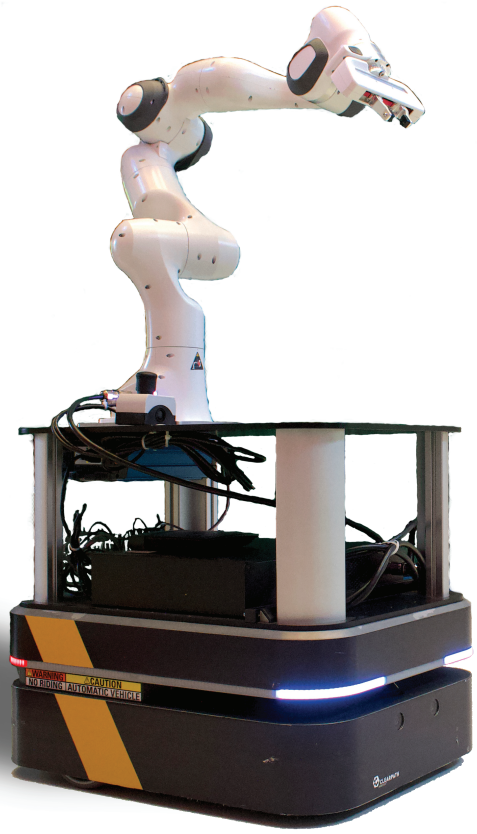
Local roadmap adaptation for mobile manipulators in incrementally changing environments

E.J. Heerkens

Master of Science Thesis



TU Delft Delft University of Technology

A.R THE AI FOR RETAIL LAB DELFT

Local roadmap adaptation for mobile manipulators in incrementally changing environments

by

E.J. Heerkens

to obtain the degree of Master of Science in Mechanical Engineering
at the Delft University of Technology,
to be defended on July 21, 2021 at 1:00 PM.

Student number:	4375610	
Project duration:	September 1, 2020 – July 21, 2021	
Thesis committee:	Dr. J. Alonso-Mora,	TU Delft, supervisor
	M. Spahn,	TU Delft, daily supervisor
	Dr.Ing. S. Grammatico,	TU Delft
	Prof.Dr.Ir. M. Wisse,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Supplementary data available, doi: 10.4121/14912766.v1

TU Delft

Abstract

Mobile manipulators will be deployed in supermarkets for a large variety of tasks, for instance, for restocking products. The operation time of mobile manipulators can be reduced by generating coupled trajectories for the base and the robot's arm. When planning for high Degree of Freedom (DOF) robots, such as a mobile manipulator, in an obstacle-cluttered environment, the graph construction for sampling-based planners is time-consuming. If changes in the environment occur, most sampling-based algorithms reconstruct the entire graph. In some dynamic environments, planning can be simplified by the assumption of an incrementally changing environment; this is a mostly static environment where slight changes occur that do not violate the connectivity of the free configuration space, indicating that a significant part of the graph remains valid.

The main contribution of this thesis is a new motion planning algorithm: the adaptive roadmap algorithm (ARM). ARM is a multi-query sampling-based motion planning algorithm that can locally adapt vertices and edges of the graph to account for incremental changes in the environment to allow faster planning than algorithms that reconstruct the entire graph. ARM generates a 3D grid to represent the workspace. The grid cells are marked as occupied or free based on the presence of obstacles in the environment. To determine what vertices and edges of the roadmap need to be updated due to a change in the occupancy of the 3D grid by an incremental change, ARM assigns the vertices and edges to the 3D grid cells. ARM performs this assignment based on the workspace representations of the vertices and edges of the roadmap by 3D bounding boxes surrounding robot configurations. If the occupancy of one or multiple grid cells is changed due to an obstacle, the algorithm resamples the vertices associated with the occupied grid cells and removes the edges associated with the occupied grid cells. Then, the updated roadmap is used for motion planning, and if additional changes occur, this roadmap update is repeated.

We carried out different experiments in simulation performing coupled motion planning for mobile manipulators. A simplified implementation of ARM, which enables the implementation in the Robot Operating System, reported a $35-40\%$ speedup of the planning time compared to the single-query algorithm rapidly-exploring random tree, which reconstructs the entire graph for every new query or change in the environment. The speedup the simplified implementation gained compared to existing planners will be magnified for the non-simplified ARM as the roadmap adaptation by ARM is 10% faster than by the simplified ARM. Further experiments demonstrated that the algorithm successfully adapted the roadmap for a real-world system, not merely in simulation. We conclude that local roadmap adaptation by our proposed algorithm allows faster planning than algorithms that reconstruct the entire graph for mobile manipulators in incrementally changing environments.

Nomenclature

Abbreviations

ARM	Adaptive roadmap algorithm
DOF	Degrees of freedom
ER	Elastic roadmap algorithm
EXOTica	Extensible Optimization Toolset
OMPL	Open Motion Planning Library
PRM	Probabilistic roadmap
PRM (MQ)	Probabilistic roadmap, conventional multi-query implementation
PRM (SQ)	Probabilistic roadmap, single-query implementation that clears the roadmap after every query
ROS	Robot Operating System
RRT	Rapidly-exploring random tree
sARM	Simplified adaptive roadmap algorithm

Symbols

(n_x, n_y, n_z)	Index of a 3D grid cell of ARM/sARM, where n_x, n_y, n_z are the indices in the $x-$, $y-$ and $z-$dimension in the grid.
Δq	Interpolation resolution for collision checking of edges
Δq_e	Interpolation resolution for assigning edges to grid cells in ARM
δ_k	Size of the 3D grid cells of ARM/sARM, where k denotes the dimension in $\mathcal{W}$: $k \in \{x, y, z\}$
$\mathbb{R}^N$	Euclidean space
$\mathbb{S}^N$	N-sphere
$\mathbf{q}$	Robot configuration
$\mathbf{q}_{\text{goal}}$	Desired configuration
$\mathbf{q}_{\text{inv}}$	Invalid configuration representing a vertex invalidated due to an incremental change
$\mathbf{q}_{\text{new}}$	Configuration replacing $\mathbf{q}_{\text{inv}}$ in the roadmap adaptation by ARM
$\mathbf{q}_{\text{start}}$	Initial configuration
$\mathcal{CO}$	Occupied configuration space
$\mathcal{C}$	Configuration space
$\mathcal{C}_{\text{free}}$	Free configuration space
$\mathcal{C}_{\text{sub}}$	Subset of the configuration space close by the vertex invalidated due to an incremental change, in which random samples are drawn to find a replacing vertex in ARM/sARM
$\mathcal{O}$	Obstacle

$\mathcal{W}$	Workspace
σ	Path
$bb_{\mathbf{q}}$	3D bounding box representing a robot configuration in $\mathcal{W}$
$bb_{\max_k}$	Maximum boundary of $bb_{\mathbf{q}}$ for every dimension $k \in \{x, y, z\}$
$bb_{\min_k}$	Minimum boundary of $bb_{\mathbf{q}}$ for every dimension $k \in \{x, y, z\}$
E	Set of edges
e_L	Edge length
$e_{L_{val}}$	Longest valid segment of an edge
$G(V, E)$	Graph that consists of a set of vertices V and edges E
IE_e	Set of 3D grid cell indices associated with an edge e determined based on the interpolation by Δq_e in ARM
IO	Set of occupied grid indices of ARM/sARM
$IQ_{\mathbf{q}}$	Set of 3D grid cell indices associated with the bounding box representing $\mathbf{q}$ in ARM
M	Mapping from 3D grid cells to associated vertices and edges
$R(V, E)$	Roadmap, which is a multi-query graph, that consists of a set of vertices V and edges E
$R_a(V, E)$	Roadmap adapted by ARM/sARM that consists of a set of vertices V and edges E
V	Set of vertices

Contents

List of Figures

List of Tables

1 Introduction

Robots are deployed for improving productivity in industrial applications by performing repetitive tasks. For retail stores, robots can reduce the workload of store employees, for example, by stocking shelves. The AI for Retail (AIR) Lab Delft [1] researches the use of robotics in stores and warehouse environments. The robot used for in-store proceedings is a mobile manipulator (see Figure 1.1), which consists of a moving platform with a robotic arm on top. Mobile manipulators are useful for manipulating objects over large distances [27]. Clearly, such robots should operate autonomously and fast. An important part of the autonomous behaviour of robots is navigating through the environment they operate in [32]. The planning problem is the problem of finding a sequence of valid configurations from an initial to the desired configuration while adhering to constraints, such as obstacles to be avoided and joint limits not to be exceeded. The planning problem is solved by global motion planning algorithms. Local motion planning algorithms are reactive approaches, which are used to deal with unforeseen situations, such as the appearance of obstacles, while executing the global plan [32]. The space a robot operates in is defined as the workspace.

1.1. Motivation

1.1.1. Coupled motion planning for a mobile manipulator

A challenging feature of planning for a mobile manipulator is the large number of degrees of freedom (DOFs) [35], in the case of the mobile manipulator in the AIRLab (Figure 1.1): 10 DOFs. In many applications, the planning for the base and arm is decoupled to simplify planning for mobile manipulators. This results in a sequenced motion, where locomotion and manipulation never occur at the same time [43]. However, in the interest of improving the efficiency in supermarkets, reducing the operation time to perform tasks by retail robots is desired.

The operation time of mobile manipulators can be reduced by generating coupled trajectories for the base and the robot's arm, which results in the robot behaving more human-like by moving the arm towards the desired configuration while moving through the workspace [54, 55]. Additionally, this planning approach increases flexibility, ensuring movement of the arm if a problem is infeasible when not considering the arm. For instance, the robot could fold its arm to avoid an obstacle hanging from the ceiling. This flexibility enables fulfilling more complex planning problems due to considering the robot as one instead of two separate parts. In the interest of reducing the operation time, coupled planning for all DOFs of the mobile manipulator is favorable over planning for the base and arm separately.

1.1.2. Motion planning for high-DOF robots

The configuration space is introduced to prevent keeping track of all points on the robot to ensure a collision-free path [36]. In the configuration space, the robot is viewed as a point, and the obstacles are mapped to this space. This simplifies planning for an arbitrarily shaped robot to planning for a point. A configuration is a specification of all DOFs of the robot. The configuration space contains all possible configurations of the robot. The dimension of the configuration space equals the number of DOFs of the robot. For algorithmic purposes, one often discretises the configuration space, which gives rise to a graph. This graph consists of vertices representing valid robot configurations and edges representing feasible paths (see Figure 1.2 for an example). The graph reduces the planning problem to a discrete planning problem and is used to query for a

solution [32]. In Chapter 2 we will introduce the configuration space and graphs more formally.

Figure 1.1: This figure presents the mobile manipulator robot from the AI for Retail (AIR) Lab Delft [1], which consists of a moving base and an arm on top. This robot consists of 10 degrees of freedom (DOF): 3 of the base and 7 of the arm.

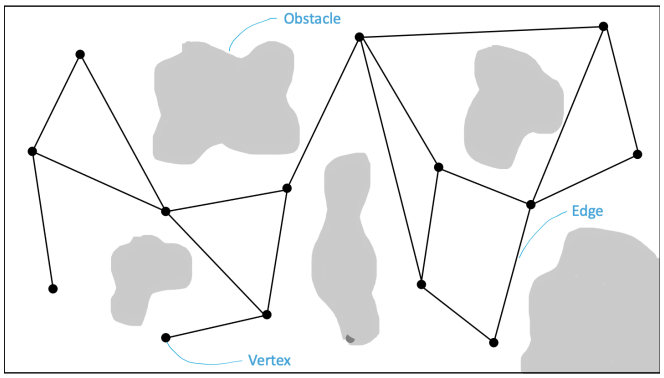


Figure 1.2: This figure visualises a graph in a 2D configuration space that is used to query for a solution and thereby reduces the planning problem to a discrete planning problem. A graph consists of vertices representing valid robot configurations and edges representing feasible paths. The obstacles are mapped to the configuration space.

To solve planning problems, different types of global motion planning algorithms exist. Based on the graph construction, two types of motion planners can be distinguished: combinatorial and sampling-based planners [32]. Combinatorial planners discretise the space without losing information because they explicitly represent all valid and invalid configurations, resulting in correctly reporting the existence or non-existence of a solution, which is defined as the completeness of a planner [29]. Due to the explicit representation of all configurations, the number of vertices of the graph in the configuration space grows exponentially with every added DOF. This implies that the planning time, which is the time it takes to find a path from an initial to a goal configuration, increases as well. In practice, the computational cost for planning for high DOF robots, such as a mobile manipulator, is substantial when using combinatorial planners[32]. Sampling-based planners avoid explicit construction of all configurations and, with this, deliberately trade completeness for a shorter planning time by sampling the configuration space [22]. When planning in a high dimensional configuration space, the graph of sampling-based planners does not grow exponentially with every added DOF, ensuring the planning problems can be solved within practical time bounds [21]. Therefore, sampling-based planners are often more practical than combinatorial planners when planning for high-DOF robots such as mobile manipulators.

Single-query algorithms construct a graph for a specific query, which is a specification of a start and goal configuration. Multi-query algorithms allow planning for many queries in the same space by using a single graph. A roadmap is a multi-query graph. Roadmaps need to cover the entire planning space to allow planning for different queries, resulting in more time spend on graph construction than single-query planners [32]. When planning for a high-DOF robot in an obstacle cluttered environment, such as a supermarket, the graph construction is time-consuming due to the excessive sampling necessary to construct a graph because more samples are invalid than in an environment with fewer obstacles [23]. For planning problems that require an expensive graph construction, reusing the graph is desirable. However, a common drawback of multi-query sampling-based algorithms is that if changes in the workspace occur, the time-consuming graph construction must be repeated. This time-consuming reconstruction makes multi-query planners unsuitable for dynamic environments; therefore, single-query algorithms are widely used in these environments [48]. The robot remains stationary while a single-query algorithm generates a new graph for every query. Resampling is the repeated sampling for graph construction by probing the configuration space for every environmental change or new query. This resampling is time-consuming because the graph construction requires excessive sampling as was previously explained.

1.1.3. Incrementally changing environment

To deploy a robot in a supermarket, we make assumptions about the environment to simplify the planning problem; as in a dynamic environment, there are no restrictions concerning the magnitude of changes.

Firstly, we assume that the free configuration space is connected, which indicates that any collision-free configuration can reach another collision-free configuration [56]. For motion planning, this indicates that always a path exists between two collision-free configurations. Then, we simplify the dynamic supermarket environment: although a supermarket is a dynamic environment, we assume a significant part of the environment remains unchanged because static shelves occupy it. In the corridors between the shelves, obstacles such as customers, baskets and boxes appear. We assume that these obstacles do not violate the connectivity of the free configuration space. We call an environment that is mostly static, where slight changes occur, that do not violate the connectivity of the free configuration space an incrementally changing environment. For a graph generated by a sampling-based algorithm, incremental changes in the environment imply that a significant part of the graph remains valid between two queries, indicating that a significant part of the vertices and edges remain collision-free.

The incrementally changing environment assumption allows various algorithms proposed in earlier work to locally adapt a multi-query graph, which prevents time-consuming resampling for an incremental change or new query by reusing the graph [14, 24–26, 34, 41, 51, 58–60]. However, these algorithms are not suitable for planning for mobile manipulators in supermarket environments for various reasons. Some algorithms plan in a 2D workspace for a low-DOF robot [14, 24–26, 41], which is unsuitable for a mobile manipulator that simultaneously plans for the base and arm. Other algorithms are computationally expensive [34, 51, 58–60], which may cause the robot to remain stationary during the roadmap adaptation. These algorithms and their limitations will be discussed more extensively in Chapter 3.

1.2. Research question

When performing coupled motion planning for the base and the arm of a mobile manipulator in a supermarket environment, time-consuming resampling for graph reconstruction due to incremental changes must be avoided to reduce the planning time. Algorithms proposed in earlier work locally adapt a roadmap, avoiding graph reconstruction. However, these algorithms are not yet applicable for mobile manipulators in supermarkets. Therefore, the research question for this MSc Thesis is:

How can local roadmap adaptation allow fast planning for mobile manipulators assuming the environment changes incrementally?

Here "fast" indicates a speedup compared to algorithms that reconstruct the graph due to a change in the environment.

1.3. Contribution

The main contribution of this thesis is the introduction of a new motion planning algorithm: the adaptive roadmap algorithm (ARM) that allows fast planning for mobile manipulators in incrementally changing environments by locally adapting the roadmap. ARM does not have to reconstruct its roadmap due to incremental changes or a new query, which decreases the operation time compared to algorithms that reconstruct their graph for every query or environmental change, and remain stationary while doing so. Reconstructing the graph is time-consuming for high-DOF robots in obstacle-cluttered environments, which makes ARM particularly convenient for coupled motion planning for mobile manipulators in obstacle-cluttered environments, such as supermarkets.

While there are some limitations, which we discuss in Chapter 6, our algorithm achieves fast planning for mobile manipulators in incrementally changing environments.

1.4. Requirements

We set requirements for an algorithm that locally adapts the roadmap in the context of mobile manipulators in incrementally changing environments. The algorithm must:

R1 reuse its roadmap for multiple queries.

- Reusing the roadmap prevents the time-consuming reconstruction the roadmap.

R2 locally adapt the roadmap to changes; these changes can be due to appearing or disappearing obstacles.

- Locally adapting the roadmap to changes ensures all vertices and edges remain valid. We additionally aim to deal with disappearing obstacles to prevent parts of the environment where once an obstacle was present from becoming inaccessible.

R3 be applicable for a high-DOF robot (high dimensional configuration space).

- Being applicable for high-DOF robots enables planning for a mobile manipulator.

R4 be applicable in a 3D workspace.

- Being applicable in a 3D workspace prevents collision of the entire mobile manipulator, including the protruding arm, with obstacles.

R5 adapt the roadmap faster than replanning.

- Adapting the roadmap fast ensures retaining the advantage of reusing the roadmap over reconstructing a graph for every query or change in the environment.

1.5. Thesis structure

This thesis starts with some preliminaries in Chapter 2. Then, we present related work proposing algorithms that locally adapt a roadmap in Chapter 3, followed by the methods describing our proposed algorithm in Chapter 4. Then, we describe experiments and present the results in Chapter 5. Finally, we discuss conclusions and proposals for future work in Chapter 6.

2

Preliminaries

This chapter describes the background and formal definitions to prepare for the further contents of this thesis. We discuss the motion planning problem in Section 2.1, motion planning properties in Section 2.2, graph theory in Section 2.3, sampling-based motion planning in Section 2.4 and two widely used sampling-based motion planners in Section 2.5 and Section 2.6. Readers familiar with these topics may skip this chapter.

2.1. Motion planning problem

The motion planning problem is defined as finding a sequence of valid configurations from an initial to the desired configuration while adhering to some set of constraints, such as avoiding obstacles and not exceeding joint limits [32]. A configuration $\mathbf{q}$ is a specification of all degrees of freedom (DOFs) of the robot. We denote the initial configuration by $\mathbf{q}_{\text{start}}$ and the desired configuration by $\mathbf{q}_{\text{goal}}$. We formulate the environment by:

- The robot is moving in the workspace $\mathcal{W}$, which is a Euclidean space $\mathbb{R}^N$ (with $N = 2,3$).
- The obstacles represented by rigid objects in $\mathcal{W}$ are denoted by $\mathcal{O}_i$, or equivalently $\mathcal{WO}_i$, for $i = 1,...,n_o$, where n_o is the number of obstacles.

Then, we formulate the motion planning problem as: given $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$, generate a path σ, which consists of a continuous sequence of robot configurations from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$ while adhering to the constraints. We want to keep track of when the robot avoids obstacles. For this purpose, we introduce the configuration space.

2.1.1. Configuration space

The configuration space is introduced to prevent keeping track of all points on the robot to ensure a collision-free path [36]. This space allows a more formal definition of the motion planning problem. In $\mathcal{C}$ we view the robot as a point and map the obstacles to this space to simplify planning for an arbitrarily shaped robot to planning for a point. All possible configurations of the robot are in $\mathcal{C}$, leading to the addition of a dimension to $\mathcal{C}$ for every DOF of the robot. We model $\mathcal{C}$ as the Cartesian product of all DOFs

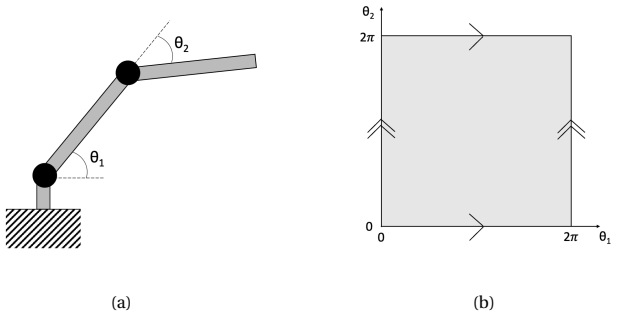
$$\mathcal{C} = \mathcal{C}_1 \times ... \times \mathcal{C}_{N_{DOFs}} \tag{2.1}$$

with N_{DOFs} the number of DOFs of the robot. We describe the DOFs by bounded intervals on $\mathbb{R}$, and circles on $\mathbb{S}$. We present an example of a 2-DOF manipulator, with a 2D $\mathcal{C}$ represented by the parameters: θ_1 and θ_2 (see Figure 2.1a), which are angular coordinates described by $\mathbb{S}$, assuming the joints have no limits. If the joints would have had limits, we describe the angles by a segment of the circle, which is a finite interval and would have been described by $\mathbb{R}$ [32]. We determine $\mathcal{C}$ of this robot as the Cartesian product $\mathbb{S} \times \mathbb{S}$. The DOFs can be numerically presented in $\mathbb{R}^2$ by $[0,2\pi) \times [0,2\pi)$ (see Figure 2.1b).

2.1.2. Obstacle representation

We map the obstacles to $\mathcal{C}$ to describe in-collision configurations. Assume that we represent the robot by a rigid object, $\mathcal{A} \subset \mathcal{W}$, we can find the in-collision configurations by:

$$\mathcal{CO}_i = \{q \in \mathcal{C} | \mathcal{A}(\mathbf{q}) \cap \mathcal{O}_i \neq \emptyset\} \tag{2.2}$$



(a) (b)

Figure 2.1: These figures illustrate (a) the workspace $\mathcal{W}$ and (b) the numerical representation of the configuration space $\mathcal{C}$ of a manipulator with 2 degrees of freedom (DOFs) described by two angles θ_1 and θ_2 which are assumed to have no limits.

where $\mathcal{A}(\mathbf{q})$ denotes the subset of $\mathcal{W}$ occupied by a robot configuration $\mathbf{q}$. If the robot consists of m bodies, we find $\mathcal{CO}_i$ in Equation (2.2) for $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_m\}$. The configuration space obstacle region is the union of $\mathcal{CO}_i$:

$$\mathcal{CO} = \bigcup_{i=1}^{n_o} \mathcal{CO}_i. \tag{2.3}$$

The free configuration space can be determined from $\mathcal{C}$ and $\mathcal{CO}$:

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{CO}. \tag{2.4}$$

2.1.3. Path

We can now define the motion planning problem as finding a path from a start configuration $\mathbf{q}_{\text{start}}$ to a goal configuration $\mathbf{q}_{\text{goal}}$ in $\mathcal{C}_{\text{free}}$. A valid solution to the motion planning problem is the continuous function σ mapping path parameter τ, usually in $[0,1]$, to a curve in $\mathcal{C}_{\text{free}}$, that is, a continuous map

$$\sigma : [0,1] \rightarrow \mathcal{C}_{\text{free}} \tag{2.5}$$

with:

$$\sigma(0) = \mathbf{q}_{\text{start}},\ \sigma(1) = \mathbf{q}_{\text{goal}} \text{ and } \sigma(\tau) \in \mathcal{C}_{\text{free}} \quad \forall \tau \in [0,1].$$

2.1.4. Planning with differential constraints

Besides the geometrical constraints induced by the obstacles and the joint limits, there may be additional constraints on the motion, such as differential constraints, which restrict the robot's motion based on the time derivative of the robot configurations. If the robot configuration depends on the path taken to reach it, the differential constraints are nonintegrable. These constraints are also called nonholonomic constraints.

