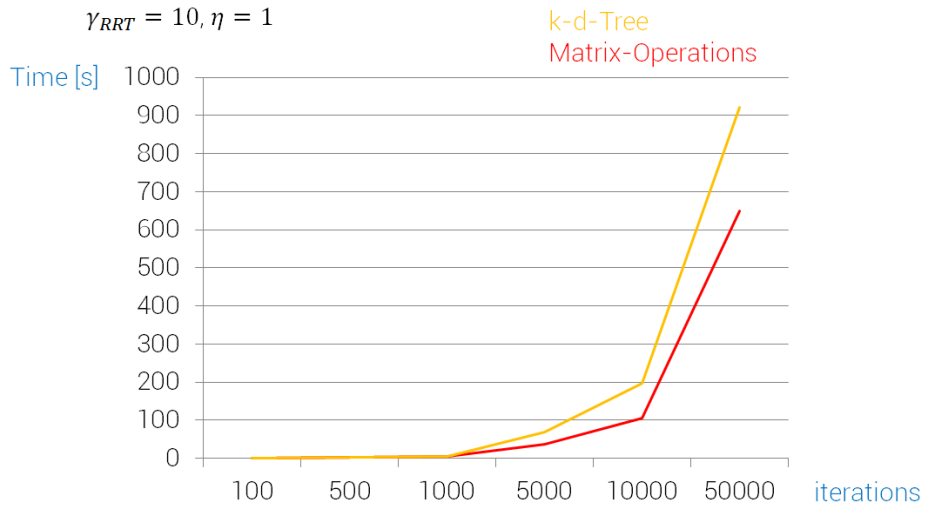

Figure 4.3: Calculation time: k-d-tree and Matrix Operations in comparison

Code 4.1: Nearest State Search

```
difference_array=states_array(:,1:2)-repmat([x_sample,↩
    y_sample],size(states_array,1),1);

distance_array=sqrt(sum(difference_array.^2,2));

[~,idx]=ismember(min(distance_array),distance_array);
```

Code 4.2: Neighbours Search

```
distance_array=states_array(:,1:2)-repmat([states_array↩
    (i,1),states_array(i,2)],size(states_array,1),1);

distance_from_neighbor=sqrt(sum(distance_array.^2,2));

neighbors=find(distance_from_neighbor<=search_radius);
```

4.1.2 Path Curvature

In a further step, the influence of the cost function on the curvature of the path was evaluated. Both, the euclidean distance as well as the angular change between line segments were considered with corresponding weight-factors in the step cost function (see formula 4.1). The angle between two connections were considered in square in order to penalize large angular changes.

$$cost_{step} = w_1\, d + w_2\, \alpha^2 \tag{4.1}$$

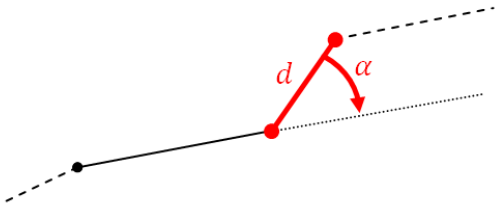

Figure 4.4: Angle α between two connections and distance d.

4.2 3D-RRT*

The implemented algorithms and functions for the 2D plane was adapted for the 3D configuration space. The main problem was the implementation of a 3D occupancy grid, which will be illustrated in the following section.

4.2.1 3D Occupancy Grid

Since *Octomap*[2] does not exist for *MATLAB* and there were no other 3D occupancy grid implementation in the form of a toolbox, the main challenge in the 3D configuration space was the development of a function to differentiate between free and occupied areas. Furthermore, it should be able to use a point cloud (see Figure 4.5) as an input to generate a discretized obstacle model for the planner.