Considering the wheels underneath the robot is an example of a nonholonomic constraint. The structure of the wheels prevents the robot from instantaneously moving sideways and rotating in place. This gives rise to a nonholonomic constraint characterised by the minimum turning radius of a robot. To adhere to this constraint in motion planning, we introduce Dubins curves [12] and Reeds-Shepp curves [44] to describe a path segment in the x-y plane by composing it into a sequence of motions, as describing them by straight lines is impossible. For Dubins curves, these motions consist of: straight, right turn, and left turn. For Reeds-Shepp curves, these motions are additionally available in reverse. A straight line describes a straight motion and an arc of a circle of the minimum turning radius describes the turns (see Figure 2.2). Dubins curves consist of three path segments and Reeds-Shepp curves of three to five path segments.

2.2. Motion planning properties

Several properties to describe motion planning algorithms are defined in literature:

- The guarantee of correctly reporting the existence or non-existence of a solution is defined as completeness. An algorithm is incomplete if no solution may be found, even if one does exists [29]. Additionally, if $\mathcal{C}$ is represented by a grid, an algorithm is resolution complete if a valid path is found, if one exists,

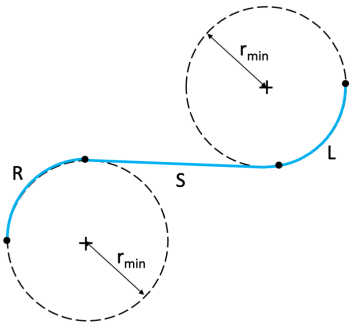


Figure 2.2: This figure illustrates Dubins and Reeds-Shepp segments used for describing path segments of robots with differential constraints induced by wheels underneath a robot. These segments consist of right turns (R), left turns (L) and straight lines (S). The turns are arcs of a circle with the minimum turning radius (r_{min}). For Reeds-Shepp curves these motions are additionally available in reverse.

at a certain level of resolution of the grid. A motion planning algorithm is probabilistic complete if the probability that the algorithm finds a solution goes to one as the running time goes to infinity; however, it does not return that no solution exists if this is the case [30].

- The optimality is defined as finding an optimal path for a given cost function in finite time. An algorithm is asymptotically optimal if it returns a sequence of solutions converging to an optimal solution [7]. An algorithm is non-optimal if no guarantees exist that the solution is optimal.

- Anytime is defined as providing a solution when the algorithm is terminated before it ends. However, the longer the algorithm is running, the better the solution quality [11].

- The time it takes to find a path from an initial to a goal configuration is the planning time of an algorithm.

- The number of operations required to solve a computational problem is defined as the complexity. The complexity is described by the resources required, such as time and storage. Usually, one studies the worst-case time complexity because proving lower bounds is much more difficult. The upper bound on the amount of time or storage required to solve a problem by an algorithm is described by the big O notation [52].

2.3. Graph theory

Graph theory is the basis for sampling-based motion planning algorithms that are widely used to solve motion planning problems. Graph theory is the study of graphs in mathematics used to model relations between objects. In this section, we introduce essential concepts of graph theory.

2.3.1. Graph

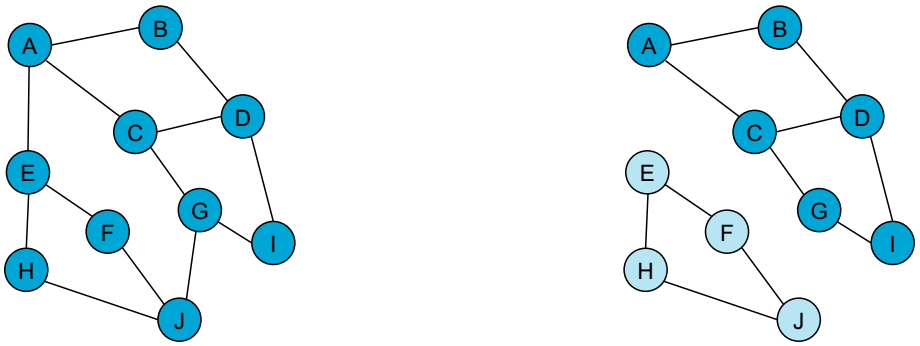
A graph ($G = (V, E)$) is a set of of vertices (V) and edges (E), where each edge is a connection between vertices (see Figure 2.3a). The set of vertices is:

$$V = \{v_1, v_2, ..., v_{n_v}\} \quad ,$$

where n_v is the number of vertices of the graph. The set of edges is:

$$E \subseteq \{(v_i, v_j) | v_i, v_j \in V \ \& \ v_i \neq v_j\}$$

with vertices v_i and v_j the source and target vertex of the edge (v_i, v_j), respectively.



(a) (b)

Figure 2.3: These figures illustrate (a) a connected graph consisting of vertices (A-J) and edges, which are the connections between the vertices and (b) a graph that consists of two connected components that contain a subset of the vertices.

2.3.2. Connectivity

A graph is connected if a path exists between any pair of vertices in the graph (Figure 2.3a). Connected components are subsets of vertices, $V_c \subseteq V$, for $c \in \{1, ..., n_c\}$, with n_c the number of connected components, which comprise at maximum of all vertices in V (see Figure 2.3b). If a graph consists of more than one connected component, the graph is disconnected. Two vertices are connected if a path, a sequence of vertices connected by edges, exists between them. Vertices in the same connected component are connected.

2.4. Sampling-based motion planning

Sampling-based algorithms are widely used to solve motion planning problems. In this section, we introduce the essential concepts.

2.4.1. Description

Sampling-based algorithms avoid explicit construction of $\mathcal{CO}$ by sampling to probe $\mathcal{C}$ [33]. A collision checking module facilitates the probing, which returns whether a sample is valid or not. By avoiding the exact representation of $\mathcal{C}$, sampling-based planners reduce the time complexity of solving planning problems. However, this is generally done at the cost of completeness, resulting in sampling-based planners being probabilistic complete at best. Sampling-based planners capture the connectivity of $\mathcal{C}_{\text{free}}$ by sampling to construct a graph. For sampling-based motion planning, the vertices are collision-free robot configurations: $V \subseteq \mathcal{C}_{\text{free}}$. There will only be an edge if there exists a path in $\mathcal{C}_{\text{free}}$ between two vertices.

Figure 2.4 shows a simple graph in a 2D $\mathcal{C}$ ($\mathcal{C} = \mathbb{R}^2$). Note that in this figure, the obstacles are visualised; however, a sampling-based algorithm is not aware of these because it does not explicitly reconstruct $\mathcal{CO}$. A collision checking module returns whether a vertex or edge is valid, which ensures the graph is in $\mathcal{C}_{\text{free}}$. When $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are in the same connected component, we can extract a path. Different sampling-based algorithms implement different approaches for constructing the graph and finding a path. We describe the most commonly used methods in this section.

2.4.2. Metrics

Sampling-based algorithms require a function for measuring the distance between two points in $\mathcal{C}$. To determine metrics for $\mathcal{C}$, we introduce the notion of a metric space (X, ρ), which is a topological space X with a distance function $\rho : X \times X \rightarrow \mathbb{R}$. We denote the distance by $\rho(a, b)$, where $a, b \in X$. A metric for DOFs described in $\mathbb{R}^N$ is the L_p metric:

$$\rho_p(x, x') = \left(\sum_{i=1}^{N} \left| x_i - x'_i \right|^p \right)^{\frac{1}{p}} \quad \forall p \geq 1. \tag{2.6}$$

The most common case is L_2, which is known as the Euclidean distance in $\mathbb{R}^N$. The distance for DOFs described in $\mathbb{S}$ is better described by the distance along the circle, as we define $\rho(x, x')$ for $x, x' \in [0, 2\pi]$ by:

$$\rho(x, x') = min\left\{ \left| x - x' \right|, 2\pi - \left| x - x' \right| \right\}. \tag{2.7}$$

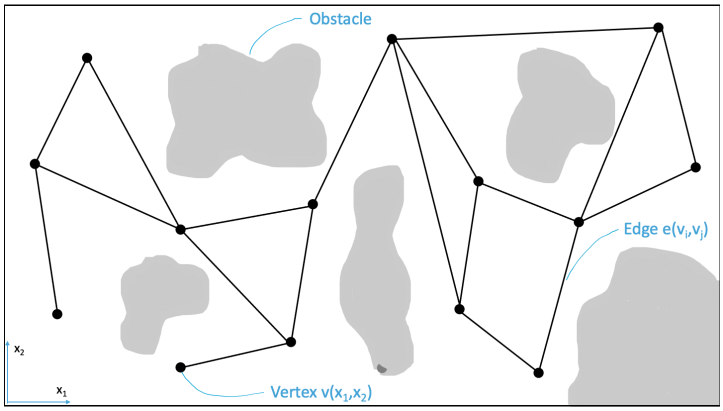


Figure 2.4: This figure illustrates a graph $G(V,E)$ in a 2D configuration space ($\mathcal{C} = \mathbb{R}^2$), constructed by a sampling-based motion planner. A vertex is a collision-free possible configuration of the robot, in this example described by the degrees of freedom (DOFs) x_1 and x_2, and an edge is a connection between two vertices, in this example between v_i and v_j. Note that a sampling-based motion planner is not aware of the obstacles as they are illustrated because it does not explicitly reconstruct them.

Nonholonomic robots need a more complicated metric for their base coordinates than L_2, because two points can not be connected by a straight line. The path is described by Dubins or Reeds-Shepp curves. For these robots, the metrics of Equation (2.6) and Equation (2.7) are combined to find the length of a Dubins or Reeds-Shepp curve for the base trajectory.

2.4.3. Graph construction

The graph construction consists of generating vertices, edges and performing collision checks to check whether the attempted vertices and edges are in $\mathcal{C}_{\text{free}}$.

- **Generating vertices** that comprise the graph is performed by sampling in $\mathcal{C}$. The performance of the planning algorithm is dependent on the order of the sampling since the graph construction is often terminated early [32]. Denseness means that the vertices come close to any configuration in $\mathcal{C}$. A set of vertices (V) of the same topological space as $\mathcal{C}$ is dense in $\mathcal{C}$ if the closure, which consists of all points including the limit points of a set, of V is equal to $\mathcal{C}$. Achieving denseness is impossible if $\mathcal{C}$ is infinite. Randomly sampling a uniform sequence in $\mathcal{C}$ yields a probably dense set V, which means that the probability that V is dense approaches one as more samples are generated [32]. Suppose $\mathcal{C} = [0,1]$ and k samples are independently at random drawn from $\mathcal{C}$, the probability that none of the samples falls into the interval, $I \subset [0,1]$, of length e is $(1-e)^k$, which converges to zero if k goes to infinity. In other words: for any nonzero-length interval in $\mathcal{C}$, the probability that it contains no samples converges to zero as k goes to infinity. If k is not very large, no guarantee of coverage of $\mathcal{C}$ by vertices exists. If $\mathcal{C}$ is given by $X_1 \times X_2$ and the uniform random samples x_1 and x_2 are drawn from X_1 and X_2, respectively, the sample (x_1, x_2) is a uniform random sample of $\mathcal{C}$. The generated sample is added to the graph as a vertex if the collision checking module returns it is in $\mathcal{C}_{\text{free}}$.

- **Generating edges** that connect vertices is necessary to enable the robot to move from one vertex to another. The most common approach for determining what vertices to attempt to connect to is the k-Nearest Neighbour method, which attempts to connect to the k nearest neighbours. Here, k is a predetermined value. The nearest neighbours are determined using the metrics corresponding to the shapes ($\mathbb{R}$ or $\mathbb{S}$) of the dimensions of $\mathcal{C}$ [32] (Section 2.4.2). The edge is added to the graph if the collision checking module returns it is in $\mathcal{C}_{\text{free}}$.

- **Collision checking** is considered to be a black box by the motion planning algorithm, which ensures independence of geometrical models of the robot and environment. The input of the collision checking module is a configuration, $\mathbf{q}$, in $\mathcal{C}$. To check whether an edge is collision-free, we define an edge as a path segment, $e : [0,1] \rightarrow \mathcal{C}$, and find whether $e([0,1]) \subset \mathcal{C}_{\text{free}}$. The most common approach is to interpolate the interval $[0,1]$ of the edge. The corresponding collision checking module is called for the interpolated configurations [32]. The interpolation resolution, Δq is related to the edge length e_L:

$$e_L = e(1) - e(0) \tag{2.8}$$

and the longest valid segment of an edge $e_{L_{val}}$:

$$e_{L_{val}} = \Delta q \cdot e_L. \tag{2.9}$$

Commonly, a fixed Δq in $\mathcal{C}$ is determined [32], which has to be small enough to prevent missing obstacles. However, it should not be smaller than necessary to prevent checking a large number of configurations for collision. Therefore, we must determine $e_{L_{val}}$ to find a suitable value of Δq. A recursive binary strategy ensures the fastest collision checking [15], which starts by checking the configuration in the middle of the edge, whereafter it recurses on the two halves. If the edge is collision-free, this does not lead to a difference in time spend on collision checking; however, if an edge is in-collision, it often saves time compared to checking the edge for collision from 0 to 1. The computational cost of sampling-based motion planners is mainly driven by the time necessary for collision checking [39].

2.4.4. Finding a path

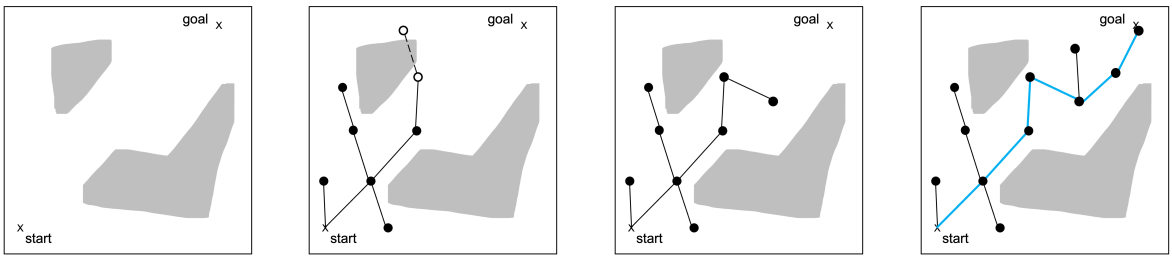
By constructing a graph, we simplify the problem to a discrete planning problem, and we can extract a path when $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are in the same connected component of the graph. The path can be extracted by, for example, performing a graph search on the constructed graph or by returning the path once these configurations are connected.

Single-query algorithms plan for one specific $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$. If a new $\mathbf{q}_{\text{start}}$ or $\mathbf{q}_{\text{goal}}$ is provided, a new graph is constructed to obtain a feasible path [32]. A widely used single-query algorithm is rapidly-exploring random tree (RRT), which will be more extensively introduced later in this chapter.

Multi-query algorithms plan for multiple $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ from one single graph. If a new $\mathbf{q}_{\text{start}}$ or $\mathbf{q}_{\text{goal}}$ are provided, the same vertices and edges are used to find a new path [21]. These vertices and edges form a roadmap. Because the roadmap is not specific for a certain query, it must capture the connectivity of $\mathcal{C}_{\text{free}}$ to ensure that the algorithm can answer future queries. A widely used multi-query algorithm is probabilistic roadmap (PRM), which will be more extensively introduced later in this chapter.

2.5. Rapidly-exploring random tree

A widely used single-query algorithm is rapidly-exploring random tree (RRT) [31], to which literature proposed various extensions. RRT builds a collision-free tree by incrementally sampling in $\mathcal{C}$ from $\mathbf{q}_{\text{start}}$, without having to set the number of samples a priori. As each sample is drawn, it is attempted to connect this to the nearest vertex in the existing tree. If this is successful, the sample is included in the graph as a vertex. If a vertex is within a predefined distance to $\mathbf{q}_{\text{goal}}$, the path is found, and the algorithm is terminated. The algorithm queries for a solution by going back in the tree to the parent of the vertex from $\mathbf{q}_{\text{goal}}$ to $\mathbf{q}_{\text{start}}$. RRT allows specification of a bias towards specific areas, such as the goal area. The goal bias is quantified by the probability that $\mathbf{q}_{\text{goal}}$ instead of a random sample is selected during tree expansion. Figure 2.5 presents a visualisation of RRT in a simple environment containing two obstacles.



(a) (b) (c) (d)

Figure 2.5: These figures present an illustration of the sampling-based rapidly-exploring random tree (RRT) algorithm finding a path in an environment containing two obstacles. (a) A tree is grown from the start ($\mathbf{q}_{\text{start}}$) to the goal configuration ($\mathbf{q}_{\text{goal}}$), by sampling in the configuration space $\mathcal{C}$ and connected as a new vertex to the tree if it is collision-free, (b) if sample is generated that is not collision-free connectable to the nearest neighbour in the tree, the sample and edge are not added, (c) shows the exploring of the free space and (d) the planner terminates when a sample is close to $\mathbf{q}_{\text{goal}}$, the path is returned (blue).

We present the graph construction of the RRT algorithm in Algorithm 1. Firstly, empty lists are created for the vertices (V) and edges (E) of the graph, and it is determined what the maximum number of expansions

the algorithm will perform (n) is. Then, as long as the number of expansions is less than n, new configurations, $\mathbf{q}$, are randomly generated in $\mathcal{C}$ (line 4), and if a valid connection can be made to the closest neighbour in V (line 5-6), it is stored as a vertex in V (line 7) and the connection is stored in E (line 8). This is repeated until the number of expansions is equal to n or if $\mathbf{q}_{\text{goal}}$ is reached. The output of this algorithm is the constructed graph $G(V,E)$. The graph construction can be terminated early, before the number of expansions reaches n, dependent on the implementation of the algorithm, for example, if the path is found.

Algorithm 1 RRT graph construction algorithm [8]

Input: n: maximum number of expansions
Output: $G(V,E)$: Graph

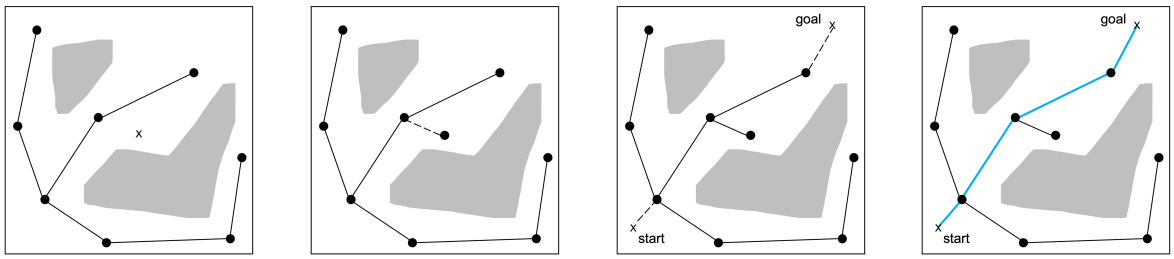
1: $V \leftarrow \mathbf{q}_{\text{start}}$
2: $E \leftarrow \emptyset$
3: **for** $i < n$ **do**
4: $\quad \mathbf{q} \leftarrow$ a random configuration in $\mathcal{C}$
5: $\quad \mathbf{q}' \leftarrow$ closest neighbour to $\mathbf{q}$ in V
6: $\quad$ **if** $f(\mathbf{q},\mathbf{q}') \in \mathcal{C}_{\text{free}}$ **then**
7: $\quad\quad V \leftarrow V \cup \mathbf{q}$
8: $\quad\quad E \leftarrow E \cup \{(\mathbf{q},\mathbf{q}')\}$
9: $\quad$ **end if**
10: **end for**
11: // Check if $\mathbf{q}_{\text{goal}}$ is reached
12: **return** $G(V,E)$

2.6. Probabilistic roadmap

A widely-used multi-query algorithm is the probabilistic roadmap (PRM) [23] algorithm, to which literature proposed various extensions. Conventional PRM consists of two steps:

1. Construct roadmap.
2. Query for a solution.

The roadmap construction is performed by generating uniform random samples in $\mathcal{C}$ and if they are in $\mathcal{C}_{\text{free}}$, added as vertices to the roadmap. After that, connections between the vertices are created and if they are in $\mathcal{C}_{\text{free}}$, added as edges. Then, as part of the second step, $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are connected to the roadmap by searching for edges in $\mathcal{C}_{\text{free}}$ connecting to neighbours in the roadmap, and lastly, a graph search algorithm is performed to find the path. If a new query is presented, the new $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are added to the roadmap and, if they are both connected to the graph, a path can be found, and if they are not, new random uniform samples are generated until they are. Figure 2.6 presents this algorithm in an environment with two obstacles.



(a) (b) (c) (d)

Figure 2.6: These figures present an illustration of the sampling-based probabilistic roadmap (PRM) algorithm finding a path in an environment containing two obstacles. (a) Random samples in the configuration space $\mathcal{C}$ are generated (X) and (b) connected to the roadmap by a local planner if the connection is collision-free, then, (c) the start ($\mathbf{q}_{\text{start}}$) and goal configuration ($\mathbf{q}_{\text{goal}}$) are connected to the roadmap and lastly, (d) a graph search algorithm finds a path from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$ (blue).

We present the basic roadmap construction of PRM in the pseudocode Algorithm 2 [8]. Firstly, the maximum number of vertices the roadmap will consist of (n) and the value of k for the k-nearest neighbour search,

to know what neighbours a vertex should connect to, are set. Then, empty lists are created for the vertices (V) and edges (E) of the roadmap (line 1-2). As long as the number of vertices in V is less than n, new configurations, $\mathbf{q}$, are randomly generated in the space in $\mathcal{C}$ (line 5), and if a configuration is in $\mathcal{C}_{\text{free}}$, it is stored as a vertex in V (line 7). This repeats until the number of vertices in V is equal to n or the graph construction is terminated. After that, for all vertices in V the k nearest neighbours, which are not already connected, it is searched whether a connection can be made between the two vertices ($f(\mathbf{q},\mathbf{q}')$) (line 10-12). If this is the case, the connection is added to the edges of the graph (E) (line 13). The output of this algorithm is the constructed roadmap $R(V,E)$. The graph construction can be terminated early, before the roadmap consists of n vertices, dependent on the implementation of the algorithm. For example, if the path is found, if these configurations are added before completing the roadmap construction instead of successively as described in the two steps above.

Algorithm 2 PRM roadmap construction algorithm [8]

Input: n: number of vertices to put in the roadmap
k: number of closest neighbours to examine for configuration
Output: $R(V,E)$: Roadmap

1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: **while** $|V| < n$ **do**
4: **repeat**
5: $\mathbf{q} \leftarrow$ a random configuration in $\mathcal{C}$
6: **until** $\mathbf{q} \in \mathcal{C}_{\text{free}}$
7: $V \leftarrow V \cup \mathbf{q}$
8: **end while**
9: **for all** $\mathbf{q} \in V$ **do**
10: $N_{\mathbf{q}} \leftarrow$ the k closest neigbours of $\mathbf{q}$ chosen from V according to a distance metric
11: **for all** $\mathbf{q}' \in N_{\mathbf{q}}$ **do**
12: **if** $(\mathbf{q},\mathbf{q}') \notin E$ **and** $f(\mathbf{q},\mathbf{q}') \in \mathcal{C}_{\text{free}}$ **then**
13: $E \leftarrow E \cup (\mathbf{q},\mathbf{q}')$
14: **end if**
15: **end for**
16: **end for**
17: **return** $R(V,E)$

3

Related Work

Algorithms that locally adapt a roadmap to an incremental change are a promising solution to deal with changes while keeping the computation time low. We can achieve a critical speed-up by eliminating the reconstruction of the graph. The adjustments in the roadmap may be necessary due to either appearing or disappearing obstacles.

In this chapter, we discuss algorithms that locally adapt a roadmap proposed in earlier work. Table 3.1 at the end of this chapter presents the main properties, test conditions, methods of roadmap adjustment, and to which extent they fulfil the requirements of Section 1.4. The eight discussed algorithms decide in what area the roadmap needs to be changed and adapt it. All algorithms in this chapter are applicable under the assumption of the incrementally changing environment. This assumption makes it possible to change the roadmap locally since the environment remains mostly unchanged. These algorithms would not be applicable without this assumption. All algorithms are extensions to PRM, introduced in Section 2.6. The roadmap adaptation occurs in all algorithms after the roadmap construction of PRM and then use the adapted roadmap $R_a(V,E)$ to query for a solution. We divide the algorithms into two subcategories: methods that adapt edges and that adapt vertices. We discuss these algorithms in the items in the corresponding section.

3.1. Roadmap adaptation based on edge adaptation

- An algorithm that deforms the edges to change the roadmap is the reactive deforming roadmaps (RDR) algorithm [14]. Instead of edges connecting the vertices, it consists of deformable links to make the roadmap adjust to the movement of obstacles. A deformable link is the representation of an edge by particles. Instead of removing the links that are in $\mathcal{CO}$, the links are deformed based on Newtonian Physics and Hooke's Law resulting in no loss of connectivity information. If it is not possible to deform the link to prevent collision, or if this results in an increase in the path length above a predefined threshold, the link is removed. Besides, this algorithm instantiates the movement of the vertices if this is necessary. In Figure 3.1 the deformation and movement of the edges as a response to a moving obstacle is visualised. As the obstacle moves, it may be necessary to add vertices or links to capture the connectivity. The algorithm keeps track of the removed links for testing whether these are valid at a later point during the planning, creates new edges between two vertices and samples in unexplored regions. Therefore, RDR can deal with disappearing obstacles. Experiments in a 2D environment with a 3-DOF mobile base robot and 14 similar 3-DOF mobile base robots as dynamic obstacles showed that the roadmap is repaired after the invalidation of the previously planned roadmap by dynamic obstacles. However, this algorithm is a proof-of-concept implementation without being optimized, and thus the performance is insufficient for implementation. This algorithm was created for a 2D workspace and a low-DOF robot, and therefore, not suitable for planning for a mobile manipulator.

- The replanning with path deformation algorithm combines path deformation with replanning to adjust the roadmap [60]. This algorithm deforms an edge or reconstructs the roadmap if an edge is in $\mathcal{CO}$ and can find shortcuts. A shortcut is an edge connecting two vertices directly, which were previously connected by two edges with another vertex in between by moving the vertex $\mathbf{q}_i$ in Figure 3.2 toward the interpolated point between the vertices $\mathbf{q}_{i-1}$ and $\mathbf{q}_{i+1}$ to $\mathbf{q}_0$. Due to triangular inequality, this shortcut

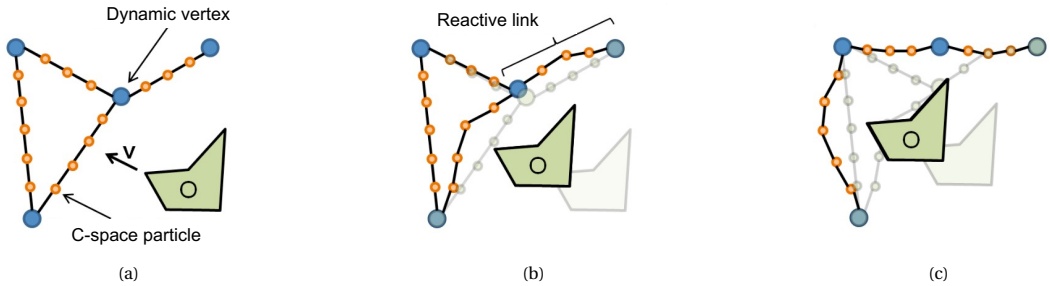


Figure 3.1: These figures present the deformation of the edges proposed in the reactive deforming roadmaps algorithm (RDR) [14]. RDR represents its roadmap as (a) dynamic vertices (blue) and reactive links as edges (orange) and (b) deforms the reactive links in response to a moving obstacle (green) or (c) moves the vertices and removes the links. The links are deformed based on Newtonian Physics and Hooke's Law. Adapted from [14].

edge is always the same length or shorter than the sum of the two edges, which previously connected the two vertices. If an edge is within the repulsion area of an obstacle, the deformation function moves $\mathbf{q}_0$ to $\mathbf{q}_{\text{new}}$ projected onto the boundary of the repulsion area around the obstacle. The path deformation procedure is initiated for every change in the environment. If deformation does not find a path in $\mathcal{C}_{\text{free}}$, replanning, as proposed in earlier work [61], was suggested, in which planning and execution run parallel, ensuring replanning while the robot is moving. If obstacles in the environment disappear, the replanning framework ensures that this area will be accessible for new samples. However, no new vertices are placed in previously occupied areas. The algorithm's performance was assessed by testing with a 7-DOF manipulator in a 3D workspace in an environment with a static wall and a moving bar-shaped object as obstacles. The deformation of the edges according to changes in the environment was done successfully. The combination of edge adaptation and replanning was 4 and 6 times faster than with either deformation or replanning, respectively. This algorithm is undesirable for planning in incrementally changing environments because the entire roadmap is continuously adapted, while only a slight adaptation is necessary, resulting in unnecessary, time-consuming computations.

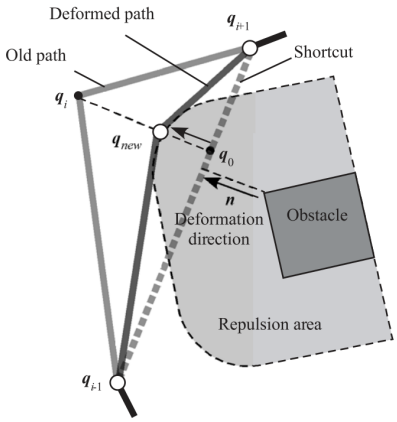


Figure 3.2: This figure presents the process of finding shortcuts in the replanning with path deformation algorithm [60], in which a direct connection from vertex $\mathbf{q}_{i-1}$ to vertex $\mathbf{q}_{i+1}$ can be made instead of being connected through vertex q_i, resulting in a shorter path due to triangular inequality. The algorithm deforms the shortcut edge if it is within the repulsion area of an obstacle and returns $\mathbf{q}_{new}$, which is a projection of the interpolated point $\mathbf{q}_0$ of an edge to the boundary of the repulsion area around the obstacle [60].

3.2. Roadmap adaptation based on vertex adaptation

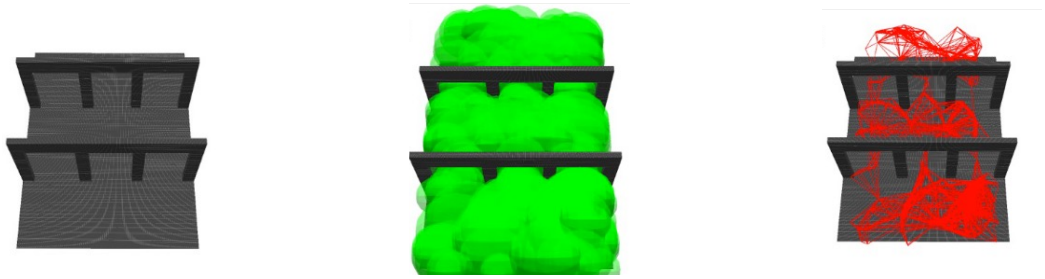
- An algorithm that adapts vertices by deliberately trading probabilistic completeness for computational efficiency is the elastic roadmap (ER) algorithm [58, 59]. The ER algorithm represents the roadmap in the workspace. Vertices are virtual placements of the robot in the environment, connected with an edge to another vertex if a controller can move the robot from one vertex to the other. ER generates vertices close to obstacle boundary features, e.g., at corners and next to edges, with the configuration

of the robot arm pointing towards the obstacle. Controllers constantly adapt vertices to allow reactivity in changing environments. When collision can no longer be avoided within the bounds of the controller, ER deactivates the vertices. The algorithm reactivates the vertices if an obstacle disappears or moves on. Lastly, it extracts a sequence of edge controllers. If an obstacle moves, the vertex moves along with the obstacle. This space, previously occupied by the vertex, is not covered with new vertices, resulting in this area not being accessible for motion planning. Experiments verify the working principle of the algorithm by assigning a task to move the end-effector of a robot in a 3D environment along a trajectory. The experiments deploy two robots: a 10-DOF mobile manipulator, the UMass mobile manipulator, in the presence of two to five mobile robots as obstacles or two static cones incrementally appearing in the environment, and with a stationary 12-DOF manipulator in the presence of a moving truss. The algorithm successfully adapted the roadmap and ensured the robots operate autonomously while adhering to all motion constraints. The errors of the end-effector tracking the path were mostly below $2cm$ in all three dimensions. This algorithm is not suitable for a robot in a store environment because it is computationally expensive as the vertex controllers need accurate real-time information about $\mathcal{C}_{\text{free}}$ to ensure the vertices remain valid, consequently increasing the planning time.

- To ensure that the ER algorithm can deal with environmental uncertainty associated with unstructured environments by using local sensor data which instantiates an ER, the expected shortest-path elastic roadmap (ESPER) algorithm [51] extends ER. ESPER assumes that the robot can only obtain information about the environment through onboard sensors, on top of a global map with static obstacles, that place and adjust vertices of the roadmap to reduce the computational complexity of dealing with uncertainty. Apart from locally determining whether a motion results in collision, this algorithm deals with edges in $\mathcal{CO}$ in the global environment in the currently known state by removing edges that are frequently in $\mathcal{CO}$ or for a long time. The Expected Shortest-Path algorithm [5] was used to compute a policy that provides, for each sensory input, the edges with minimum expected costs. Additionally, this algorithm makes it possible to wait for an edge to become unblocked, dependent on the estimation of how long this will take. Experiments assessed the performance of this algorithm with a 10-DOF mobile manipulator, consisting of a Nomadic XR-4000 holonomic mobile base with a 7-DOF Barrett WAM robotics arm mounted on top in a 3D environment with as a dynamic obstacle a human. ESPER ensured whole-body motion while taking the task constraints into account, based on two onboard sensors: an RGB-D sensor and a laser range finder. The end-effector error with respect to the base localisation error was $6cm$ at a maximum. The running time of ESPER and ER was equal if obstacles are moving slowly with respect to the robot velocity and up to 10% shorter for ESPER if the ratio of obstacle velocity to robot velocity was above 0.5. This algorithm is not suitable for high-DOF robots in incrementally changing environments for the same reason as ER.

- To ensure the ER algorithm can deal with more drastic changes in the environment, such as opening doors, the incremental elastic roadmap (IER) algorithm [34] extends the ER algorithm. IER implements a more efficient method to capture connectivity at all times. A sphere-based wavefront is the basis for the workspace connectivity graph [6]. The wavefront floods the free workspace with spheres and minimises the number of spheres by preferring large spheres over small ones. After that, the workspace connectivity graph is created by placing vertices at the sphere centres and adding edges between the vertices if the corresponding spheres overlap significantly. This graph guides an underlying sampling-based planner in $\mathcal{C}$ to locally generate new vertices and add them to the existing roadmap and remove invalid vertices if these are not within the free workspace. Figure 3.3 presents a visualisation of the roadmap connectivity graph. Experiments evaluated the performance of this algorithm. A 10-DOF mobile manipulator moved a tray upright to a goal location in an unknown 3D environment, in which four walls were placed and removed. The algorithm successfully updated the workspace connectivity graph and ensured navigation to the goal in continuous motion while satisfying the motion constraints. For non-drastic changes, as is the case in this thesis, this adaptation locally runs a sampling-based planner to explore the space, where adapting a single vertex would also be sufficient.

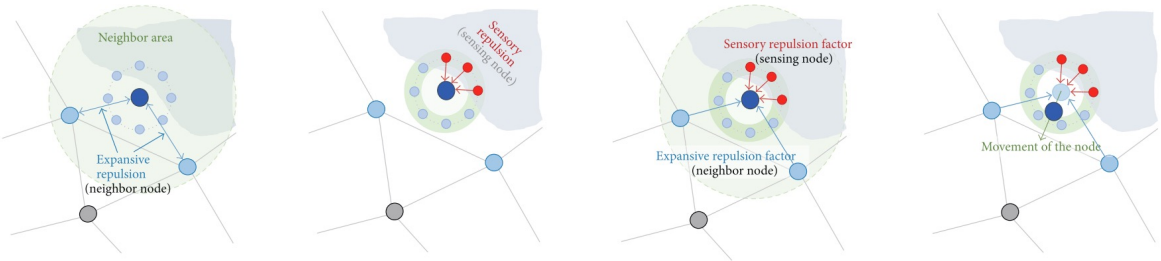
- The adaptation algorithm adapts the roadmap to changes in the environment by automatically moving the vertices to the free space [41]. This algorithm based the movement of vertices on the motion dynamics of evenly distributed gases in the environment and the repulsion of particles to each other [20].



(a) (b) (c)

Figure 3.3: These figures present an illustration of the workspace connectivity graph based on a sphere-based wavefront to guide an underlying sampling-based planner for roadmap adaptation in the incremental elastic roadmap (IER) algorithm [34]. In (a) the workspace that is decomposed by (b) maximum sized spheres to create the (c) workspace connectivity graph by creating vertices at the centers of the spheres and adding edges between the vertices if the spheres overlap significantly [34].

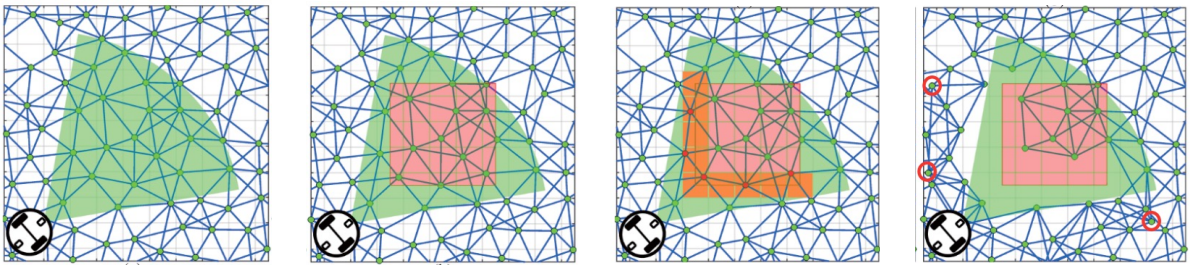
An expansive repulsion between vertices ensures that the vertices do not get too close to each other. Additionally, a sensory repulsion factor pushes the vertices away from the obstacles observed by the robot's sensors. The automatic movement of vertices to the free space ensures that this algorithm deals with disappearing obstacles. Figure 3.4 presents the repulsion between vertices and sensory repulsion. After rearranging the vertices, the algorithm updates the edges. Experiments with a 2-DOF two-link robot in a 2D environment with five different-sized rectangular dynamic obstacles showed that if the environment changes suddenly or if dynamic obstacles are present, the algorithm successfully adapted the roadmap. This algorithm is unsuitable for planning for a mobile manipulator because it is created for a 2D workspace and a low-DOF robot.



(a) (b) (c) (d)

Figure 3.4: These figures present a visualisation of the roadmap adjustment of the adaptation algorithm [41] based on the motion dynamics of gases with (a) expansive repulsion between vertices (nodes) and (b) sensory repulsion between an obstacle and a vertex that (c) are combined to initiate (d) movement of a vertex [41].

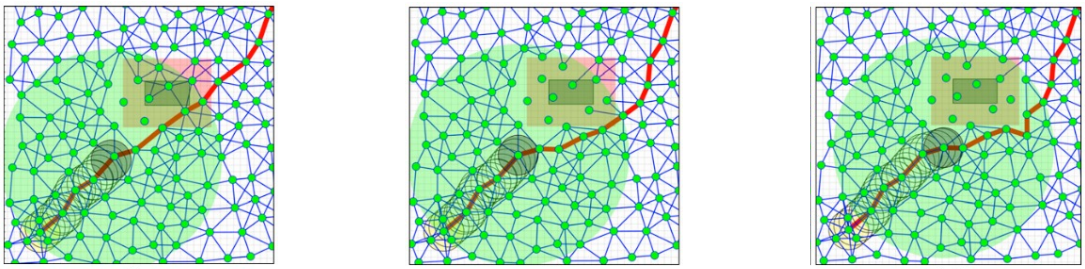
- The self-adjusting roadmaps algorithm [24] clusters the generated vertices in a grid-based structure based on their spatial coordinates, making it affordable to check for collision to deal with unknown obstacles quickly. The size of the grid cells depends on the number of obstacles in the environment. If the robot moves through the environment and senses an obstacle, the grid cells corresponding to the sensed part of this obstacle are marked. The algorithm pushes all vertices within the grid cell away from the obstacle to maintain the roadmap connectivity. Figure 3.5 shows the process of pushing away the vertices. Experiments in different 2D environments with stationary and randomly moving polygonal-shaped obstacles on a 3-DOF mobile base showed that the running time of this algorithm was lower, with a higher success rate and similar path length to conventional sampling-based algorithms [16, 31] and ER. Additionally, the algorithm outperformed the conventional algorithms when dealing with narrow corridors. If obstacles in the environment disappeared, the algorithm ensured that this area is accessible for new samples. However, no new vertices were placed in previously occupied areas. This algorithm is unsuitable for planning for a mobile manipulator because it is created for a 2D workspace and a low-DOF robot, and merely the vertices are adapted, not the edges.
- The incremental adaptive randomized roadmaps algorithm was proposed [25, 26] to locally adapt its roadmap using the information of clustered vertices, based on spatial coordinates and their uncer-



(a) (b) (c) (d)

Figure 3.5: These figures present the roadmap adjustment proposed in the self-adjusting roadmaps algorithm [24]. An initial roadmap covers the entire space, ignoring the obstacles: in (a) this roadmap, the robot and the area covered by the sensor (green) are visualised. If (b) an obstacle (red) appears to be on the path of the robot, (c) the grid cells in which the part of the obstacle sensed by the robot (orange) are determined and (d) the vertices in the corresponding grid cells are pushed away (red circles). After this roadmap adjustment, the roadmap is ready to be used for graph search [24].

tainty. The algorithm is based on PRM* [21], an asymptotically optimal variant of PRM, and performs hybrid sampling classification and self-adjustment to deal with planning uncertainty. The vertices are stored in a matrix structure based on their Cartesian coordinates to ensure a cheap computational search for vertices in a specific region; based on this matrix, an initial roadmap is created. A second matrix with uncertainty data corresponding to the vertices is created. As the robot moves, it scans the surrounding area. The collision status is updated in the uncertainty matrix for all visible cells if the status differs from the previously known status. After that, the matrix with the stored vertices will be updated based on the changes in the uncertainty matrix. The algorithm removes all connections to the newly occupied vertex and creates new connections to the newly free vertices. Additionally, it checks the collision status of the vertices around those with a change in collision status. Figure 3.6 shows the roadmap adaptation. This procedure repeats with a predefined constant frequency. The algorithm deals with the disappearance of obstacles by changing the value in the uncertainty matrix, making the vertex accessible. Experiments in four different 2D office environments on a 3-DOF mobile base robot with either two (unknown) rectangular static or dynamic obstacles of different sizes showed successful navigation from start to goal, without stopping if the solution path is updated. The running time of the proposed algorithm was shorter, the path length was slightly shorter, and the failure rate was lower than that of single-query algorithms reconstructing the roadmap [3, 42]. This algorithm is unsuitable for planning for a mobile manipulator because it is created for a 2D workspace and a low-DOF robot, and merely the vertices are adapted, not the edges.



(a) (b) (c)

Figure 3.6: These figures visualise the roadmap adaptation method of the incremental adaptive randomized roadmaps algorithm [25]. If (a) the robot is following a planned path (red) and a new obstacle (grey) is sensed, (b) the associated vertices are temporarily deactivated and the corresponding edges are removed, and (c) the path is adjusted accordingly. The vertices and edges, and therefore the path, are continuously updated due to changes in the environment. The grey circle is the robot, the green circle is the vision range of the robot, and the red area around the obstacle is the expanded obstacle for safety where the neighbouring vertices to the vertices in $\mathcal{CO}$ are additionally deactivated [25].

3.3. Discussion

None of the discussed algorithms is applicable for coupled motion planning for a mobile manipulator in an incrementally changing environment. No algorithm meets all requirements in Section 1.4, as is presented in Table 3.1. Therefore, in this research, we aim to develop an algorithm that meets all requirements, unlike the existing algorithms, enabling roadmap adaptation for coupled motion planning for a mobile manipulator in an incrementally changing environment.

The proposed algorithm extends the principle of vertex and edge assignment for quick lookup of the self-adjusting roadmap algorithm proposed in [25] and extends this for a high-DOF robot in a 3D workspace and for dealing with invalid edges. The remaining algorithms are less suitable as foundation because they have an unnecessary high planning time due to the adaptation [34, 51, 58–60], resulting in, in the worst case, the robot remaining stationary during the roadmap adaptation or are challenging to extend for a high-DOF robot [14, 25, 26, 41].

			Algorithm							Implementation												Performance											Requirements						
			Sensors		Vertex localization		Adjusting the roadmap			Workspace			Robot			Incremental obstacles				Test		Running time compared to			Path length compared to			Success rate compared to			Error compared to	R1	R2			R3	R4	R5
Paper	Algorithm name	Citation	On-board sensors	Uncertainty incorporated	With respect to obstacles	Clustered based on coordinate	Move vertices	Deform edges	Remove edges	2D	3D	2-link robot	Mobile robot	Manipulator	Mobile manipulator	3-DOF robots(s)	Rigid object(s)	Human	Walls	Simulation	Real-world	ER	Conventional SQ algorithm(s)	Conventional MQ algorithm(s)	ER	Conventional SQ algorithm(s)	Conventional MQ algorithm(s)	ER	Conventional SQ algorithm(s)	Conventional MQ algorithm(s)	ER	Multi-query	Adapt vertices and edges	Appearing obstacles	Disappearing obstacles	High-DOF robot	3D workspace	Fast
Gayle et al. (2007)	Reactive deforming roadmaps (RDR)	[14]						x	x	x			x			x				x											x	x	x	x			
Yang et al. (2007), Yang et al. (2010)	Elastic roadmaps (ER)	[58,59]	x		x		x		x		x		x	x	x	x	x			x	x											x	x	x	x	x	x	
Yoshida et al. (2011)	Replanning with path deformation	[60]						x			x			x			x			x												x	x	x			x	
Sieverling et al. (2014)	Expected shortest-path elastic roadmap (ESPER)	[51]	x	x	x		x				x				x			x		x	x	+									–	x	x	x	x	x	x	
Lehner et al. (2015)	Incremental elatic roadmap (IER)	[34]	x				x				x				x				x	x	x											x	x	x	x	x	x	
Park et al. (2016)	Adaptation algorithm	[41]	x				x			x		x					x			x												x	x	x	x			x
Khaksar et al. (2018)	Self-adjusting roadmap	[24]	x			x	x			x			x				x			x	x	+	+	+	=	=	=	+	+	+		x		x				x
Khaksar et al. (2018), Khaksar et al. (2020)	Incremental adaptive randomized roadmaps	[25,26]	x	x		x			x	x			x				x			x	x		+			+			+			x		x	x			x

Table 3.1: Table summarizing the information about the algorithms that locally adapt a roadmap discussed in this chapter, their performance compared to other algorithms in the table or conventional single-query (SQ) or multi-query (MQ) algorithms, and whether the algorithms meet the requirements from Section 1.4. Note that a + in a cell indicates better performance on this metric for the proposed algorithm it does not indicate that the value for this metric is higher. The empty cells in the cells comparing two algorithms indicate that no comparison between these algorithms is presented in the paper proposing the algorithm.