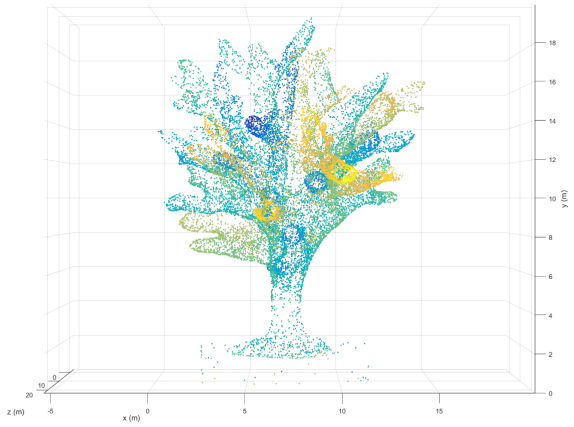

Figure 4.5: Pointcloud model of a coral.

Visualization

The visualization of the discretized model was done in two steps. The whole configuration space was divided with a certain resolution into single cubical cells. Cells which are occupied by at least one data point from the point cloud were highlighted.

[2] 3D mapping framework based on octrees.

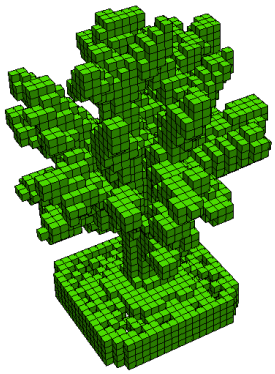

Figure 4.6: Discretized model of the coral.

Collision-Check

For collision-checking, the workspace was transformed into a three dimensional matrix according Formulas 4.2 to 4.4. The size corresponds to the number of cells in the configuration space. Obstacle-free cells carry the value 0 while occupied cells the value 1.

$$occupancy = zeros(m_{length}/b_{res}, m_{width}/b_{res}, m_{height}/b_{res}) \tag{4.2}$$

$$[x_{occ}, y_{occ}, z_{occ}] = [ceil(x_{PC}/b_{res}), ceil(y_{PC}/b_{res}), ceil(z_{PC}/b_{res})] \tag{4.3}$$

$$occupancy(x_{occ}, y_{occ}, z_{occ}) = 1 \tag{4.4}$$

- m_{length}, m_{width} and m_{height} are the map sizes,
- x_{PC}, y_{PC} and z_{PC} are the datapoints from the point cloud,
- b_{res} is the map resolution respectively the length of a single cubical cell.

4.3 Object Scanning Route

The implemented algorithm was used for an object scanning application. For this, the sampling function needed to be rewritten in order to generate solutions within a short amount of time.

4.3.1 Defined Sampling Region

The RRT* algorithm is not able to generate a feasible paths between all viewpoints (see Figure 3.6) within a relatively short period of time and with a fixed iteration number. This problem was solved by rewriting the sampling function. Instead of sampling in the whole map, the new function samples with 50% in the goal region (see Figure 4.7). Therefore, a path is more likely to be found. Furthermore, the program is defined as such that it automatically terminates when all feasible paths are generated.

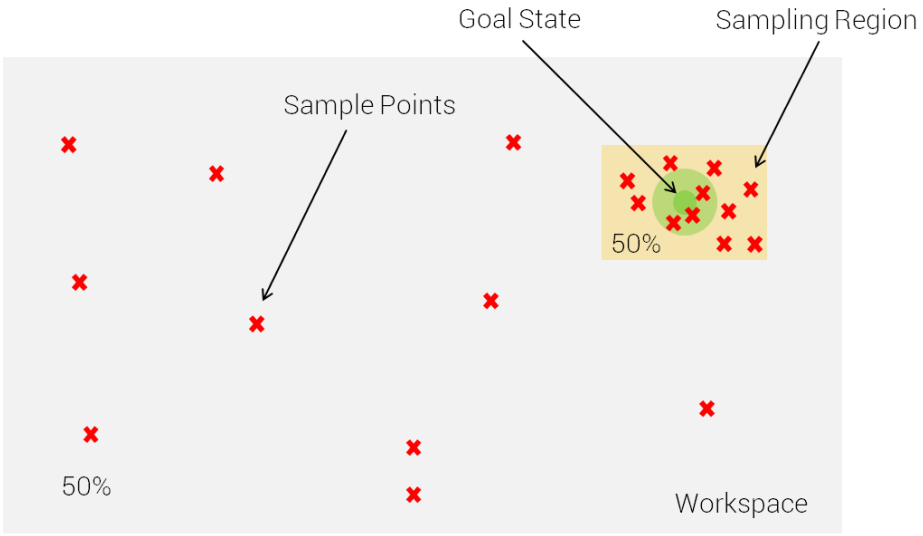

Figure 4.7: Defined sampling region.