4

Methods

This chapter describes the proposed algorithm, the adaptive roadmap algorithm (ARM), that locally adapts its roadmap for mobile manipulators to ensure the roadmap is reusable for multiple queries even though incremental changes in the environment occur. The algorithm extends the principle of vertex and edge assignment for quick lookup of the self-adjusting roadmap algorithm [25]. Firstly, we discuss requirements on completeness, optimality and anytime of motion planning algorithms.

4.1. Requirements on the completeness, optimality and anytime

The motion planning properties completeness, optimality, and anytime, introduced in Section 2.2, are not requirements of ARM.

All existing algorithms that perform roadmap adaptation, discussed in Chapter 3, relax the completeness requirement to compute a solution in a reasonable amount of time. As we aim to reduce the operation time, we want to decrease the computation time necessary for adapting the roadmap.

Optimality is not a requirement of the algorithm because due to changes in the environment, the optimality of the algorithm would cause recomputation of the path to ensure it remains optimal, which increases the planning time. As we aim to reduce the operation time, we benefit from decreasing the planning time.

The result of relaxing the completeness and optimality of the algorithm is that the algorithm generates motion plans for high-DOF robots at interactive rates.

Lastly, the algorithm does not need to be anytime since we do not aim to provide solutions if the algorithm is terminated early. If the roadmap adaptation algorithm is terminated early the roadmap is not entirely in $\mathcal{C}_{\text{free}}$. Therefore, we do not need solutions of an early terminated algorithm.

4.2. Self-adjusting roadmap algorithm

We establish extensions to the self-adjusting roadmap algorithm [25] (Chapter 3) since it does not meet requirements R2, R3, and R4. We shortly introduce extensions: to $\mathcal{W} = \mathbb{R}^3$ (R4), to a high-DOF robot (R3), and to deal with edges in $\mathcal{CO}$ (R2). We do not extend the algorithm to deal with disappearing obstacles (R2) since it does not place vertices in the previously occupied areas. However, ARM makes the area accessible for vertices to extend the algorithm in future work by placing vertices in areas where obstacles have disappeared.

4.2.1. 3D workspace

The assignment of a robot configuration $\mathbf{q}$ must be extended from $\mathcal{W} = \mathbb{R}^2$ to $\mathcal{W} = \mathbb{R}^3$ to adhere to R4. ARM represents the environment with 3D grid cells for assigning configurations instead of the 2D grid cells; we refer to this structure as the 3D grid. The cells in the 3D grid are either occupied or free, which can be updated based on appearing or disappearing obstacles.

4.2.2. High-DOF robot

ARM represents each robot configuration by a bounding box around the mobile manipulator in $\mathcal{W}$ since this is the space that contains information about the updated environment. This representation enables using the 3D grid to assess whether a configuration is valid. For the circular mobile robot in the self-adjusting roadmap

algorithm, this representation is more straightforward because $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$ easily maps to $\mathcal{W} = \mathbb{R}^2$ by inflating the configuration with the robot radius. For a mobile manipulator, the Cartesian coordinates of all points on the robot must be taken into account to prevent, for example, a collision of the arm while the base is in $\mathcal{C}_{\text{free}}$. ARM determines the bounding box based on the Cartesian coordinates of informative points on the robot. ARM assigns this box to 3D grid cells, and if an associated grid cell is occupied, ARM initiates the roadmap adaptation.

We altered the vertex adaptation procedure for a high-DOF robot by random sampling in $\mathcal{C}$ in the neighbourhood of the invalid vertex to generate a replacing vertex. In [25], the vertex is pushed away from the centre of the appeared obstacle by altering the Cartesian coordinates of the base. However, for the extension to a high-DOF robot, ARM adapts all DOFs of the robot to ensure a configuration is collision-free. ARM performs random sampling in $\mathcal{C}$, which this is a commonly used method to generate robot configurations. ARM samples in the neighbourhood of the invalid vertex to maintain the connectivity of the roadmap. If the randomly drawn sample is collision-free, ARM adds it as a vertex and connects it to the roadmap.

4.2.3. Invalid edges

ARM removes invalid edges to ensure the entire roadmap is in $\mathcal{C}_{\text{free}}$. In [25], the edges are not updated, except when the source or target vertex of the edge is associated with occupied grid cells. However, if the grid cells to which the source and target vertex are assigned are free, and the middle of the edge is associated with occupied grid cells, the self-adjusting roadmap algorithm does not initiate adaptations. ARM assigns the edges to grid cells by interpolating them and assigning the interpolated configurations to grid cells. If a grid cell is occupied, the algorithm removes the associated edges.

4.3. Method overview

The adaptive roadmap algorithm (ARM) is a multi-query sampling-based motion planning algorithm that can locally adapt vertices and edges of the graph to account for incremental changes in the environment. The algorithm requires an initial roadmap, generated with a sampling-based planner such as PRM. ARM generates a 3D grid to represent the workspace. The grid cells are marked as occupied or free based on the presence of obstacles in the environment. To determine what vertices and edges of the roadmap need to be updated if the occupancy of the 3D grid changes due to an incremental change, a mapping from the grid cells to the vertices and edges is required. To acquire this mapping, the algorithm represents the vertices in $\mathcal{W}$ by a 3D bounding box surrounding the entire robot configuration and the edges by surrounding interpolated configurations. ARM uses these representations to assign all vertices and edges of the roadmap to the 3D grid cells to obtain a mapping. ARM initiates the roadmap adaptation due to a changing occupancy of grid cells and resamples the vertices and removes the edges associated with occupied grid cells. The local adaptation of the roadmap ensures it is reusable if obstacles appear. Figure 4.1 visualises these steps. The adapted roadmap $R_a(V, E)$ can be extracted to perform a graph search and find a path. Then, if the environment changes incrementally, this roadmap is adapted.

4.4. Method description

In this section, we elaborate on the parts of the proposed algorithm indicated with bright blue in Figure 4.1.

4.4.1. 3D grid generation

The proposed algorithm represents the environment with a 3D occupancy grid. The size of the grid cells in the 3D grid are denoted by δ_k, where k denotes the dimension of $\mathcal{W}$: $k \in \{x, y, z\}$. Figure 4.2a presents the 3D grid structure in an environment with obstacles. Initially, the algorithm marks all grid cells that are (partially) occupied by the static obstacles (visualised in black) as occupied (visualised in grey). Then, if an obstacle appears or disappears, ARM updates the occupancy of the grid cells associated with this obstacle. We refer to a grid cell by its grid index (n_x, n_y, n_z), where n_x, n_y and n_z are the indices in the x-, y- and z-dimension in the grid, respectively. We denote the set of occupied grid indices by IO.

4.4.2. Assignment of vertices and edges to cells

The algorithm assigns vertices and edges to 3D grid cells to ensure checking which vertices and edges are in-collision is quick by extracting the vertices and edges associated with the changed grid cells. The assignment of vertices and edges to the 3D grid cells consists of three steps:

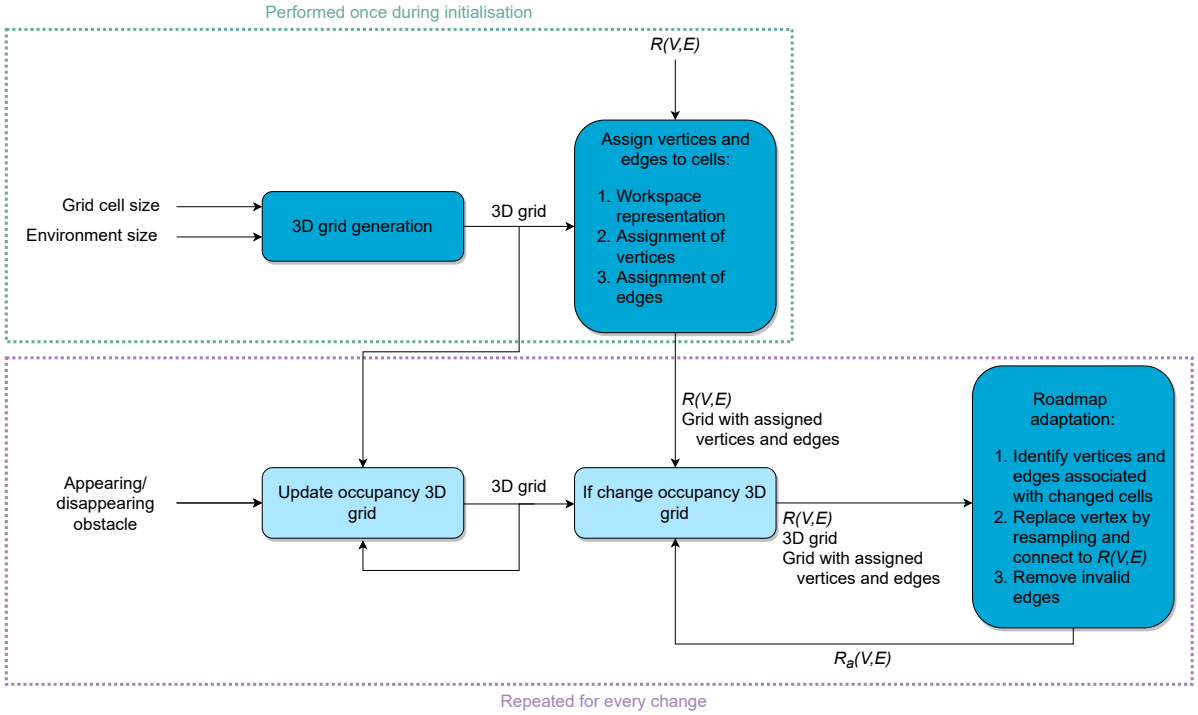


Figure 4.1: This flow diagram describes the adaptive roadmap algorithm (ARM) proposed in this thesis. The initialisation (green box) consists of the 3D grid generation and the assignment of vertices and edges to the 3D cells. ARM requires an initial roadmap $R(V,E)$. The algorithm generates a 3D grid that represents the workspace. The grid cells can be marked occupied or free to represent the occupancy of the environment. Next, it assigns the vertices and edges to the grid cells to obtain a mapping from grid cell to associated vertices and edges, which ensures a quick lookup of invalid vertices and edges if the occupancy of a grid cell is changed. For every incremental change, the occupancy of the 3D grid is updated and the roadmap adaptation of the vertices and edges associated with the occupied grid cells is initiated (purple box). The adapted roadmap $R_a(V,E)$ can be extracted to perform a graph search and find a path. Section 4.4 discusses the bright blue steps in the flow chart.

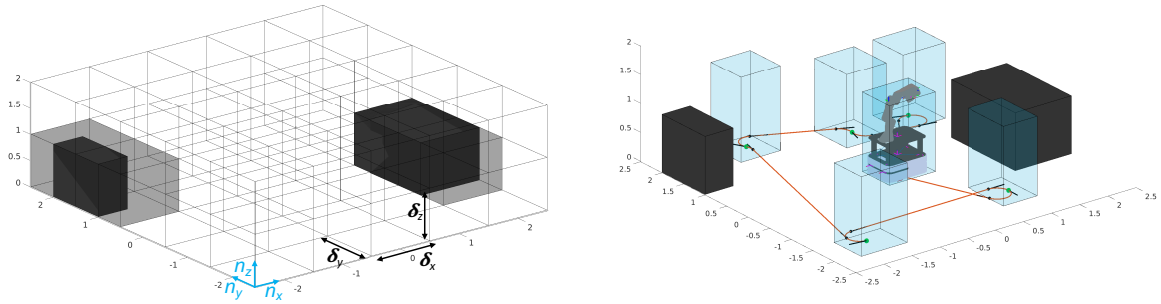
1. **Workspace representation:** to assign configurations to grid cells in the 3D grid, ARM represents them in $\mathcal{W}$ by creating a 3D bounding box $bb_{\mathbf{q}}$ around the robot in a certain configuration. The bounding box is described by six values: the minimum $bb_{\min_k}$ and maximum $bb_{\max_k}$ boundary for every dimension: $k \in \{x, y, z\}$. ARM creates the bounding box based on the Cartesian coordinates of informative points on the robot since representing the entire robot with Cartesian coordinates is time-consuming. For mobile manipulators, the informative points are the Cartesian coordinates of points exterior of the base, such as the corner points and the centres of the arm joints, obtained from forward kinematics. The algorithm determines the minimum and maximum value for every dimension from these points and sets them to $bb_{\min_k}$ and $bb_{\max_k}$, respectively. The bounding boxes have a yaw of zero to simplify the assignment to grid cells. We present a visualisation of the bounding box representations of mobile manipulators in $\mathcal{W}$ in Figure 4.2b.

2. **Assignment of vertices:** for every cell in the 3D grid, ARM determines what vertices are associated with it based on the bounding box representation of all vertices. Suppose we are given some bounding box $bb_{\mathbf{q}}$ for the vertex represented by $\mathbf{q}$. For $k \in \{x, y, x\}$ the minimum and maximum coordinates of the bounding box are given by $bb_{\min_k}$ and $bb_{\max_k}$. We then denote by $IQ_{\mathbf{q}}$ the set of indices associated to bounding box representing $\mathbf{q}$, which is the set of all indices (n_x, n_y, n_z) such that

$$\lfloor bb_{\min_k}/\delta_k \rfloor \leq n_k \leq \lceil bb_{\max_k}/\delta_k \rceil \tag{4.1}$$

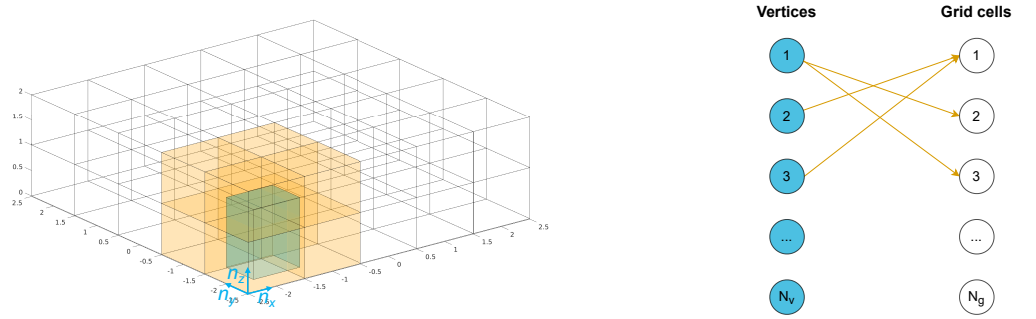
   for $k \in \{x, y, z\}$. $\lfloor \ \rfloor$ and $\lceil \ \rceil$ round the number to the lower and higher integer, respectively. The vertex is assigned to the associated grid cells. Figure 4.3 presents visualisations of the assignment of vertices to grid cells.

3. **Assignment of edges:** to ensure the entire roadmap is collision-free, and not merely the vertices, ARM determines the edges associated with the grid cells by assigning interpolated configurations of the edge



(a) (b)

Figure 4.2: These figures illustrate (a) the representation of the environment by a 3D grid and its occupancy (Section 4.4.1) and (b) the representation of the roadmap vertices in $\mathcal{W}$ (Section 4.4.2) in the same environment to allow assignment of the vertices to grid cells in the initialisation of the proposed adaptive roadmap algorithm. The 3D grid representation of the environment consists of grid cells of size $\delta_x \times \delta_y \times \delta_z$, where grid cells can be marked occupied (grey) or free (white). The indices of the grid cells in $x-$, $y-$ and $z-$ direction are denoted by n_x, n_y and n_z, respectively. Initially, ARM marks all grid cells associated with static obstacles (black) as occupied; if an obstacle appears or disappears, the occupancies of the grid cells associated with this obstacle are updated. We visualise the roadmap (b) in $\mathcal{W}$ without the 3D grid from (a) for clarity. The blue bounding boxes are the vertices in $\mathcal{W}$, of which the size is dependent on the entire robot configuration. We visualise the edges by projecting the (x, y) values of the path (orange lines) and the source and target vertex (green dots) on the ground plane. The edges are created with Dubins curves with a minimum turning radius of 0.2m. Note that we do not visualise the arm configurations in the projection of the edges.



(a) (b)

Figure 4.3: These figures illustrate the assignment of vertices to grid cells performed in the initialisation of the proposed adaptive roadmap algorithm. In (a) the $\mathcal{W}$-representation (blue) of one of the vertices from Figure 4.2b is visualised in the 3D grid (Figure 4.2a), and assigned to the eight associated grid cells (orange) using Equation (4.1). The indices of the grid cells in $x-$, $y-$ and $z-$ direction are denoted by n_x, n_y and n_z, respectively. This assignment is performed for all vertices of the roadmap to obtain the mapping from vertex to grid cell in (b), where N_v denotes the number of vertices and N_g the number of grid cells.

to grid cells. ARM represents the interpolated configurations by bounding boxes and assigning them to grid cells, equivalent to the assignment of vertices. The interpolation strategy is equivalent to the interpolation for collision checking of the edges in Section 2.4.3. However, the interpolation resolution, determined with Equation (2.9), is larger than the resolution for collision checking of edges to prevent unnecessary computations. We aim to find interpolated configurations on the edge that are likely in different grid cells. Therefore, $e_{L_{val_j}}$ in Equation (2.9) is based on the grid cell size. For this application, $e_{L_{val_x}}$ and $e_{L_{val_y}}$ describe the distance between two configurations that are likely in different grid cells, based on the DOFs describing the x- and y-coordinate of the base. For every edge, Δq_{e_x} and Δq_{e_y} in Equation (2.9) are computed by setting $e_{L_{val_x}}$ to δ_x and $e_{L_{val_y}}$ to δ_y. The fixed resolution for all DOFs Δq_e for the assignment of the edge to grid cells with Equation (2.9) is the minimum value of the two to prevent missing an associated grid cell:

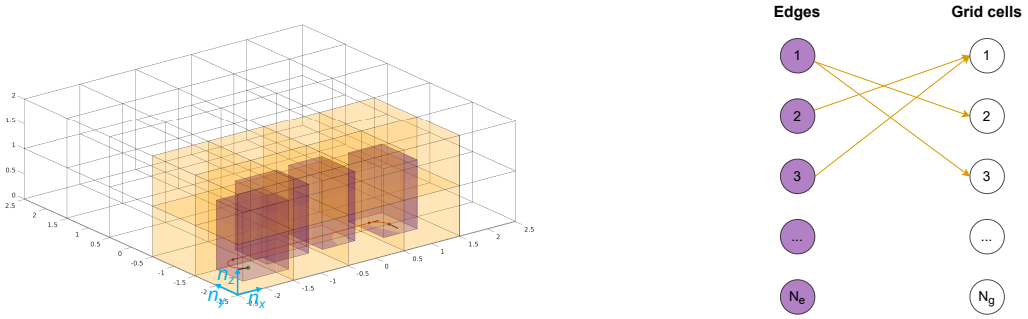
$$\Delta q_e = \min(\Delta q_{e_x}, \Delta q_{e_y}). \tag{4.2}$$

ARM additionally includes the source and target vertex of the edge as well as interpolated configurations for the assignment to grid cells. We denote by IE_e the set of indices associated with an edge

determined by the assignment of interpolated configurations:

$$IE_e = \bigcup_{m=1}^{N_m} IQ_{\mathbf{q}_m} \tag{4.3}$$

for $m = 1, ..., N_m$, where N_m is the number of interpolated configurations. The edge is assigned to the associated grid cells. Figure 4.4 presents visualisations of the assignment of edges to grid cells.



(a) (b)

Figure 4.4: These figures illustrate the assignment of edges to grid cells performed in the initialisation of the proposed adaptive roadmap algorithm. In (a) one of the edges from Figure 4.2b (orange line on the $xy-$plane) is visualised in the 3D grid (Figure 4.2a) with the corresponding interpolated configurations in $\mathcal{W}$ (purple) for the assignment. The interpolated configurations consist of the source and target vertex of the edge, and configurations in between that are likely in different grid cells (item 3 in Section 4.4.1). The indices of the grid cells in $x-$, $y-$ and $z-$ direction are denoted by n_x, n_y and n_z, respectively. This assignment is performed for all edges of the roadmap, to obtain the mapping from edge to grid cell in (b), where N_e denotes the number of edges and N_g the number of grid cells.

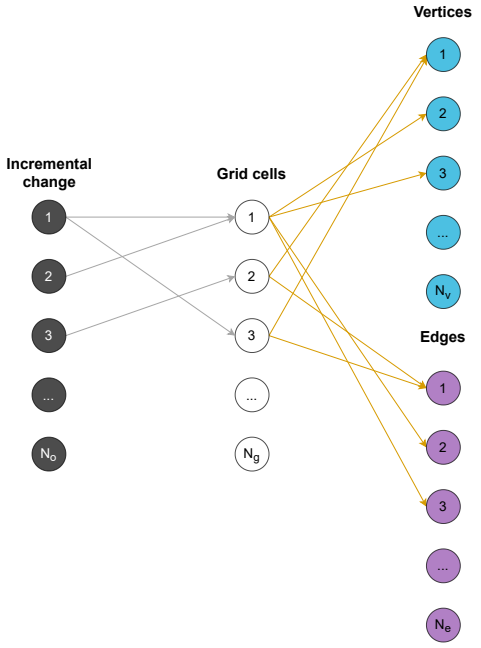


Figure 4.5: This diagram illustrates the mapping from grid cells occupied by the incremental changes to the associated vertices and edges the proposed adaptive roadmap algorithm uses to determine what vertices and edges need adaptation. Firstly, the incremental changes are assigned to grid cells (Section 4.4.1). Then, the previously performed assignment of vertices and edges to the grid cells (Section 4.4.2) allows us to map from the grid cells to the associated vertices and edges. These vertices and edges need to be adapted. ARM performs this mapping for all incremental changes. N_o, N_g, N_v and N_e denote the number of incremental changes, grid cells, vertices and edges, respectively.

4.4.3. Roadmap adaptation

The algorithm replaces invalid vertices by random sampling in $\mathcal{C}$ in the neighbourhood of the invalid vertex and removes invalid edges. We present the roadmap adaptation in Algorithm 3. The roadmap adaptation

Algorithm 3 Roadmap adaptation by ARM

Input: $R(V,E)$: Roadmap
IO: occupied grid cells
M: mapping from grid cells to assigned vertices and edges
k: number of closest neighbours to examine for configuration
d: resampling area that defines the bounds on $\mathcal{C}_{\text{sub}}$
Output: $R_a(V,E)$: Adapted roadmap

1: **if** change in IO **then**
2: **for all** $v \in V$ associated with changed IO (from M) **do** ▷ Section 4.4.3 step 1
3: $\mathbf{q}_{\text{inv}} \leftarrow v$
4: $V = V \setminus \mathbf{q}_{\text{inv}}$
5: $E = E \setminus e(\mathbf{q}_{\text{inv}})$
6: $\mathcal{C}_{\text{sub}} \leftarrow$ subspace of $\mathcal{C}$ to sample close by $\mathbf{q}_{\text{inv}}$ ▷ Equation (4.4)
7: updated = false
8: **while** updated = false **do**
9: $\mathbf{q}_{\text{new}} \leftarrow$ random configuration in $\mathcal{C}_{\text{sub}}$
10: $bb_{\mathbf{q}_{\text{new}}} \leftarrow$ workspace bounding box representing $\mathbf{q}_{\text{new}}$ ▷ Section 4.4.2 step 1
11: $IQ_{\mathbf{q}_{\text{new}}} \leftarrow$ grid cell indices associated with $\mathbf{q}_{\text{new}}$ ▷ Equation (4.1)
12: **if** $IQ_{\mathbf{q}_{\text{new}}} \cap IO = \emptyset$ **then**
13: $\mathrm{N}_{\mathbf{q}_{\text{new}}} \leftarrow k$ neighbours to $\mathbf{q}_{\text{new}}$ in V
14: **for all** $\mathbf{q}' \in \mathrm{N}_{\mathbf{q}_{\text{new}}}$ **do**
15: $IE_{e'} \leftarrow$ grid cell indices associated with $(\mathbf{q}_{\text{new}},\mathbf{q}')$ ▷ Equation (4.3)
16: **if** $IE_{e'} \cap IO = \emptyset$ **then**
17: **if** $\mathbf{q}_{\text{new}} \notin V$ **then**
18: $V \leftarrow V \cup \mathbf{q}_{\text{new}}$
19: **end if**
20: $E \leftarrow E \cup (\mathbf{q}_{\text{new}},\mathbf{q}')$
21: **end if**
22: **end for**
23: **if** $\mathbf{q}_{\text{new}} \in V$ **then**
24: updated = true
25: **end if**
26: **end if**
27: **end while**
28: **end for**
29: **for all** $e \in E$ associated with changed IO (from M) **do** ▷ Section 4.4.3 step 1
30: $E = E \setminus e$ ▷ Section 4.4.3 step 3
31: **end for**
32: **end if**
33: **return** $R_a(V,E)$

can be described by the following steps, which are additionally visualised in Figure 4.6 after the discussion of each step:

1. **Identify vertices and edges associated with changed cells:** identify vertices (line 2) and edges (line 29) associated with the changed grid cell indices (IO) from the mapping M, which is the mapping from grid cells to its associated vertices and edges, determined from the previously performed mapping of vertices and edges to grid cells (Section 4.4.2). We present a visualisation of the mapping from occupied grid cells to vertices and edges in Figure 4.5.

2. **Replace vertex by resampling and connect to *R(V,E)*:** firstly, ARM removes the configuration $\mathbf{q}_{\text{inv}}$ representing the invalid vertex (line 4), including the in- and outgoing edges (line 5). To ensure no areas become inaccessible due to the absence of vertices, ARM generates a configuration for the replacing vertex $\mathbf{q}_{\text{new}}$ nearby $\mathbf{q}_{\text{inv}}$ in $\mathcal{W}$. By generating a replacing vertex nearby the invalid vertex, a narrow passage in $\mathcal{W}$ due to an appearing obstacle, which often suggests the presence and location of a narrow passage in $\mathcal{C}$, is more likely to remain accessible. To obtain $\mathbf{q}_{\text{new}}$ nearby $\mathbf{q}_{\text{inv}}$, ARM specifies a subspace of $\mathcal{C}$ to draw samples from: $\mathcal{C}_{\text{sub}} \subset \mathcal{C}$, with bounds on the x- and y-dimension based on $\mathbf{q}_{\text{inv}}$ (line 6). ARM sets the bounds to a a predetermined distance d from $\mathbf{q}_{\text{inv}}$. The values for the x- and y-dimension of $\mathbf{q}_{\text{new}}$ are bounded by:

$$\begin{aligned} \mathbf{q}_{\text{inv}}(x) - d \;\leq\; \mathbf{q}_{\text{new}}(x) \;\leq\; \mathbf{q}_{\text{inv}}(x) + d \\ \mathbf{q}_{\text{inv}}(y) - d \;\leq\; \mathbf{q}_{\text{new}}(y) \;\leq\; \mathbf{q}_{\text{inv}}(y) + d. \end{aligned} \tag{4.4}$$

The remaining DOFs do not have additional bounds in $\mathcal{C}_{\text{sub}}$, other than the joint limits.

If ARM draws a random sample $\mathbf{q}_{\text{new}}$ from $\mathcal{C}_{\text{sub}}$ (line 9), it determines the indices of the associated grid cells $IQ_{\mathbf{q}_{\text{new}}}$ of the random sample using the bounding box $\mathcal{W}$-representation $bb_{\mathbf{q}_{\text{new}}}$ from Section 4.4.2 (line 10-11). If no associated grid cell is occupied (line 12), ARM determines with a nearest neighbour search the set of vertices of the roadmap $\mathbf{q}_{\text{new}}$ can connect to: $\text{N}_{\mathbf{q}_{\text{new}}}$ (line 13). ARM assigns these connections, also candidate edges e', between $\mathbf{q}_{\text{new}}$ and corresponding neighbours $\mathbf{q}'$ to grid cell indices (line 15). If at least one connection exists that is not associated with occupied grid cell indices IO (line 16), ARM adds the sample as a vertex (18) and the valid connections as edges $(\mathbf{q}_{\text{new}}, \mathbf{q}')$ (line 20) to the roadmap. Lastly, ARM assigns $\mathbf{q}_{\text{new}}$ and its associated edges to grid cells and adds them to M, to prepare for future incremental changes. The vertex replacement is completed if the first random sample $\mathbf{q}_{\text{new}}$ adheres to the remaining constraints, is not associated with occupied grid cells and is connectable to the roadmap. The algorithm performs random sampling until a $\mathbf{q}_{\text{new}}$ satisfies these conditions (line 8-27).

3. **Remove invalid edges:** the algorithm removes the invalid edges (line 30). Figure 4.6b shows an edge that is in-collision, while the source and target vertex of the edge are not, pointing out the importance of assigning and removing the invalid edges.

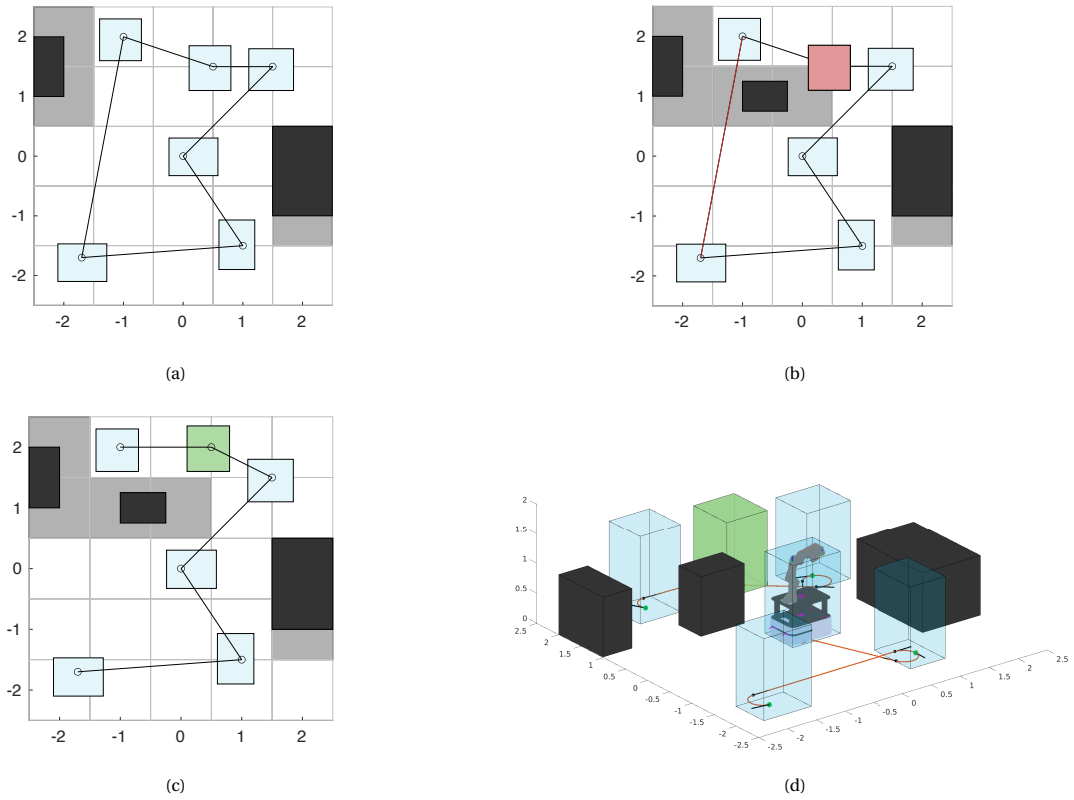


Figure 4.6: This figure illustrates the roadmap adaptation performed by the proposed adaptive roadmap algorithm. The grid cells associated with the obstacles (black) are marked as occupied (grey) and the bounding boxes representing the vertices are visualised in light blue. We represent the initial roadmap from Figure 4.2b in (a) 2D by projecting the bounding boxes and the 3D grid to the ground plane, and visualizing the edges with straight lines (black) for a clearer visualisation of the algorithm steps. In (b) an obstacle is added and the associated grid cells are marked (grey), and the associated vertex and edge, that require adaptation, are determined (red). (c) Shows the adapted roadmap, where one vertex is adapted (green) and one edge is removed, which (d) visualises in 3D.

4.5. Parameters

The magnitude of several parameters considerably influences the performance of ARM: the initial roadmap, 3D grid cell size, nearest neighbours, the safety margin of the bounding box representation of the robot, and the resampling area.

4.5.1. Initial roadmap

The performance of ARM depends predominantly on the initial roadmap since ARM does not generate additional vertices besides replacing the invalid vertices. The roadmap must consist of sufficient vertices and edges to capture the connectivity of $\mathcal{C}_{\text{free}}$ to ensure the roadmap contains a path between any collision-free $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$. We know a path exists as this is in the definition of an incrementally changing environment Chapter 1. Additionally, if a roadmap that consists of few vertices is adapted, the roadmap may become disconnected. The configurations $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ may connect to two different connected components, resulting in failure of the algorithm. Therefore, the initial roadmap must consist of sufficient vertices. However, if it consists of many vertices, ARM initiates many adaptations if one obstacle appears. One must verify whether the initial roadmap captures the connectivity of $\mathcal{C}_{\text{free}}$ without too many vertices.

4.5.2. 3D grid cell size

The size of the 3D grid cells that ensures the best performance is dependent on the environment. The grid cells must not be too large to prevent large areas from becoming inaccessible due to an occupied cell because an obstacle occupies a small part of the grid cell. This could result in narrow passages becoming inaccessible. Therefore, the width of the narrow passages must be taken into account when determining the grid cell size. Additionally, if an environment consists of obstacles with curved bounds, a large grid cell size results in inaccessibility of areas in the environment close by the curved obstacle bounds.

If the grid cells are small with respect to the appearing obstacles, more computations are necessary before the adaptation is initiated. The components δ_x and δ_y are dependent on the size of the incremental changes, and δ_z is dependent on whether the environment contains obstacles above ground level. If δ_z is 1m, and an obstacle appears with lower bound $z = 1.5$m, the area from $z = 1$m to $z = 2$m is marked as occupied due to the discretisation: the robot is allowed underneath if it is below 1m. If δ_z is 0.5m, the robot can go underneath if it is below 1.5m.

4.5.3. Nearest neighbours

The number of nearest neighbours $\mathbf{q}_{\text{new}}$ attempts to connect to affects the computational costs of the roadmap adaptation. If the value is too large, the algorithm performs many collision checks to check whether a valid motion exists between two vertices, while $\mathbf{q}_{\text{new}}$ was already connected to the roadmap. If this value is too small, the new vertex will likely not be able to connect to the roadmap, and the algorithm keeps resampling until it finds a connectable $\mathbf{q}_{\text{new}}$, resulting in a time-consuming roadmap adaptation. If the environment contains few obstacles, the necessary number of nearest neighbours for connecting the new vertex to the roadmap is lower than in an obstacle-cluttered environment because more connections to nearest neighbours are collision-free.

4.5.4. Safety margin bounding box

A safety margin can be applied to inflate the bounding box used for the representation of the robot in $\mathcal{W}$ to ensure the robot does not get too close to obstacles. However, if the safety margin is too large, unnecessary roadmap adaptations will be initiated due to the conservative representation of the robot in $\mathcal{W}$. We suggest determining the safety margin depending on the environment.

4.5.5. Resampling area

If the area in which ARM generates a replacing vertex is too small, finding a replacing vertex is computationally expensive, and if it is too large, it affects the connectivity of $\mathcal{C}$. The space in which replacing vertices are generated is $\mathcal{C}_{\text{sub}}$, of which the size is denoted by the parameter d. If d is too small, $\mathcal{C}_{\text{sub}}$ is mainly occupied by the appeared obstacle, and the new samples are likely to be in-collision, which makes finding $\mathbf{q}_{\text{new}}$ computationally expensive. However, if d is too large, $\mathbf{q}_{\text{new}}$ may be further from $\mathbf{q}_{\text{inv}}$, which could result in an area becoming inaccessible due to the absence of vertices.

4.6. Discussion

The proposed algorithm performs roadmap adaptation based on the mapping from 3D grid cells representing the workspace to vertices and edges, which allows quick lookup of what vertices and edges are associated with occupied grid cells and need to be adapted to ensure the roadmap remains collision-free. The parameters discussed in Section 4.5 greatly influence the performance of the algorithm and are dependent on the planning problem to be solved. The parameter settings of the proposed algorithm must be determined specifically for the planning problem to be solved. The proposed algorithm meets requirements R1-R4. A proof-of-concept implementation of the roadmap adaptation is required to confirm that ARM meets these requirements. To evaluate whether ARM meets R5, we must perform benchmarking experiments to compare ARM to conventional sampling-based algorithms.

5

Experiments & Results

This chapter describes the implementation of ARM within the Robot Operating System (ROS), experiment setups and results for all experiments performed. Experiments evaluate the effect of the number of DOFs on the planning time, compare the time spent on roadmap adaptation of the proposed ARM and its simplified implementation within ROS, perform parameter selection for sARM, evaluate the performance of sARM compared to state-of-the-art sampling-based planners, and test sARM in real-world experiments. Table 5.9 at the end of this chapter presents the experiment setups.

5.1. Implementation ARM within the Robot Operating System

We integrate the proposed algorithm in the Robot Operating System (ROS) [2] to allow using the algorithm for robots in simulation and real-world. ROS is an open-source collection of frameworks useful for the development of robot software. ROS supports collaboration among researchers all around the world in the robotics community. For the implementation of the planning algorithm within ROS, we use the Extensible Optimization Toolset (EXOTica) [19]. EXOTica is an interface for setting motion planning problems and solvers. This enables us to set our algorithm, created in the Open Motion Planning Library (OMPL) [53], as a solver in EXOTica.

5.1.1. Open Motion Planning Library

We created an OMPL planner that adapts the roadmap. OMPL is a library that contains state-of-the-art sampling-based motion planners written in C++. This library exists merely of planning algorithms and not of collision checkers, a representation of $\mathcal{W}$ or a form of visualisation, which provides the freedom to set these separately. These elements are considered black boxes for OMPL. OMPL is easily integrable into ROS with other software packages, such as EXOTica and MoveIt [10]. We use OMPL due to its existing state-of-the-art sampling-based planners that allow benchmarking and because it facilitates the addition of a new sampling-based planning algorithm to the library.

In an sampling-based planner in OMPL, a query is presented, and the planner is aware of certain space information, such as the number of robot DOFs and the optimisation objective. The sampling-based planner aims to find a collision-free path for the query. The collision checking module is set externally.

5.1.2. Extensible Optimization Toolset

We use EXOTica to set the planning problem by specifying the robot, the environment, and the collision checking approach. Additionally, we set our OMPL planner to solve the motion planning problem. EXOTica is an interface of software tools created for developing and evaluating motion synthesis algorithms within ROS. We use EXOTica because it has a more generic and open architecture than the commonly used toolbox MoveIt, and therefore, allows for fast prototyping for research purposes.

In EXOTica, the environment is set by its size and the additional models that may interact with the robot. This information is used by the collision checking module, which the user can set. The input of this module is a configuration, and the output is whether it is valid or not. Figure 5.1 presents a flow diagram of the structure of EXOTica and the implementation of an OMPL planner. The collision checker and planning scene contents are considered a black box for the planner in OMPL.

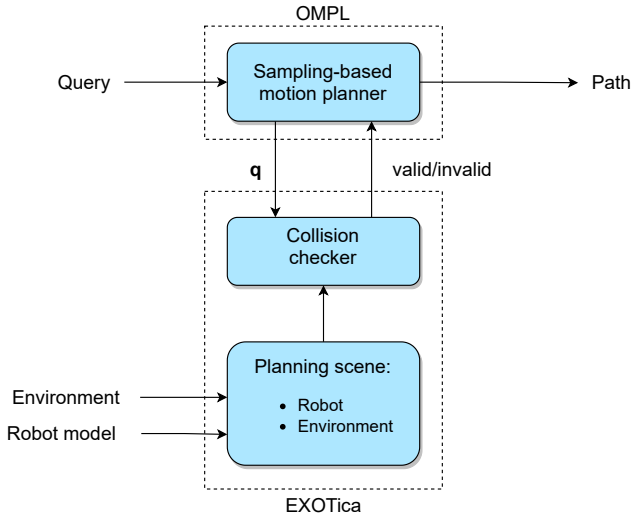


Figure 5.1: This flow diagram describes the structure of a planning problem in the Extensible Optimization Toolset (EXOTica), where an Open Motion Planning Library (OMPL) algorithm solves the planning problem. Within EXOTica, the planning scene is set, which consists of the environment and the robot. EXOTica uses this planning scene in the collision checker, which the user can set. A sampling-based motion planner from OMPL is set to solve the problem and find a path. A robot can execute this path in simulation or real-world. The communication between the planner in OMPL and the collision checker in EXOTica consists merely of the configuration sent to the collision checker, which returns whether the configuration is valid or not.

5.1.3. Limitations implementation using OMPL and EXOTica

Limitations due to a lack of communication between OMPL and EXOTica prevent the implementation of ARM using EXOTica. The planner in OMPL is aware of the roadmap, and the collision checker in EXOTica is aware of the grid cells and their occupancy; however, the communication between the planner and collision checker consists merely of a configuration sent to the collision checker, which returns whether it is valid or not. The collision checker in EXOTica can perform the assignment of configurations sent by the planner to grid cells. However, this information is not available to the planner in OMPL and cannot be used by the planner to identify invalid vertices and edges due to a change, to initiate roadmap adaptation.

We establish three approaches to make the 3D grid information available for the planner for implementation of ARM in ROS using OMPL and EXOTica:

- Hard-code the initial environment and robot model in the planner in OMPL and perform the cell assignment of configurations as well in the collision checker in EXOTica as in the planner in OMPL. Besides performing the same assignment double, which slows down the planning time and makes the planner inapplicable if the initial environment or robot changes.

- Use a lookup table that consists of configurations and associated grid cells. The collision checker appends this lookup table in EXOTica for every configuration it receives from OMPL. Besides the state validity for collision checking, an additional information stream is necessary to access this lookup table from OMPL. Additionally, we need the updated occupancy of the 3D grid in the planner, which allows the planner to look up which configurations are associated with the changed grid cells. For every call to the collision checker, which is done often for checking candidate vertices and edges [23], the lookup table is appended because the probability that the exact configuration as a random sample is already in the lookup table is zero.

- Use a lookup table, as in the second approach, that is not appended for every collision check. If configurations are close to each other, we can assume that these are associated with the same grid cells. For this implementation, interpolation is necessary to assign configurations that are not in the lookup table to grid cells. This interpolation prevents the size of the lookup table from infinitely being increased with new configuration-grid cell pairs. The lookup table must consist of a sufficient number of configurations and assigned grid cells to ensure a successful assignment. We suggest implementing supervised learning to learn the lookup table from training data to ensure the most informative grid cell-configuration pairs are added. We suggest performing this learning before the motion planning to

prevent continuous information streams between OMPL and EXOTica. Additionally, we must implement an information stream with the updated occupancy of the 3D grid to the planner. Due to time constraints, we suggest implementing learning to create a lookup table for future work.

We did not implement these three approaches to overcome the lack of communication regarding the grid cell assignment of vertices and edges due to the discussed limitations. Therefore, we suggest a simplified ARM algorithm to enable ARM implementation in ROS using OMPL and EXOTica.

Additionally, we provide a proof-of-concept implementation of the assignment and lookup of vertices and edges using the 3D grid cells and roadmap adaptation by the non-simplified ARM. This implementation can not be used as a planner.

5.1.4. Simplified adaptive roadmap algorithm

We implement a simplified version of ARM (sARM) using OMPL and EXOTica to allow implementation in ROS. sARM does not perform the initial assignment of vertices and edges to grid cells because the planner in OMPL can not use this information. Therefore, we implemented a slightly different approach to identify the invalid vertices and edges before the roadmap adaptation. Instead of instantly determining the invalid vertices and edges based on the initial assignment of vertices and edges to grid cells (Section 4.4.2), sARM checks all vertices for validity for an incremental change. Checking all vertices does not drive the planning time because the number of collision checks performed by checking all vertices of the roadmap is around 600-1000 times less compared to constructing a new roadmap of the same size with PRM [23]. The 3D grid generation, updating the occupancy of this grid, representation of configurations in $\mathcal{W}$ and the adaptation of the vertices are equal to the proposed ARM (Section 4.4.1).

Figure 5.2 presents a flow diagram of sARM. To summarise, sARM generates a 3D grid, where grid cells represent the occupancy of the environment. For every change of the occupancy of one or multiple grid cells, the roadmap adaptation is initiated. The algorithm checks all vertices for validity to determine which to adapt. The roadmap adaptation consists of removing invalid vertices and connected edges, replacing them by resampling and connecting them to the roadmap by creating edges. sARM repeats this procedure for every change in occupancy of grid cells in the 3D grid. The adapted roadmap $R_a(V,E)$ can be extracted to perform a graph search and find a path.