Chapter 5

Results

The results of the developed algorithms will be presented in this chapter. The mentioned programs and functions can be found on the provided CD.

5.1 RRT* Path Planning in 2D

In this section, the results of the path curvature evaluation on the 2D plane are discussed. In the cost function, one considers euclidean distances on the one hand and angular changes on the other. A cost function consisting of a combination of both attributes does not bring any benefits and therefore was not considered. The m-file `RRTstar_Obstacles_2D.m` in the folder `Curvature_Evaluation` is used and the following parameters are chosen:

- 10×9.5 map
- $x_{initial} = 4.75$ and $y_{initial} = 2.75$
- $x_{initial\_parent} = 5$ and $y_{initial\_parent} = 2.75$ [1]
- $x_{final} = 4.75$ and $y_{final} = 6.75$
- $\gamma_{RRT^*} = 10$
- $\eta = 1$
- $iteration = 50'000$

5.1.1 Cost Function: Considering Euclidean Distance

- $w_1 = 1$ and $w_2 = 0$
- $140\,s$ to complete the calculation.

As one can see in Figure 5.1, the RRT* algorithm calculates a path, which follows mostly the edges of the obstacles. By inflating the obstacles in the order of the radius of *Scubo*, one can create buffer zones for the robot to move without hitting obstacles. Furthermore, the path takes sharp turns at edges. This is also not a problem for the real system, since the robot is omnidirectional and is able to move in any direction without prior change of its orientation.

[1] These additional variables are needed to define the heading in the initial configuration

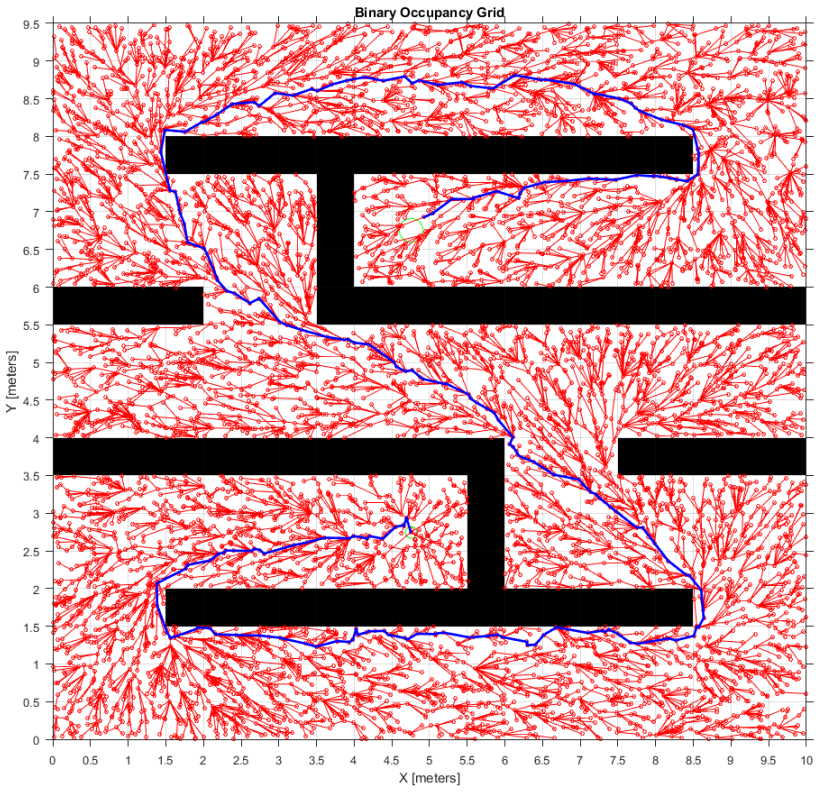

Figure 5.1: 2D RRT* Path: Considering euclidean distance in the cost function.

5.1.2 Cost Function: Considering Angular Change

- $w_1 = 0$ and $w_2 = 1$
- $102\,s$ to complete the calculation.

As expected, a much smoother path can be generated under the same condition by considering the angular change in the cost function (see Figure 5.2). Surprisingly, this task takes less time to compute than the one described in the previous Section 5.1.1.

5.2 RRT* Path Planning in 3D

The visualization in 3D is done with *Paraview*. The m-file `RRTstar_Obstacles_3D.m` is used to generate the map, the states and the resulting paths. With `VtkWriter.m`, the results are transformed into VTK-format in order to display it in *Paraview*. The following parameters are chosen:

- $10 \times 10 \times 10$ map
- $x_{initial} = 1$, $y_{initial} = 1$ and $z_{initial} = 1$
- $x_{final} = 1$, $y_{final} = 1$ and $z_{final} = 9$
- $\gamma_{RRT^*} = 10$

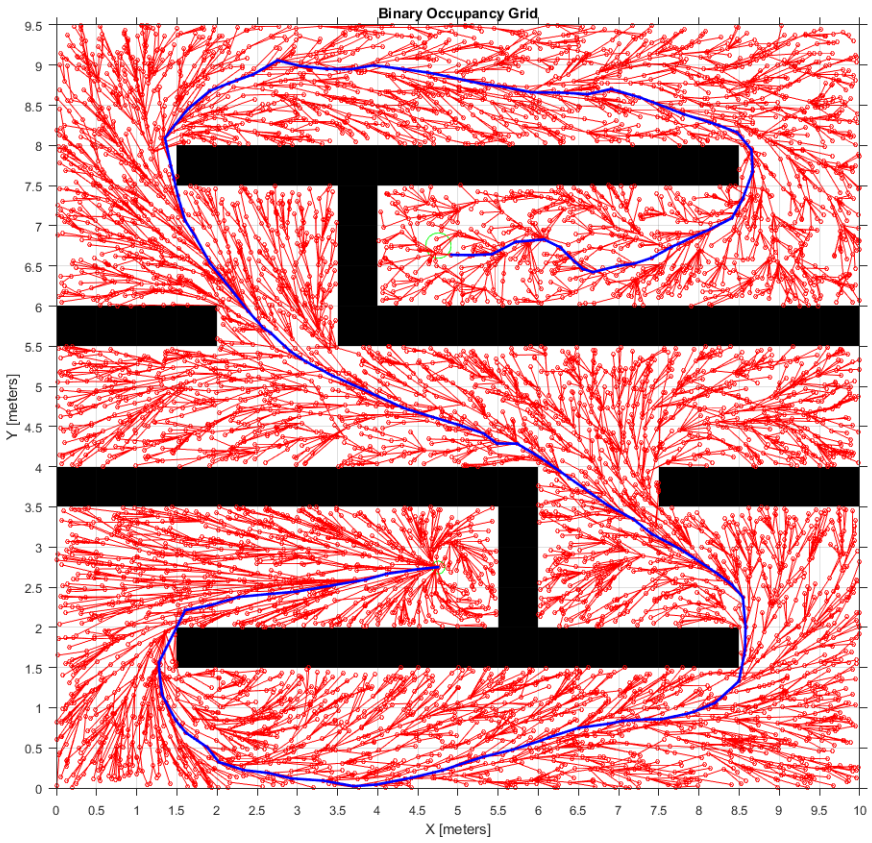

Figure 5.2: 2D RRT* Path: Considering angular change in the cost function.

- $\eta = 2$
- $iteration = 1000$

As one can see in Figure 5.3, the generated path in 3D is feasible. The red wall is an obstacle with a hole in the middle. The algorithm is able to find a path between start and goal through the obstacle.

5.3 Object Scanning

For object scanning, the following files were used. Point cloud datas were stored in `coral.vtr`. The file `occupancy_calculation.m` calculates the occupancy matrix regarding Formulas 4.2, 4.3 and 4.4. The viewpoints are set in `view_states_calculation.m` and the corresponding cost matrix and the possible paths between all viewpoints is computed in `traveling_salesman_cost.m`. The order of the viewpoints can be defined in two ways. The file `ts_greedy_algorithm.m` sets the order according the greedy algorithm. For a order by the LKH algorithm, one can use the file `compute_traveling_salesman.m` with the corresponding executables[2].