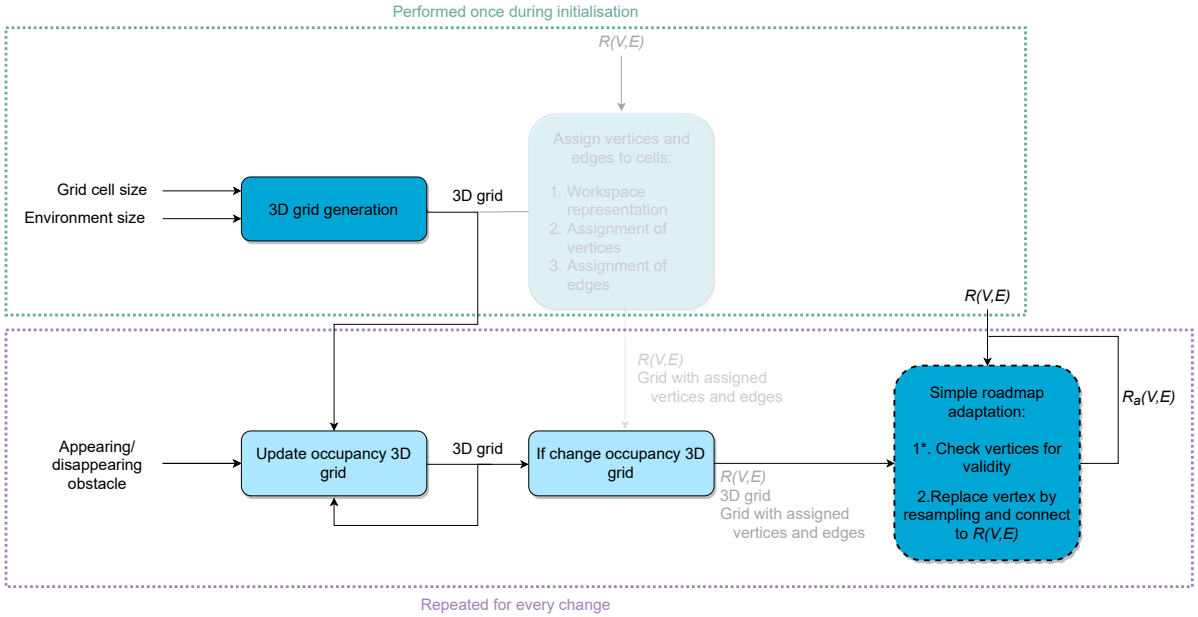


Figure 5.2: This flow diagram describes the simplified adaptive roadmap algorithm (sARM), which enables the implementation in the Robot Operating System for this thesis. This algorithm is slightly different compared to the proposed adaptive roadmap algorithm (ARM) in Figure 4.1. We visualise the elements that are not in sARM, but are in ARM transparent, and the step that is altered with a dashed border. The initialisation (green box) consists of the 3D grid generation. For every incremental change, the occupancy of the 3D grid is updated and the roadmap adaptation of the invalid vertices is initiated (purple box). sARM checks all vertices for validity to determine which to adapt because the initial assignment of vertices and edges to grid cells is not performed. The adapted roadmap $R_a(V,E)$ can be extracted to perform a graph search and find a path. We discuss the bright blue step in the flow chart in Section 5.1.4

We elaborate on the roadmap adaptation, indicated in bright blue in Figure 5.2, since the remaining steps

are equal to ARM and discussed in Section 4.4. Algorithm 4 presents the roadmap adaptation performed by sARM, which is slightly different from the adaptation of ARM, described in Algorithm 3: the identification of invalid vertices is altered, and lines 29-31 of Algorithm 3 are removed. As the roadmap adaptation is non-divergent from the adaptation in lines 5-30 in Algorithm 3, we refer to these lines in Algorithm 4. The two steps of the roadmap adaptation of this algorithm are:

1*. **Check all vertices for validity**: for every change in the environment (line 2). sARM checks all vertices $v \in V$ for validity using the collision checker set in EXOTica, which assigns a configuration to grid cells, to identify invalid vertices due to the incremental change. If the set $IQ_{\mathbf{q}}$, the indices of the grid cells associated with the configuration representing the vertex, intersects with the set IO, the indices associated with occupied grid cells (line 4), the configuration is invalid: $\mathbf{q}_{\text{inv}}$. If the corresponding grid cells are free, the configuration is valid. This step is divergent from step 1 of the roadmap adaptation in Section 4.4.3, hence the *, because the initial assignment of vertices and edges to grid cells is not performed, disabling the quick lookup by the mapping M.

2. **Replace vertex by resampling and connect to *R(V,E)***: see step 2 of the roadmap adaptation in the proposed ARM (Section 4.4.3) and lines 5-30 in Algorithm 3.

Algorithm 4 Roadmap adaptation by sARM

Input: $R(V,E)$: Initial roadmap
IO: occupied grid cells
k: number of closest neighbours to examine for configuration
d: resampling area that defines the bounds on $\mathcal{C}_{\text{sub}}$
Output: $R_a(V,E)$: Adapted roadmap

```
1: if change in IO then
2:   for all v ∈ V do                                   ▷ Section 5.1.4 item 1*
3:     q ← v
4:     if IQ_q ∩ IO ≠ ∅ then
5:       q_inv ← q
6-31:      Algorithm 3: lines 5-30
32:    end if
33:  end for
34: end if
35: return R_a(V,E)
```

The edges that become invalid due to the incremental change, of which the source and target vertex are valid, are not adapted by sARM as is done in step 3 of ARM (Section 4.4.3). Checking all edges for validity, as is done with the vertices, is costly due to the interpolation of edges for collision checking.

5.1.5. Technical details on the implementation

We add a new planner to the planners in OMPL that can locally adapt the roadmap. The implementation of the planner is based on the implementation of PRM in OMPL. The roadmap in OMPL is an object of the class ompl::base::PlannerData, which inherits from the graph structure adjacency_list of the Boost Graph Library [49, 50]. The planner in OMPL requires an initial roadmap, which it retains valid by checking whether the vertices are valid if the algorithm is called and adapting the invalid vertices, as is described in 5.1.4. If sARM successfully adapts the roadmap to a change in the environment, a graph search with A* provides the path for the query, which is an optimal and efficient algorithm that bases its cost value on the travelled distance and a heuristic based on the distance to the goal [46].

The planning problem is set in EXOTica, which implements the planner from OMPL as the solver. Within the planning problem, EXOTica loads the robot from a Unified Robot Description Format (URDF) file, which contains information about the kinematic structure, and a Semantic Robot Description Format (SRDF) file, which contains semantic information such as joint groups or additional transforms. EXOTica loads the obstacles in the environment from a .scene file, in which the obstacle locations, sizes and shapes are specified. One can specify additional parameters, such as the environment bounds and joint groups to plan for, in the planning problem in EXOTica. Lastly, we set the constraints for the planning problem in a task map that maps $\mathcal{C}$ to the task space and can be loaded as a plugin in EXOTica; an example of a task map is a collision checking module. The task map for sARM checks for collision with obstacles, based on the occupancy of the 3D

grid, and additionally checks adherence to the joint limits and whether the robot is not in self-collision. The task map creates the 3D grid in the initialisation and sets the grid cells associated with the static obstacles to occupied. If obstacles appear or disappear, the occupancy of grid cells associated with the obstacle is updated. For every configuration the task map receives from OMPL for collision checking receives, it performs the following steps:

1. Check whether the joint limits are satisfied.
2. Check whether the configuration is not in self-collision.
3. Assign configurations to grid cells based on their representation in $\mathcal{W}$. Check whether the associated grid cells are free.

The task map returns the sample is valid to OMPL if all steps are satisfied. If any step is not satisfied, the sample is invalid and the successive steps are not performed.

The planning problem in EXOTica can be altered, by changing the robot, obstacles or task map specific parameters, such as the 3D grid cell size, which makes this implementation of sARM suitable for various problems.

5.1.6. Discussion

No instant determination of the vertices and edges associated with a grid cell is performed in sARM due to the lack of communication between the planner in OMPL and the collision checker in EXOTica. We simplify ARM to sARM to enable implementation of the planner in ROS and use this planner to evaluate the effect of locally adapting the roadmap on the planning time for mobile manipulators. However, it must be noted that in the implemented algorithm, the roadmap is not entirely in $\mathcal{C}_{\text{free}}$, because the edges associated with a newly occupied grid cell, of which the source and target vertices are in free grid cells, are not adapted. For future research, we suggest implementing the non-simplified ARM as a planner, which additionally ensures all edges are in $\mathcal{C}_{\text{free}}$ and enables quick lookup of vertices and edges associated with an incremental change. In this thesis, we provide a proof-of-concept implementation of the assignment and lookup of vertices and edges using the 3D grid cells and roadmap adaptation by the non-simplified ARM.

5.2. Planning scenarios

Before presenting the experiment setups, we discuss details regarding the problem and planner settings referred to in the experiments.

5.2.1. Planners

In the experiments, we implement two state-of-the-art sampling-based motion planners: RRT (Section 2.5) and PRM (Section 2.6). RRT represents the single-query for comparison, and PRM the multi-query algorithm. The third planner referred to in these experiments is the proposed algorithm: sARM. The planner-specific settings are set for every experiment individually.

- **RRT:** the implementation of RRT in OMPL [53] is used for the experiments. Settings of RRT that can be specified in OMPL are the range parameter, which is the maximum length of an edge to be added in the tree, and the goal bias, introduced in Section 2.5. OMPL suggests to set the goal bias to 0.05, which we do for the experiments in this chapter. The range parameter is set to 15m, determined with the experiment in Appendix A.1.
- **PRM:** the implementation of PRM in OMPL [53] is used for the experiments. The implementation of PRM in OMPL adds $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ to the roadmap as first configurations in the roadmap construction step and if they are connected, the solution is returned. This is slightly different from the description of PRM in Section 2.6, where first, the entire roadmap is constructed, and then, $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are connected. The setting for PRM that can be specified in OMPL is the number of nearest neighbours, introduced in Section 2.6. We set the number of nearest neighbours parameter to 10, determined with the experiment in Appendix A.2.
- **sARM:** we use sARM (Section 5.1.4), which we implemented in OMPL, for the experiments. The initial roadmaps for sARM in these experiments are generated with PRM in the corresponding environment. As ARM uses an equivalent connection strategy for a new vertex to nearest neighbours as PRM, we set

the number of nearest neighbours parameter to 10, which resulted in a good performance for PRM in Appendix A.2.

5.2.2. Robots

We plan for two robots in the experiments: a 3-DOF mobile robot and a 10-DOF mobile manipulator. Both robots consist of the same mobile robot, which is considered nonholonomic due to the wheels underneath the robot. To adhere to these constraints, Reeds-Shepp Curves describe the path between two vertices. We set the minimum turning radius of the robots in these experiments to 0.2m. The distance function for the DOFs represented in $\mathbb{R}$ are computed with Equation (2.6), and for the DOFs represented in $\mathbb{S}$ with Equation (2.7).

- **3-DOF mobile robot**: the mobile robot is the Boxer indoor mobile robot developed by Clearpath, which has three DOFs: translation in the 2D plane and yaw: $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$, with a fixed top mount. Figure 5.3 presents the properties of the mobile robot, which is the base of the visualised mobile manipulator. More technical details concerning the mobile base are provided by Clearpath [9].
- **10-DOF mobile manipulator:** the mobile manipulator consists of a base, which is the previously introduced 3-DOF Clearpath mobile robot, and a manipulator on top, which is the 7-DOF PANDA robotic arm developed by Franka Emika [13]. Therefore, this robot has 10-DOF in total, with $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S} \times \mathbb{R}^7$, because all arm joints have limits. We perform coupled motion planning for the base and arm of the mobile manipulator. Figure 5.3 presents an image of the mobile manipulator with indicated properties. More technical details concerning the robotic arm are provided by Franka Emika [13].

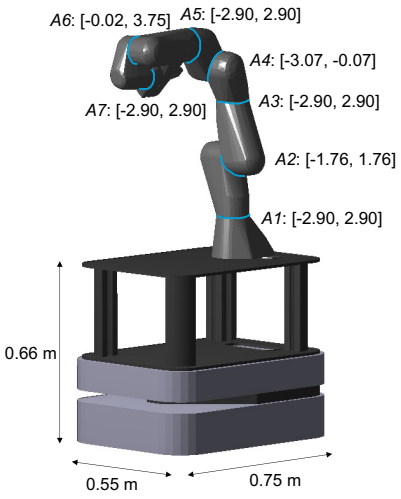


Figure 5.3: This figure presents the properties of the 10 degrees of freedom (DOFs) mobile manipulator, which includes the 3-DOF mobile robot as base. The 7 DOFs of the robot arm, $A1$ to $A7$, rotating at junctures highlighted with blue lines, are indicated with the corresponding joint limits (in radians).

5.2.3. Initial environment

In the experiments, we plan in four 3D environments: an environment without obstacles, an environment with three obstacles, a supermarket environment with ten obstacles, and a partially lowered supermarket environment with eight obstacles. All environments have a ground plane of equal size ($20x20$m) and a height of 2m because the robots can not extend higher than 2m. Figure 5.4 presents the environments.

- **Zero-obstacle environment:** every configuration is valid if the joint and environment limits are satisfied.
- **Three-obstacle environment:** see Figure 5.4a.
- **Ten-obstacle environment:** representing a supermarket, see Figure 5.4b. We use this environment because a supermarket is an incrementally changing environment, as was explained in Chapter 1.

- **Eight-obstacle environment:** representing a partially lowered supermarket environment, see Figure 5.4c where the robot must bend the arm to go under the obstacles in the air and can not go around. Environments where the robot has to fold the arm to pass by an obstacle require the flexibility from coupled motion planning. The robot cannot pass by this obstacle if the planning for the base and arm is decoupled.

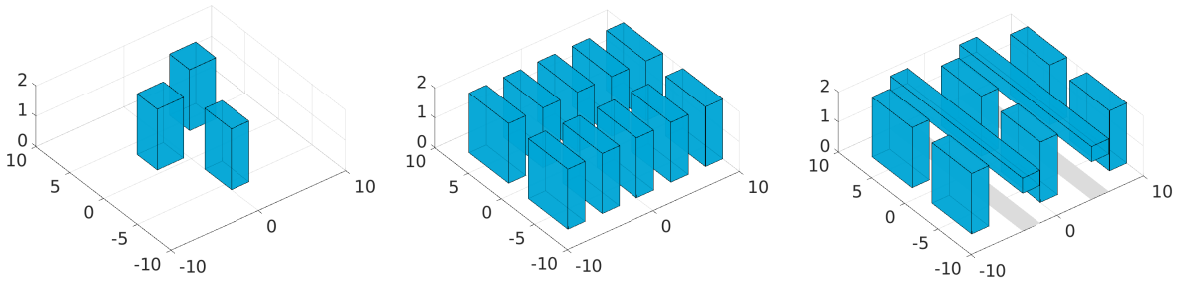


Figure 5.4: These figures present the initial environments of equal size ($20x20x2$m) with obstacles for the experiments, with (a) three obstacles, (b) ten obstacles, representing a supermarket with the smallest isle width of 2m and (c) eight obstacles, representing a partially lowered supermarket environment, where we replaced four obstacles from the supermarket environment with two obstacles in the air with a lower bound of $z = 1.25$m.

5.2.4. Incremental changes

In the experiments for assessing the performance of sARM, the planning environments are incrementally changing. We add one obstacle in an unoccupied space for every new query and remove the previously added obstacle while the initial environment remains constant. Adding an obstacle for every query ensures equivalent behaviour as if an obstacle appears during the execution of a trajectory since both induce roadmap adaptation for sARM and reconstruction for state-of-the-art sampling-based planners.

5.2.5. Collision checking

The sampling-based planners use two collision checking approaches in the experiments: sphere-based collision checking and collision checking based on a 3D occupancy grid. Both collision checking approaches determine equivalently whether a configuration adheres to the joint limits and whether no self-collision occurs due to the configuration. These methods check differently whether a robot configuration is in-collision with an obstacle.

We created both collision checkers in EXOTica in this thesis because the standard collision checker in EXOTica [40] is time-consuming and can not solve a planning problem in practical time bounds. This approach represents the robot by a triangle mesh and checks for every triangle whether it is in contact with remaining objects. This is time-consuming as many triangles represent the robot, resulting in many checks are performed for every configuration. The standard collision checker can not solve one query for the 10-DOF mobile manipulator in the ten-obstacle environment of Figure 5.4b within one hour, using RRT and PRM with various settings. If we represent the robot by primitive shapes, such as spheres and boxes, less collision checks are necessary. For instance, if the robot is represented by a box, and checks whether it is in collision with a box-shaped obstacle, merely four contacts have to be checked to determine whether the boxes overlap.

- **Sphere-based collision checking:** represent the robot and the environment by englobing spheres; this approach is widely used for collision checking, for example, in [47, 62]. A robot sphere is denoted by a and an obstacle sphere by b. Consider the robot spheres of a configuration represented by the union $SA_{\mathbf{q}}$, and the obstacle spheres by the union SB:

$$SA_{\mathbf{q}} = \bigcup_{i=1}^{n_a} a_i \;\; \& \;\; SB = \bigcup_{j=1}^{n_b} b_i$$

  Where n_a and n_b are the number of robot and obstacle spheres, respectively. A non-linear constraint for collision avoidance is set, ensuring that the spheres englobing the robot do not intersect with the

spheres englobing the obstacles:

$$SA_{\mathbf{q}} \cap SB = \emptyset,$$

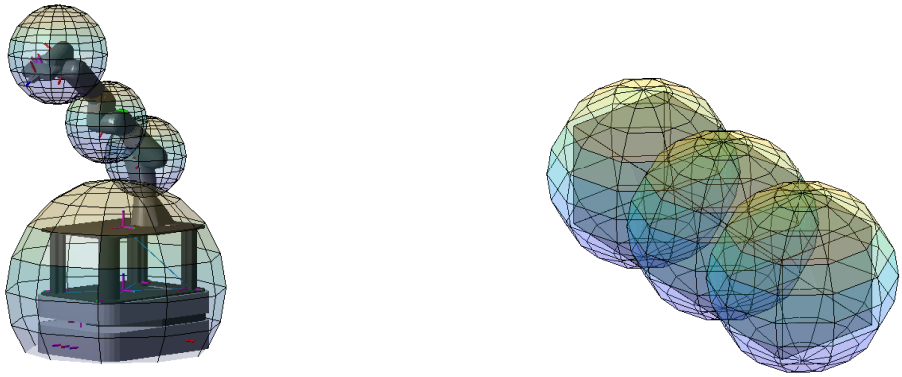
imposing that the distance between the spheres is larger than the sum of their radii:

$$\left\| \mathbf{p}_i - \mathbf{p}_j \right\| \geq r_i + r_j \quad \forall i \in 1, ..., n_a, \quad j \in 1, ..., n_b \tag{5.1}$$

Where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the centre positions of the i-th robot sphere and j-th obstacle sphere, respectively, and r_i and r_j are the radii of the i-th robot sphere and j-th obstacle sphere, respectively. Figure 5.5 presents an englobed mobile manipulator and an englobed obstacle.

The number of spheres representing the robot must be chosen carefully. More spheres represent the robot more accurately, which results in less conservative collision checking: where robot configurations are invalid while the englobing sphere is in-collision, but the robot is not. However, more computations of Equation (5.1) are necessary if the robot is represented by more spheres, increasing the planning time.

- **Collision checking based on a 3D occupancy grid:** represents the environment with a 3D grid as in Section 4.4.1. This collision checking approach represents a configuration in $\mathcal{W}$ by a bounding box and is assigned to grid cells, equivalent as we discussed in step 1 and 2 in Section 4.4.2. If an obstacle is associated with an assigned grid cell, the configuration is invalid.



(a) (b)

Figure 5.5: These figures visualise the englobing spheres of (a) the 10 degrees of freedom (DOFs) mobile manipulator and (b) an obstacle in sphere-based collision checking. If the robot spheres do not intersect with obstacle spheres, the robot configuration is in $\mathcal{C}_{\text{free}}$.

The self-collision check used in all experiments is from the standard collision checker in EXOTica [40] based on triangular meshes. The self-collision check does not check a link with all remaining robot links for collision, as it is specified that some links never collide. As the self-collision check does not check all robot links, the planning time remains within practical time bounds.

5.2.6. Queries

For the planning problems in the experiments, we generate different queries to ensure the results are not dependent on a specific query. For every new query, the new start state is the previous goal state, and the new goal state is the next valid goal state that was set. We specify the number of queries planned for in the experimental setups. The queries are environment-specific; for the zero- and three-obstacle environments, we determine them by uniform random sampling in the environment. For the ten- and eight-obstacle environments, based on supermarkets with narrow corridors, the (x, y) positions of the base for the queries are manually determined to ensure the robot can reach all corridors (see Figure 5.8). We randomly select the values for the remaining DOFs from a list of valid configurations to ensure no goal states limit the constraints or are in self-collision.

5.3. Experiment 1: Effect of robot DOFs on the planning time

We performed experiments to show the effect of the robot DOFs on the planning time for single- and multi-query algorithms in obstacle-cluttered environments, to support the relevance of the research question of this thesis.

5.3.1. Setup

We carried out experiments in simulation to compare different planners in different scenarios consisting of the various robots and environments. We set the planning problems in EXOTica [19]. We present details on the problem and planner settings in Table 5.9. Besides the planners discussed in Section 5.2.1, we included a PRM implementation that clears the roadmap after every run as a third planner to evaluate the performance of a multi-query algorithm in a changing environment, where the roadmap can not be reused. We refer to this PRM implementation as single-query PRM: PRM (SQ), where PRM (MQ) refers to the conventional multi-query implementation of PRM. We solved planning problems in different environments because the obstacles in the environment greatly influence the planning time of planning algorithms. For the sphere-based collision checking, we represented the 3-DOF robot by a single sphere englobing the entire robot, and the 10-DOF robot by four spheres for an accurate representation of the robot configuration. Table 5.1 presents the positions of the spheres on the kinematic chain and Figure 5.5a a visualisation.

We assessed the planning time for all combinations in the experiment setup. Additional metrics, such as the number of performed collision checks, number of vertices and edges of the roadmap, path length, and success rate, are additionally assessed to get a clear insight into the planning time drivers.

Object	**Parent link**	**Offset [m]**	**Radius [m]**
3-DOF robot	base link	[0.3,0,0.25]	0.54
10-DOF robot	base link	[0.3,0,0.25]	0.54
	link 1	[0,0,0]	0.20
	link 5	[0,0,0]	0.20
	link 7	[0,0,0]	0.23
three-obstacle environment	obs1	[0,1,0]	1.5
	obs1	[0,-1,0]	1.5
	obs2-obs3	[0,0,0]	2
ten-obstacle environment	obs1-obs10	[0,2,0]	1.35
	obs1-obs10	[0,0,0]	1.35
	obs1-obs10	[0,-2,0]	1.35

Table 5.1: This table visualises the positions of the spheres for sphere-based collision checking for the robots (Section 5.2.2) and two environments with static obstacles (Section 5.2.3) for Experiment 1 (Section 5.3). The centre of the sphere is at the offset from the parent link. We represented the rectangular obstacle (obs1) in the three-obstacle environment by two spheres and the two square obstacles (ob2-obs3) by a single sphere. In the ten-obstacle environment, all obstacles (obs1-obs10) were represented by three spheres.

5.3.2. Results

From the results presented in Figure 5.6, it is clear that as the number of obstacles increased, the planning time increased. Table 5.2 presents the results concerning the mean planning time, number of vertices, number of edges, number of collision checks, path length, and success rate. PRM (MQ) had the lowest planning time for both robots in the environments with three and ten obstacles. PRM (SQ) had the lowest planning time for the 3-DOF and 10-DOF robots in the empty environment.

5.3.3. Discussion

As the number of obstacles increased, all algorithms performed more collision checks on candidate vertices and edges that are invalid and, therefore, not added to the roadmap, specifically for the 10-DOF robot due to collision of the arm. Collision checking is the main drive of sampling-based planners [39], and is therefore considered the principal reason for the increase in planning time if the number of obstacles increases.

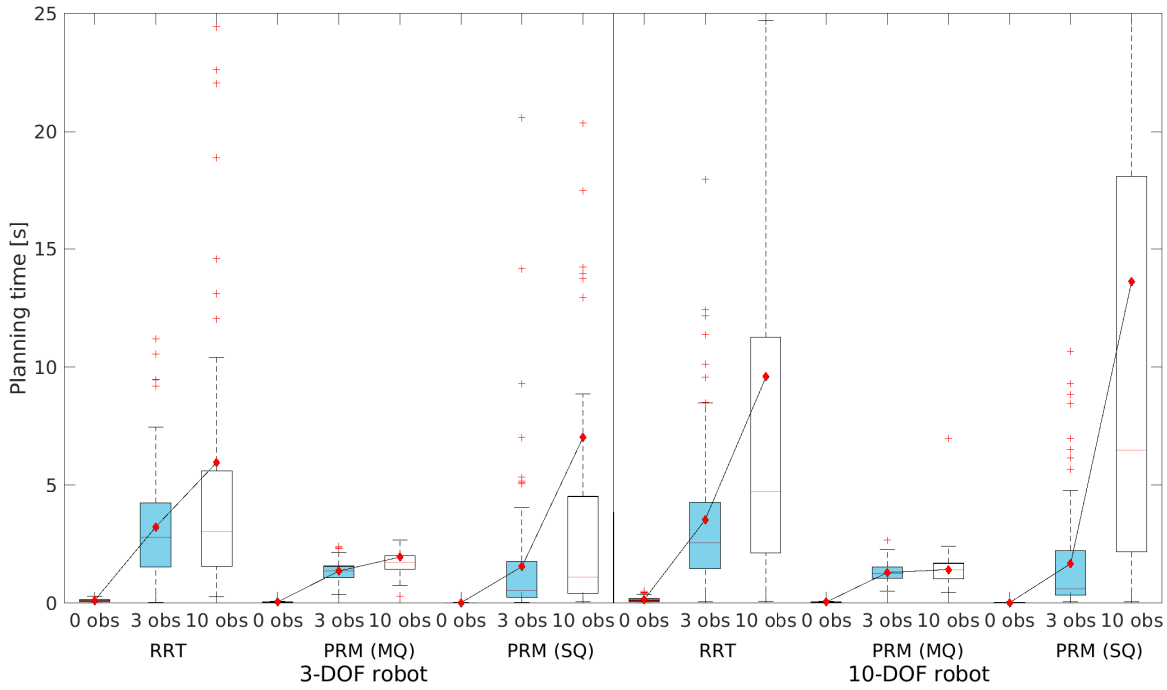


Figure 5.6: This boxplot presents the results of Experiment 1 (Section 5.3) that assessed the effect of the robot degrees of freedom (DOFs) on the planning time for hundred queries for a 3-DOF robot and a 10-DOF robot in different environments. 0 obs, 3 obs and 10 obs refer to the environments in Section 5.2.3. The black lines connect the mean planning times, indicated with red diamonds. The value of the upper adjacent of PRM (SQ) in the ten-obstacle environment is 41.5s, which is cut off in this visualisation to prevent elongation of the y-axis. PRM (SQ) indicates an implementation of PRM that clears the roadmap after every query. The experiment setup is presented in Table 5.9

	Planner	Robot DOFs	Planning time [s]	Number of vertices	Number of edges	Number of collision checks	Path length [m]	Success
			Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	
0 obstacles	RRT	3	0.12 ± 0.07	26.3 ± 16.0	26.3 ± 16.0	-	19.8 ± 8.1	100%
		10	0.15 ± 0.11	28.0 ± 20.6	28.0 ± 20.6	-	30.9 ±15.2	100%
	PRM (MQ)	3	0.05 ± 0.01	99.1 ± 57.9	113.5 ± 71.8	-	15.4 ± 5.9	100%
		10	0.05 ± 0.01	101.0 ± 58.0	117.0 ± 70.2	-	20.6 ± 8.1	100%
	PRM (SQ)	3	0.02 ± 0.00	2.0 ± 0.0	1.0 ± 0.0	-	13.2 ± 5.4	100%
		10	0.02 ± 0.00	2.0 ± 0.0	1.0 ± 0.0	-	15.1 ± 5.6	100%
3 obstacles	RRT	3	3.23 ± 2.41	22.2 ± 15.7	22.2 ± 15.7	1125.5 ± 843.8	19.6 ± 10.3	100%
		10	3.52 ± 3.16	22.9 ± 19.2	22.9 ± 19.2	1216.3 ± 1082.4	30.9 ± 17.6	100%
	PRM (MQ)	3	1.36 ± 0.38	105.1 ± 57.9	104.2 ± 69.7	490.8 ± 137.6	17.4 ± 8.5	100%
		10	1.30 ± 0.39	102.0 ± 58.0	98.7 ± 65.7	442.4 ± 124.9	23.5 ± 10.6	100%
	PRM (SQ)	3	1.55 ± 2.82	8.1 ± 10.1	5.4 ± 9.9	563.9 ± 1027.1	18.2 ± 11.7	100%
		10	1.67 ± 2.19	8.1 ± 7.6	5.6 ± 6.8	566.8 ± 746.7	27.2 ± 19.2	100%
10 obstacles	RRT	3	5.96 ± 10.91	41.1 ± 79.4	41.1 ± 79.4	2033.6 ± 3725.8	21.3 ± 9.5	100%
		10	9.60 ± 14.11	53.7 ± 83.2	53.7 ± 83.2	3027.9 ± 4492.7	39.7 ± 19.9	100%
	PRM (MQ)	3	4.33 ± 16.89	841.3 ± 195.6	1002.3 ± 235.1	1495.9 ± 5860.7	17.3 ± 6.6	100%
		10	3.79 ± 16.71	549.5 ± 166.0	532.1 ± 175.3	1213.4 ± 5329.7	39.9 ±14.5	98%
	PRM (SQ)	3	7.02 ± 15.69	24.5 ± 48.7	24.7 ± 57.5	2445.4 ± 5473.7	22.3 ± 14.8	98%
		10	14.69 ± 19.81	58.3 ± 71.5	48.4 ± 68.3	4629.4 ± 6230.9	47.4 ± 25.3	99%

Table 5.2: This table presents the results of Experiment 1 (Section 5.3) evaluating the effect of the robot DOFs on the planning time for hundred queries, concerning the mean and standard deviations of the planning time, number of vertices, edges, collision checks, path length, and success rates for the three planners. PRM (SQ) is an implementation of PRM that clears the roadmap after every query. We present the experiment setup in Table 5.9

PRM (MQ) had the lowest planning time for the 10-DOF robot in the three- and ten-obstacle environment because the time-consuming graph construction due to the obstacles is performed once. After all, the graph is reusable. However, if PRM planned in a dynamic environment and has to construct a new roadmap for every query (PRM (SQ)), the planning time was four times the planning time if the roadmap is reused (PRM

(MQ)) in the ten-obstacle environment. For the zero-obstacle environment, the planning time of PRM (SQ) was lowest due to the implementation of PRM in OMPL, where before the graph construction, $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are added to the roadmap. In the zero-obstacle environment, this resulted in directly connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ and returning the solution faster than if samples are drawn in RRT or $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are connected to the roadmap, and a path is searched in PRM (MQ).

It must be noted that the self-collision check took up 90% of the time spend on a single collision check due its costly approach (Section 5.2.5). As this is equivalent for all planning problems in all experiments, this does not affect the comparisons in this thesis. The planning times can be decreased by implementing a custom self-collision checker based on primitive shapes.

5.3.4. Conclusion

This experiment confirmed that the planning time is lowest for environments with obstacles when reusing the roadmap, specifically for a high-DOF robot. The large planning time for PRM (SQ) due to the reconstruction of the roadmap supports the use of single-query algorithms, such as RRT, in dynamic environments. More importantly, it uncovers the potential of adapting the roadmap locally to changes, which is favourable over single-query methods if the roadmap adaptation takes less time than solving the problem with a single-query algorithm.

5.4. Experiment 2: Comparison of the time spent on roadmap adaptation by ARM and sARM

In this experiment, we compared the time spent on the roadmap adaptation by ARM and sARM to gain insight into how the algorithm's planning time would change if the non-simplified algorithm is implemented as a planner. We focused on comparing the roadmap adaptation and not on the overall planning time. As we do not solve planning problems in this experiment, we do not use EXOTica. In further experiments, sARM is implemented as a planner, and we evaluate its performance. Even though we do not deploy the planners in this experiment, we refer to the methods of roadmap adaptation by ARM and sARM.

5.4.1. Setup

We used the classes from OMPL for the implementation of the roadmap adaptation by sARM and ARM. Table 5.9 presents the experimental details. We did not set the number of queries, maximum allowed time, and graph search algorithm as we do not solve planning problems.

We run the adaptation by ARM and sARM for the same initial environment where we add for every run an incremental change of $1 \times 1 \times 1$m such that space for the robot to pass by is left (see Figure 5.7). The initial roadmap is equal for both roadmap adaptation approaches. We evaluated the roadmap adaptation for fifty incremental changes.

For ARM and sARM we assessed the time spent on: identification of the invalid vertices of the roadmap, resampling a vertex and connecting the resampled vertex to the roadmap. For ARM, we additionally assessed the time spent on the initial assignment of vertices and edges and the time spent on identifying the invalid edges and removing them, as sARM does not identify and remove edges of which the source or target vertex is not in collision. We divided the time for replacing vertices, connecting them to the roadmap and removing invalid edges by the number of vertex or edge adaptations the algorithms performed to provide a straightforward comparison.

5.4.2. Results

ARM and sARM resampled on average 2.4 ± 2.8 and 2.5 ± 2.6 vertices, respectively, and ARM removed 11.2 ± 9.7 edges for each incremental change. ARM and sARM spent 7.57 ± 8.18 and 8.61 ± 8.52 seconds on the roadmap adaptation due to one incremental change, respectively; the roadmap adaptation is 10% faster by ARM than by sARM. For ARM, this excludes the 10.91 ± 0.29 seconds spent on the initial assignment of vertices and edges, which it performs once. sARM does not perform this initial assignment. Table 5.3 presents a specification of the time spent on the roadmap adaptation for ARM and sARM. ARM identifies the invalid vertices of the roadmap due to an incremental change 1000x faster than sARM.

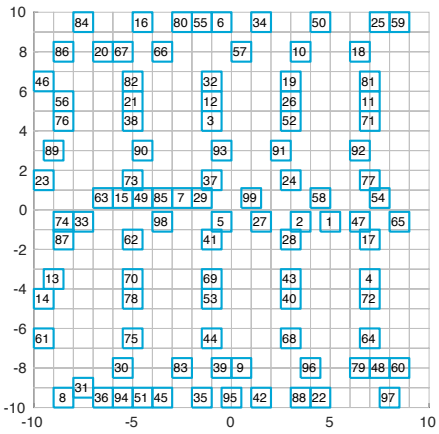


Figure 5.7: This figure presents the 2D projection of the incremental changes (Section 5.2.4) used in the experiments in the ten- and eight-obstacle environment (Section 5.2.3). The obstacles are $1 \times 1 \times 1$m (blue) and we present the order the obstacles are added and removed by the numbering. We project this to 2D for clarity. The obstacles are at the ground, 0.5m, 1m or 1.5m above the ground.

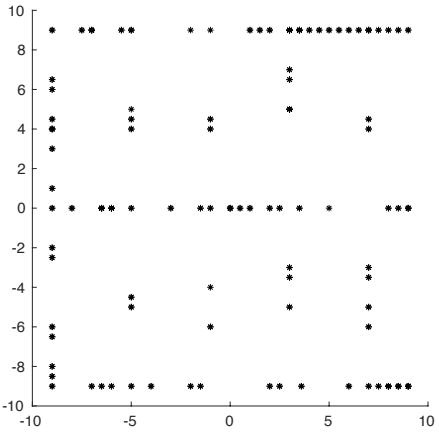


Figure 5.8: This figure presents the (x, y) position of the base of goal configurations in the ten- and eight-obstacle environments (Section 5.2.3) used in the experiments. For every new query, the start is the previous goal configuration.

	ARM	**sARM**
Update occupancy 3D grid [s]	$1.75 \times 10^{-4} \pm 8.37 \times 10^{-6}$	$3.26 \times 10^{-4} \pm 2.79 \times 10^{-5}$
Identify invalid vertices of the roadmap [s]	$8.55 \times 10^{-4} \pm 5.24 \times 10^{-5}$	0.66 ± 0.016
Identify invalid edges of the roadmap [s]	$1.04 \times 10^{-3} \pm 1.53 \times 10^{-3}$	-
Find replacing vertex (per vertex) [s]	2.80 ± 0.94	3.04 ± 1.21
Connect replacing vertex to the roadmap (per vertex) [s]	0.45 ± 0.16	0.44 ± 0.17
Remove invalid edges (per edge) [s]	$1.46 \times 10^{-6} \pm 1.63 \times 10^{-6}$	-

Table 5.3: This table presents the results of Experiment 2 (Section 5.4) that compares the time spent on roadmap adaptation by the proposed adaptive roadmap algorithm (ARM) and the simplified implementation (sARM) to gain an insight in how the planning time of the algorithm would change if the non-simplified algorithm is implemented as a planner. The time is indicated by the mean ± the standard deviation. We present the experiment setup in Table 5.9

5.4.3. Discussion

ARM and sARM perform a similar number of vertex adaptations. Additionally, the time spent on finding a replacing vertex and connecting it to the roadmap is similar as they perform identical steps for the vertex adaptation. The time required to find a replacing vertex is the primary driver of the roadmap adaptation time for ARM and sARM as many randomly drawn samples to replace the vertex are invalid due to the narrow corridors in the initial environment and the forced resampling in the neighbourhood of the invalid vertex, and thus the incremental obstacle, to maintain the connectivity of the graph.

The approaches differ by the time spent on the identification of the invalid vertices. ARM identifies the invalid vertices 1000x faster as it uses the mapping from grid cells to vertices for a quick lookup of the vertices associated with the incremental change while sARM checks all vertices for validity due to an incremental change. To obtain this mapping, ARM invests time, approximately 11s, to assign all vertices and edges to grid cells. As the difference in roadmap adaptation time for every incremental change is approximately 1s in favour of ARM, the time investment of the initial assignment of vertices and edges was returned after eleven incremental changes.

Additionally, ARM ensures that the entire roadmap is in $\mathcal{C}_{free}$, where sARM merely updates the invalid edges of which the source or target vertex is invalid. ARM removed approximately 11 edges for every incremental change, indicating that after every incremental change the roadmap is adapted by sARM, 11 invalid edges remain in the roadmap. This highlights the importance of adapting edges of which the source and target vertex are valid, but the edge itself is not. If the initial roadmap consists of short edges with respect to the obstacle size, this will not occur because the source or target vertex will also be invalid. The identifica-

tion of the invalid edges by sARM would be time-consuming as the initial assignment of vertices and edges can not be used when solving planning problems using EXOTica as was explained in Section 5.1.4. To solve planning problems with EXOTica, which enables the planner's implementation within ROS, we use sARM. We perform the remaining experiments in this chapter with sARM. However, for roadmap adaptation that ensures the entire roadmap is in $\mathcal{C}_{\text{free}}$, and for a speedup of the adaptation once the time investment of the initial assignment is returned, implementing ARM within ROS is suggested for future work.

5.4.4. Conclusion

In this experiment, the roadmap adaptation due to an incremental change is 10% faster by ARM than by sARM as a quick lookup of invalid vertices and edges is enabled. ARM invests time in the initial assignment of vertices and edges to the grid cells to allow this quick lookup. In this experiment, the invested time is returned after approximately 11 incremental changes.

5.5. Experiment 3: Effect of the resampling area on the adaptation time of sARM

In this experiment, we determined the resampling area for sARM that ensures the fastest roadmap adaptation. sARM sets a subspace of $\mathcal{C}$ is set, $\mathcal{C}_{\text{sub}}$, with (x,y) bounds determined by a shift of d from the invalid vertex to obtain a replacing vertex close to the invalid vertex (Equation (4.4)). sARM generates a new vertex in $\mathcal{C}_{\text{sub}}$ by random sampling. We assessed the time sARM spends roadmap adaptation for different values of d.

5.5.1. Setup

We carried out experiments in simulation solving motion planning problems with sARM with d = [1m,6m] in two environments. We set the planning problems in EXOTica [19]. We present details on the problem and planner settings in Table 5.9. We added an obstacle for every query and remove the previously added obstacle. We placed the obstacles in the corridors such that space for the robot to pass by is left (see Figure 5.7). The appearing obstacles were $1 \times 1 \times 1$m because this is approximately the size of incremental changes representing appearing boxes in corridors in a supermarket environment. The initial roadmap we use in these experiments was equal for all values of d. The grid cell size used for the 3D occupancy grid collision checking was $1 \times 1 \times 0.25$m for both environments because this is proportionate to the size of the appearing obstacles, does not make the narrow corridors inaccessible, and $\delta_z = 0.25$m allows the robot to bend its arm to go under the obstacles in the air. Except for the d-value, all problem and planner settings are equal.

We assessed the adaptation time, which is the planning time divided by the number of adaptations, because d affects the time necessary to perform a single adaptation. Even though the initial roadmap was equal, the number of adaptations can differ due to the randomised generation of a replacing vertex. The lowest value of d is preferred to ensure the replacing vertex is close by the invalid vertex to maintain the connectivity of the roadmap. However, this must not increase the planning time due to time-consuming resampling.

5.5.2. Results

Table 5.4 presents the adaptation time for the values of d = [1m,6m]. For the ten-obstacle environment, the results showed a similar adaptation time for d = [2m,6m]. The adaptation time for d = 1m is more than double than for the remaining d-values with a success rate of 76%, while the success rates of the remaining d-values are $98-100$%. In the eight-obstacle environment, d = [1m,6m] showed a similar adaptation time. The success rate for d = 1m is 44%, while the success rates of the remaining d-values are $98-100$%.

5.5.3. Discussion

We performed this experiment for fifty queries because too many roadmap adaptations cause the roadmap to divide into multiple connected components, which causes failure if $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are connected to a different component. When planning for fifty queries, and thus fifty appearing obstacles, no disconnected components arise for these planning problems, ensuring the results of this experiment are not affected by the roadmap getting disconnected.

Changing the $d-$value does not result in substantial differences in the adaptation time, except for d = 1m. If d = 1m, sARM resampled in a mainly occupied $\mathcal{C}_{\text{sub}}$, due to the appearing obstacle of $1 \times 1 \times 1$m, which resulted in a higher adaptation time and a lower success rate because the algorithm more often failed to find a valid replacing vertex in the allowed time.

	d [m]	Adaptation time [s]	Success
		Mean ± STD	
10 obstacles	1	9.02 ± 13.34	76%
	2	3.80 ± 1.10	100%
	3	4.08 ± 1.37	100%
	4	4.05 ± 0.92	100%
	5	4.09 ± 1.18	98%
	6	4.35 ± 1.23	100%
8 obstacles	1	4.42 ± 2.53	44%
	2	3.72 ± 1.58	100%
	3	3.60 ± 1.08	100%
	4	3.94 ± 1.25	100%
	5	4.00 ± 1.68	98%
	6	3.98 ± 1.40	100%

Table 5.4: This table presents the results of Experiment 3 (Section 5.5) for the ten-obstacle and the more challenging eight-obstacle environment for fifty queries where for every query an obstacle of $1 \times 1 \times 1$m appeared, to determine the d-value that ensures the fastest roadmap adaptation by the proposed simplified adaptive roadmap algorithm (sARM). The d-value describes the subspace of the configuration space in which samples are drawn to find a replacing sample close by the invalid sample by sARM. The adaptation time is the planning time of sARM divided by the number of performed adaptations. We present the experiment setup in Table 5.9

5.5.4. Conclusion

For both environments, we conclude sARM with $d = 2$m performs best as the adaptation time is similar to the higher d-values and the replacing vertex is sampled closer by the invalid vertex than for higher values of d, which is favorable in the interest of maintaining a connected graph.

5.6. Experiment 4: Effect of the grid cell size on the planning time of sARM

In this experiment, we evaluated which grid cell size ensured the lowest planning time for sARM. In Section 4.5 we described that the size of the 3D grid cells influences the planning time of sARM and that the grid cell size is dependent on the environment geometry and the size of the appearing obstacles.

5.6.1. Setup

We carried out experiments in simulation solving planning problems with sARM with various grid cell sizes, which we set in EXOTica [19]. We present details on the problem and planner settings in Table 5.9. We added an obstacle for every query and remove the previously added obstacle. The incremental changes were equal to Experiment 3 (Section 5.5): $1 \times 1 \times 1$m (see Figure 5.7). All planner and parameter settings were equal, except for the grid cell size. We set the value of d to 2m because this ensured the fastest roadmap adaptation in Experiment 3 (Section 5.5) in this environment.

The grid cell sizes in this experiment were $0.1 \times 0.1 \times 0.1$m, $0.25 \times 0.25 \times 0.25$m, $0.5 \times 0.5 \times 0.5$m and $1 \times 1 \times 1$m, thus one tenth, a quarter, half and the equal to the size of the incremental changes, respectively. To maintain accessibility of the corridors that are in the initial environment, the grid cell size was $1 \times 1 \times 1$m at maximum.

In this experiment, we evaluated the planning time of the implementations of sARM with different grid cell sizes.

5.6.2. Results

Table 5.5 presents the results of this experiment. The planning time was similar for the grid with cells of size $0.5 \times 0.5 \times 0.5$m and $1 \times 1 \times 1$m, slightly higher for the grid with cells of size $0.25 \times 0.25 \times 0.25$m, and more than double for the grid with cells of size $0.1 \times 0.1 \times 0.1$m. As the grid cell size increased, the number of roadmap adaptations increased, the number of collision checks increased, and the path length was higher.

5.6.3. Discussion

For sARM with larger grid cells, more vertex adaptations were initiated due to the more conservative approach of representing the occupancy of the environment by the large grid cells. The increased number of adaptations caused the increase in collision checks, which were performed to find a replacing vertex. The conservative approach also caused narrow passages getting blocked for the larger grid cells, increasing the path length.

$\delta_x = \delta_y = \delta_z$ [m]	Planning time [s]	Number of vertices	Number of edges	Number of collision checks	Path length [m]	Number of adaptations	Success
	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	
0.1	34.91 ± 24.79	160.0 ± 0.0	436.5 ± 72.85	3050.3 ± 1728.7	40.1 ± 21.2	2.2 ± 1.6	98%
0.25	14.76 ± 16.55	160.0 ± 0.0	436.5 ± 77.4	3364.9 ± 1914.6	41.5 ± 20.2	2.5 ± 1.9	100%
0.5	12.94 ± 16.67	160.0 ± 0.0	452.6 ± 85.4	3407.3 ± 2155.6	45.9 ± 24.5	2.7 ± 2.2	100%
1	12.77 ± 16.67	160.0 ± 0.0	389.0 ± 33.9	3677.9 ± 2677.3	48.8 ± 22.6	3.3 ± 2.8	100%

Table 5.5: This table presents the results of Experiment 4 (Section 5.6) for fifty queries solved by the proposed simplified adaptive roadmap algorithm where for every query an obstacle of $1 \times 1 \times 1$m appeared, to determine the effect of the grid cell size on the planning time. δ_k denotes the length of the grid cell in dimension k. We present the experiment setup in Table 5.9

The adaptation time, which is the planning time divided by the number of adaptations, is lower for larger grid cells, as more vertices were adapted while the planning time did not increase. The higher adaptation time for the smaller grid cells is because more grid cells need to be updated if an obstacle appears. Additionally, the assignment of the vertices was more time-consuming as they were assigned to more grid cells. For the grid cell sizes of $0.25 \times 0.25 \times 0.25$m and $0.5 \times 0.5 \times 0.5$m, this does not show in the planning time as the increased adaptation time was compensated by the lower number of adaptations performed. For a grid cell size $0.1 \times 0.1 \times 0.1$m, the lower number of adaptations did not compensate for the increased adaptation time as the planning time was more than double the planning time of the remaining grid cell sizes.