[2]http://webhotel4.ruc.dk/ keld/research/LKH/

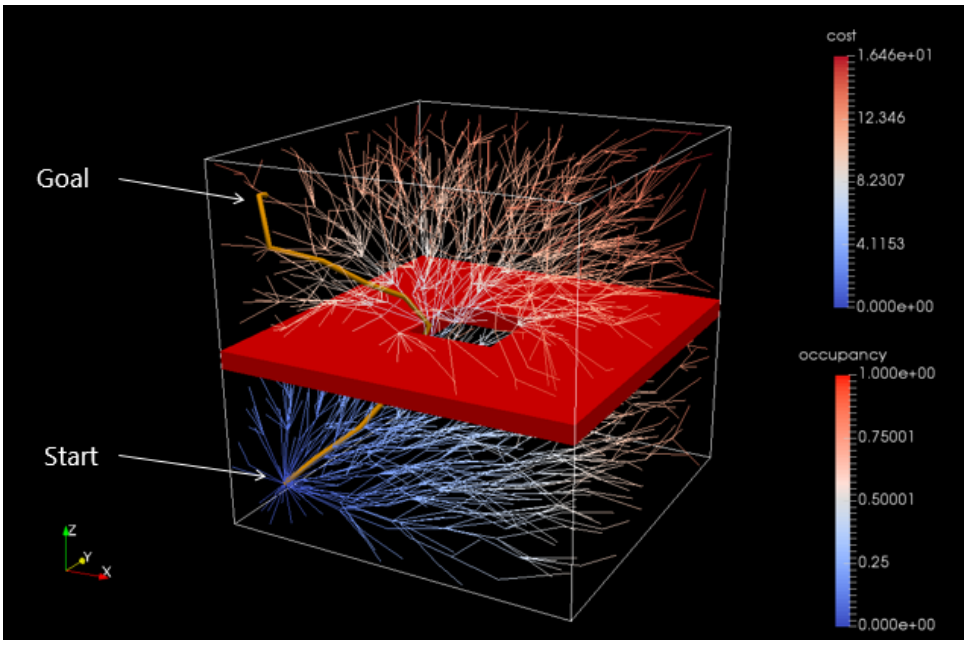

Figure 5.3: 3D RRT* Path

5.3.1 Route by Greedy Algorithm

As expected, the route generated by the greedy algorithm is not efficient, since the path go over the object two times (see Figure 5.4).

5.3.2 Route by LKH Algorithm

A much more efficient route can be generated by using the LKH algorithm. The path goes around the object in a reasonable manner and ends in the starting configuration (see Figure 5.5).

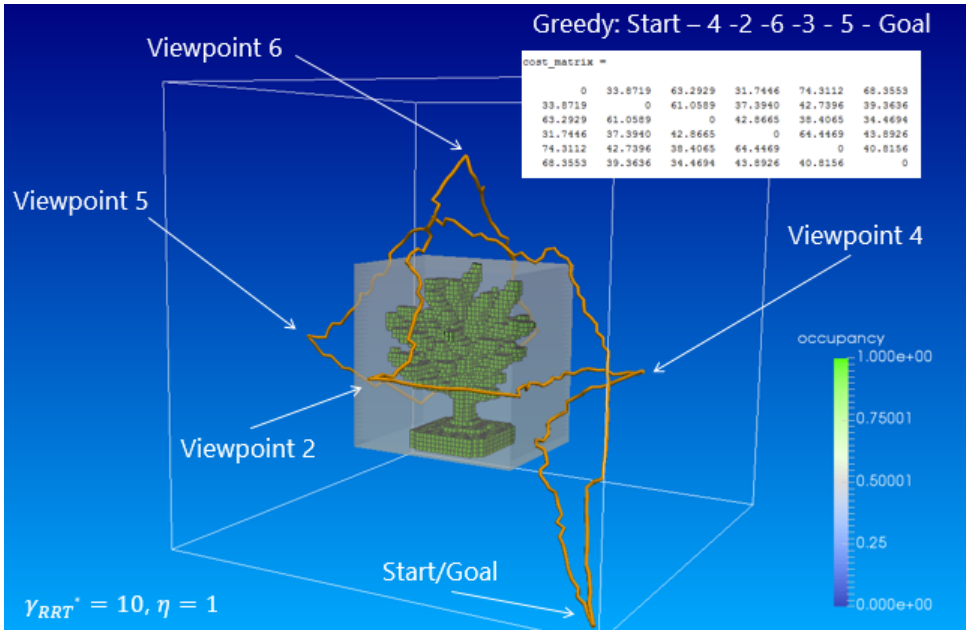

Figure 5.4: Object scanning route according greedy algorithm.

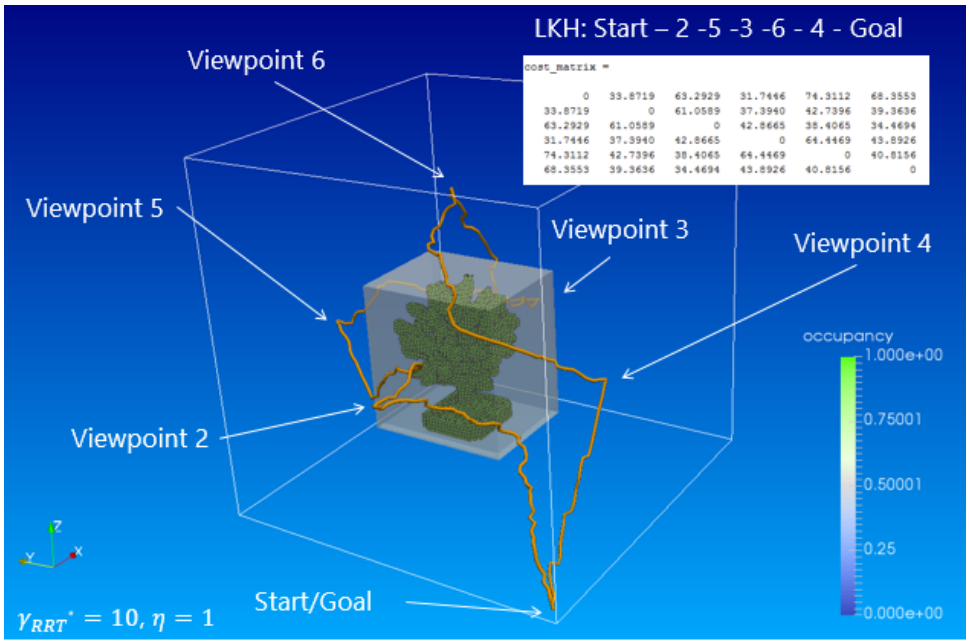

Figure 5.5: Object scanning route according the LKH algorithm.

Chapter 6

Summary

This chapter gives a brief conclusion of the results and an outlook.

6.1 Conclusion

The RRT* algorithm is proved to be highly suitable for finding paths between two points. In comparison with other methods, it is adaptable to high-dimensional spaces. However, to generate feasible path in large spaces and to get optimal results, long calculation time is required, which is a major drawback of this method. Otherwise, paths, like the one for object scanning (see Section 5.3), would result in a zig-zag-pattern. Although *Scubo* is able to handle such kind of trajectories, they are not pleasing in an efficiency matter.

6.2 Outlook

In a further step, the generated paths need to be smoothed in a post-processing step in order to remove the zig-zag-pattern. Furthermore, the configuration space can be expanded in order to produce a trajectory consisting of position and orientation information. An implementation in C++/ROS and the combination with a trajectory controller is required to test the path planner on the real system.

The path planner developed in this bachelor thesis requires information of the static map in order to produce feasible paths from a fixed starting position. However, this rarely the case in real world application. In addition to static, there are also dynamic obstacles. Therefore, an real-time RRT* algorithm in combination with SLAM must be developed in order to recognize moving obstacles and to locally replan the path. Additionally, an object identification algorithm must be produced in order to autonomously plan scanning routes.

Bibliography

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, "Autonomous mobile robots," *Massachusetts Institute of Technology*, 2004.

[3] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.

[4] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.