Note that δ_z is additionally dependent on the obstacles in the air and must be set to a value that ensures the robot can bend its arm to go under obstacles.

For environments with different geometries, this experiment must be repeated to find the grid cell size that results in the lowest planning time; as we discussed that the grid cell size must be determined based on the environment geometry and width of the narrow passages in Section 4.5.2.

5.6.4. Conclusion

As the planning times of the grid cells of half ($0.5 \times 0.5 \times 0.5$m) and equal to ($1 \times 1 \times 1$m) the size of the incremental changes were lowest, these values are preferred over smaller grid cell sizes. We favor the grid cells of half the size of the incremental changes as the path length is shorter. This experiment must be repeated when planning in a different environment as the grid cell size is environment-specific.

5.7. Experiment 5: Disconnected roadmap due to adaptations by sARM

This experiment aimed to find the contribution of the roadmap getting disconnected on the failure rate of sARM. Too many roadmap adaptations cause the roadmap to divide into multiple connected components, as was discussed in Section 5.5.3.

5.7.1. Setup

We carried out experiments in simulation where sARM adapted the roadmap to changes and solved the motion planning problem, and assessed the causes of failure. We set the planning problems in EXOTica [19]. We present details on the problem and planner settings in Table 5.9. sARM solved 200 queries reusing a single roadmap, and for every query, an obstacle was added, and the previously added obstacle was removed. Figure 5.7 presents the hundred incremental changes, which we added for the first hundred queries, and the addition was repeated for the second hundred. The grid cell size for the 3D occupancy grid collision checking was $0.5 \times 0.5 \times 0.25$m, since Experiment 4 (Section 5.6) reported the best performance and $\delta_z = 0.25$m allows the robot to bend its arm to go under the obstacles in the air.

The disconnected roadmap results in failure if $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ are connected to different connected components. Due to the assumption that the incremental changes do not violate the connectivity of the free configuration space (Section 1.1.3), we know a path exists between $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$.

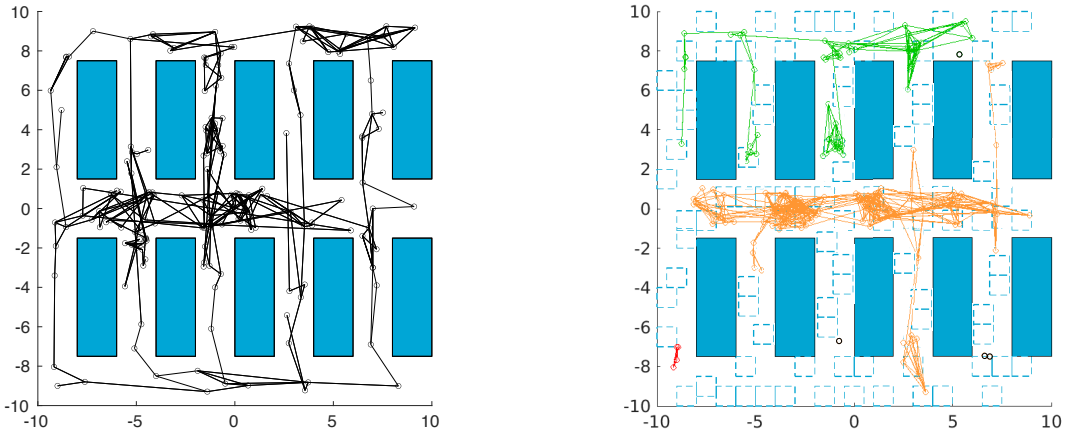
We evaluated the failure rate (100% − success rate) of sARM with respect to the total number of adaptations performed on a single roadmap. Additionally, we assessed the contribution to the failure rate of $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ in different components.

5.7.2. Results

Table 5.6 presents the failure rate of sARM with respect to the total number of adaptations it performed: the more adaptations sARM performed on a single roadmap, the higher the failure rate. Every failure of sARM was caused by $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ connected to a different connected component. Figure 5.9a presents the initial roadmap and Figure 5.9b the disconnected roadmap after 521 adaptations.

Adapted vertices	Failure
100	2.0%
200	7.5%
300	11.3%
400	14.0%
500	17.8%

Table 5.6: This table presents the results of Experiment 5 (Section 5.7) that aims to find how the roadmap disconnection due to the adaptations affects the failure rate. We report the failure rate (100% − success rate) of the proposed simplified adaptive roadmap algorithm with respect to the total number of vertex adaptations it performed on a single roadmap, evaluated for 200 queries where for every query an obstacle of 1 × 1 × 1m appeared. We present the experiment setup in Table 5.9



(a) (b)

Figure 5.9: These figures visualise a 2D projection of the (x,y) coordinates of the (a) initial roadmap and the (b) disconnected roadmap after 521 adaptations by sARM due to two hundred incremental changes in an environment with static obstacles (blue) in Experiment 5 (Section 5.7). We visualise the 2D projection of hundred incremental changes with dashed blue squares; the addition of these obstacles was repeated if all obstacles are added once. The roadmap consists of three connected components in green, orange and red, and four disconnected vertices in black. Note that all experiments are performed in 3D taking all dimensions of the robot and obstacles into account; however, the visualisation of the (x, y) coordinates of the base in the roadmap is more intuitive to show the disconnected roadmap. We present the experiment setup in Table 5.9.

5.7.3. Discussion

The roadmap getting disconnected due to adaptations is the principal reason for failure of sARM. The roadmap gets disconnected because sARM resamples the invalid vertices in their neighbourhood and the replacing vertices connect to their nearest neighbours, which are not necessarily the same as the invalid vertices. If the roadmap is disconnected, this does not necessarily result in a failure because $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ can be connected to the same component.

For the experiments with sARM in this research, the number of adaptations to a single roadmap must be limited by limiting the number of incremental changes to prevent affecting the performance of sARM. Adapting the roadmap for fifty incremental changes of 1 × 1 × 1m resulted in approximately 100 adaptations. As Table 5.6 reported a low failure rate for the 100 adaptations (2%), we suggest limiting the number of incremental changes in the experiments to fifty. However, we must note that this aspect needs improvement in future work.

For future work, we suggest performing local roadmap enhancement by generating new vertices in roadmap areas affected by many adaptations to prevent the disconnected roadmap from affecting the performance of sARM.

5.7.4. Conclusion

The roadmap getting disconnected due to too many adaptations on a single roadmap is the principal reason for failure of sARM. This prevents reusing a single roadmap infinitely.

5.8. Experiment 6: Benchmark sARM to state-of-the-art planners

We performed benchmark experiments to determine whether locally adapting the roadmap by sARM is faster than replanning with a state-of-the-art algorithm (R5 in Section 1.4).

5.8.1. Setup

We carried out experiments in simulation to benchmark sARM to state-of-the-art sampling-based motion planners, RRT and PRM, in two incrementally changing environments. We set the planning problems in EXOTica [19]. We present details on the problem and planner settings in Table 5.9. We added an obstacle for every query and remove the previously added obstacle. The incremental changes were obstacles of $1 \times 1 \times 1$m (see Figure 5.7) because this is approximately the size of incremental changes representing appearing boxes in corridors in a supermarket environment. For both environments, the grid cell size for the 3D occupancy grid collision checking was $1 \times 1 \times 0.25$m, because this is proportionate to the size of incremental changes, does not make the narrow corridors inaccessible, and $\delta_z = 0.25$m allows the robot to bend its arm to go under the obstacles in the air. This experiment planned for fifty queries, and thus fifty incremental changes, to prevent the roadmap from splitting into multiple components, as shown in Experiment 5 (Section 5.7).

We evaluated the number of collision checks, in addition to the planning time, since this is known to be the main driver of the planning time of sampling-based motion planners [39], and this metric is not dependent on the computer used to solve the problem, resulting in more universal benchmarking outcomes. Additionally, we assessed the sources of the collision checks, to get an insight in what parts of sARM are yet to be improved. For a complete interpretation of the results, we assessed the contribution of the collision checks to the planning time. Additional metrics, such as the number of vertices and edges of the graph and success rate, are determined to allow a complete interpretation of the results.

5.8.2. Results

For the three planners in the incrementally changing environments, we visualise the planning time and the time spend on collision checks in Figure 5.10a, and it is clear that collision checking was the principal driver of the planning time. We visualise the planning time and the number of collision checks in Figure 5.10b and Figure 5.10c, respectively. Table 5.7 additionally presents the number of vertices, number of edges, path length, and success rate.

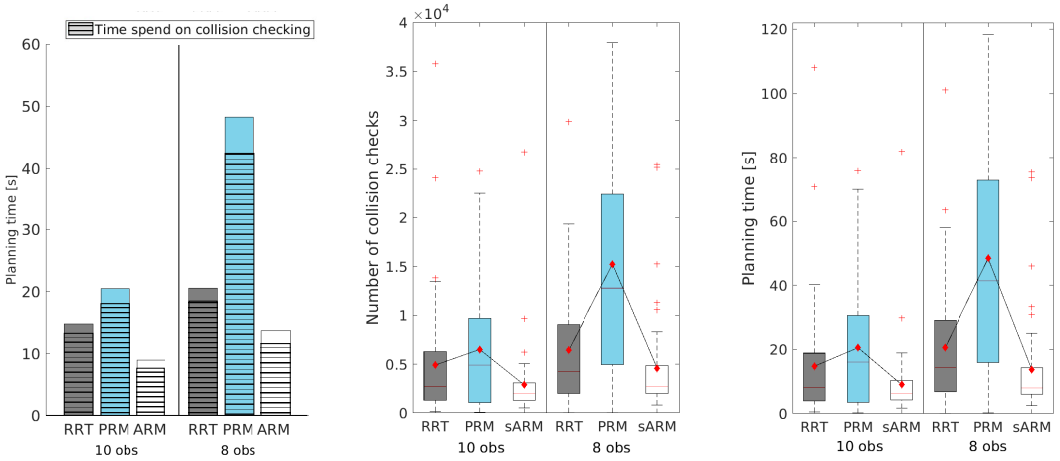
The success rate of sARM was equal to or higher than RRT and PRM in both environments. The mean number of collision checks sARM performed was approximately 40% and 30% and the mean planning time was approximately 40% and 35% less than RRT in the ten-obstacle and eight-obstacle environment, respectively. The mean number of collision checks sARM performed was approximately 60% and 70% and the mean planning time was approximately 65% and 75% less than PRM in the ten-obstacle and eight-obstacle environment, respectively. All algorithms performed more collision checks in the eight-obstacle environment.

The path length of the solution found with sARM was lower than with RRT and PRM in the ten-obstacle environment and higher in the eight-obstacle environment.

Additionally, we assessed the sources of the collision checks in sARM in the ten-obstacle and eight-obstacle environment that perform on average 2.3 and 4.7 adaptations for every incremental change, respectively:

10 obs	**8 obs**	
11%	9%	Initially checking all vertices whether they are valid because sARM does not perform the quick 3D grid lookup.
20%	30%	Generating replacing vertices close by the invalid vertex.
22%	31%	Connecting the replacing vertices to the roadmap.
47%	30%	Connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ to the roadmap.

For the eight-obstacle environment, the percentage of collision checks spent on generating replacing vertices and connecting them to the roadmap increased compared to the collision checks spent on initially checking all vertices and connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ to the roadmap.



(a) (b) (c)

Figure 5.10: These figures present the results of the benchmarking Experiment 6 (Section 5.8) of the proposed simplified adaptive roadmap algorithm (sARM) to state-of-the-art algorithms: the rapidly-exploring random tree algorithm (RRT) and the probabilistic roadmap algorithm (PRM) (Section 5.2.1). (a) Bar graph representing the planning time and time spent on collision checking, indicated in blue, to verify that the collision checking takes up the major part of the planning time for RRT, PRM and the proposed sARM and boxplots of (b) the number of collision checks and (c) the planning time for fifty queries in Experiment 6 in the ten-obstacle (10 obs) and the more challenging eight-obstacle (8 obs) environment (Section 5.2.3) for fifty queries, where for every query an obstacle of $1 \times 1 \times 1$m appeared. The red diamonds indicate the mean planning time for each planner. We present the experiment setup in Table 5.9

	Planner	Planning time [s]	Number of vertices	Number of edges	Number of collision checks	Path length [m]	Number of adaptations	Success
		Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	
10 obs	RRT	14.77 ± 19.75	120.7 ± 195.6	120.7 ± 195.6	4899.7 ± 6585.4	51.5 ± 25.3	-	96%
	PRM	20.55 ± 19.94	77.7 ± 75.0	155.3 ± 186.1	6510.0 ± 6424.5	57.9 ± 32.3	-	98%
	sARM	9.01 ± 11.88	111 ± 0.0	276.0 ± 31.4	2873.6 ± 3887.7	32.7 ± 20.5	2.3 ± 2.9	100%
8 obs	RRT	20.56 ± 20.54	122.3 ± 151.3	122.3 ± 151.3	6429.4 ± 6374.9	59.6 ± 25.8	-	98%
	PRM	48.38 ± 35.88	171.9 ± 107.2	374.6 ± 271.2	15240.0 ± 11372.0	60.0 ± 38.8	-	88%
	sARM	13.70 ± 15.59	237.0 ± 0.0	682.3 ± 130.3	4582.4 ± 5296.1	69.6 ± 36.0	4.7 ± 6.5	98%

Table 5.7: This table presents the results of the benchmarking of the proposed simplified adaptive roadmap algorithm (sARM) to the state-of-the-art planners: the rapidly-exploring random tree algorithm (RRT) and the probabilistic roadmap algorithm (PRM) (Section 5.2.1) in Experiment 6 (Section 5.8), concerning the mean and standard deviations of the planning time, number of vertices, edges, collision checks, and success rates for the three planners in the ten-obstacle (10 obs) and the more challenging eight-obstacle (8 obs) environment (Section 5.2.3) for fifty queries, where for every query an obstacle of $1 \times 1 \times 1$m appeared. We present the experiment setup in Table 5.9

5.8.3. Discussion

Based on the decreased number of collision checks, sARM performed better than PRM and RRT. This reduction in collision checks resulted in a reduction of the planning time. The better performance is due to the roadmap reusal instead of constructing a new graph for every new query or environmental change. PRM performed worst since it has to reconstruct the entire roadmap for every run, which is more time-consuming than the reconstruction of the graph by RRT, equivalent to Experiment 1 (Section 5.3).

The lower path length of the solution of sARM compared to RRT and PRM was because the narrow passages induced by the incremental changes are accessible due to the replacement of an invalid vertex by a vertex in its neighbourhood. RRT and PRM are unlikely to find a path through narrow passages by random sampling in the entire $\mathcal{C}_{\text{free}}$. In the eight-obstacle environment, the path length of the solution of sARM is higher than of RRT and PRM. This environment contained narrow corridors in the initial environment and induced by the incremental changes, making finding a valid vertex close by the invalid vertex challenging. Therefore, the resampled vertex was further from the invalid vertex and it connected to different neighbours, and the connection through the narrow passage is lost. This results in an increased path length as the path takes a detour around this passage.

In the eight-obstacle environment, RRT, PRM and sARM performed 40%, 135% and 50% more collision

checks than in the ten-obstacle environment, respectively. For RRT and PRM this is explained by the narrow passages in $\mathcal{C}_{\text{free}}$, induced by narrow passages in $\mathcal{W}$; sampling-based methods sample randomly, resulting in drawing many samples by the algorithms in the entire $\mathcal{C}_{\text{free}}$ to generate samples in the narrow passage such that connections through them appear [18, 57]. The narrow passages had a larger effect on PRM as it attempts to generate more vertices and edges that are more likely invalid due to the narrow passages. The increased number of collision checks for sARM was due to the doubling in the performed number of adaptations and the narrow passages. The initial roadmap consisted of double the number of vertices compared to the ten-obstacle environment, resulting in approximately double the vertices that become invalid due to an incremental change. Compared to the ten-obstacle environment, the doubling of the number of adaptations in the eight-obstacle environment causes the number of collision checks to increase by approximately 42%, as the collision checks spent on generating a new vertex and connecting it to the roadmap is doubled. The remaining 8% of the 50% increase in collision checks was due to the generation of new vertices in the narrow passages, where more attempted vertices were invalid. This is reflected in the specification of the source of collision checks in the eight-obstacle environment, where a higher percentage of the collision checks is spent on generating replacing vertices and connecting them to the roadmap.

The success rate of PRM was decreased for the eight-obstacle environment because the roadmap construction is more time-consuming in this environment as vertices or edges are necessary in the narrow passages to solve a planning problem. This results in more often exceeding the maximum allowed time.

The performance of PRM and RRT in the ten-obstacle environment was worse than the performance of PRM (SQ) and RRT in Experiment 1 (Section 5.3) in the same environment due to the narrow passages in $\mathcal{C}_{\text{free}}$ induced by the incremental changes.

The main drive of the collision checks of sARM (69%) was due to connecting vertices to the roadmap: either the replacing vertex, $\mathbf{q}_{\text{start}}$ or $\mathbf{q}_{\text{goal}}$. This can be reduced by decreasing the number of neighbours these vertices attempt to connect to. However, this increases the probability that the vertices will not connect to any neighbours, as the nonholonomic constraints of the base and the randomly sampled arm configurations make connecting a random configuration to neighbouring vertices challenging.

5.8.4. Conclusion

The results of this experiment demonstrate the success of sARM reducing the number of collision checks, and accordingly, the planning time, compared to RRT and PRM. sARM satisfied R5 (Section 1.4) because the planning time is decreased compared to the state-of-the-art algorithms for both incrementally changing environments.

5.9. Experiment 7: Effect of increasing the area affected by incremental changes on the speedup of sARM

In Experiment 6 (Section 5.8), we reported an increase in the number of collision checks, and thus the planning time, due to the increased number of adaptations by sARM. If more vertices are invalid due to the incremental changes, we expect the speedup of sARM compared to the state-of-the-art algorithms to disappear. In this experiment, we aim to determine the effect of increasing the area in the workspace affected by incremental obstacles on the speedup by sARM.

5.9.1. Setup

We carried out experiments in simulation to benchmark sARM to RRT and PRM while we increased the number of obstacles simultaneously added to the environment compared to Experiment 6 (Section 5.8). We set the planning problems in EXOTica [19]. We present details on the problem and planner settings in Table 5.9. We performed the experiment in the ten-obstacle environment, to ensure the increase in collision checks, and thus the planning time, was due to updating more vertices instead of updating vertices in narrow passages as would occur in the eight-obstacle environment. All problem and planner settings are equal to the ten-obstacle environment in Experiment 6 (Section 5.8), except for the number of obstacles added and removed for every query. For every query, we added two obstacles of $1 \times 1 \times 1$m to occupy double grid cells and thus invalidate double the vertices of the roadmap compared to when adding one obstacle of this size. In the ten-obstacle environment in Experiment 6 (Section 5.8), sARM performed approximately 41% fewer collision checks than RRT. We expected to loose the speedup of sARM if the number of collision checks is equal for sARM and RRT. Therefore, we added two obstacles as we expected this to increase the number of collision checks by approximately 42%, based on the sources of the collision checks reported in Experiment

6 (Section 5.8) where the remaining sources of collision checks do not scale with the number of vertices that need to be adapted.

We evaluated the number of collision checks and the planning time, as in Experiment 6 (Section 5.8). Additionally, we assessed the sources of the collision checks for sARM, the number of vertices and edges of the graph and success rate to allow a complete interpretation of the results.

5.9.2. Results

Figure 5.11 presents the number of collision checks and planning time for the three planners where for every query two obstacles are added. Table 5.8 additionally presents the number of vertices, edges, path length, number of adaptations and success rate of the planners.

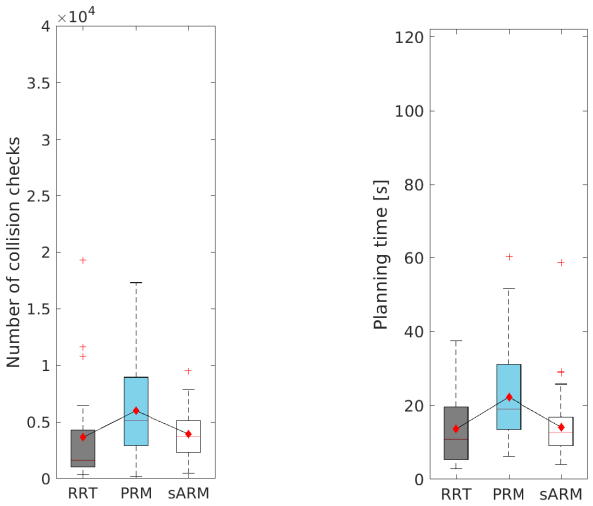


Figure 5.11: These figures present the results of the benchmarking Experiment 7 (Section 5.9) where we add two obstacles of $1 \times 1 \times 1$m for every query, to determine the effect of increasing the area affected by incremental changes on the in Experiment 6 (Section 5.8) reported speedup of sARM compared to state-of-the-art algorithms: the rapidly-exploring random tree algorithm (RRT) and the probabilistic roadmap algorithm (PRM) (Section 5.2.1). We present boxplots of (a) the number of collision checks and (b) the planning time for fifty queries in Experiment 7 (Section 5.9) in the ten-obstacle environment. The red diamonds indicate the mean value for each planner. We present the experiment setup in Table 5.9

Planner	**Planning time [s]**	**Number of vertices**	**Number of edges**	**Number of collision checks**	**Path length [m]**	**Number of adaptations**	**Success**
	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	Mean ± STD	
RRT	13.59 ± 10.02	94.0 ± 127.4	94.0 ± 127.4	4632.3 ± 4428.0	61.4 ± 50.6	-	96%
PRM	22.27 ± 11.65	128.0 ± 76.8	118.8 ± 83.8	7000.0 ± 4020.9	66.2 ± 20.2	-	98%
sARM	14.10 ± 8.99	111 ± 0.0	249.4 ± 22.9	4232.5 ± 2239.0	66.0 ± 18.8	5.5 ± 3.4	94%

Table 5.8: This table presents the results of the benchmarking Experiment 7 (Section 5.9) where we add two obstacles of $1 \times 1 \times 1$m for every query, to determine the effect of increasing the area affected by incremental changes on the in Experiment 6 (Section 5.8) reported speedup of the proposed simplified adaptive roadmap algorithm (sARM) compared to state-of-the-art algorithms: the rapidly-exploring random tree algorithm (RRT) and the probabilistic roadmap algorithm (PRM) (Section 5.2.1). We present the experiment setup in Table 5.9

The number of collision checks and planning time were similar for RRT and sARM, and approximately 60% higher for PRM. The success rates were 96% 98% and 94% for RRT, PRM and sARM, respectively.

We compare the results to the ten-obstacle environment in Experiment 6 (Section 5.8) as we use equal problem and planner settings, except that we doubled the number of obstacles added for each query. By adding two obstacles instead of one, the number of adaptations by sARM is increased by approximately 40%. The number of collision checks for PRM and sARM increased by approximately 7% and 45%, and the planning times by approximately 8% and 55%, respectively. The number of collision checks and planning time decreased by 4% and 7%, respectively. The success rates of RRT and PRM were unchanged, while the suc-

cess rate of sARM dropped from 100% to 94%. The path lengths for RRT, PRM and sARM are increased by approximately 20%, 15% and 105%, respectively, compared to the experiment with one incremental obstacle.

Additionally, we assessed the sources of the collision checks by sARM that performed on average 5.5 adaptations:

7%	Initially checking all vertices whether they are valid because sARM does not perform the quick 3D grid lookup.
30%	Generating replacing vertices close by the invalid vertex.
33%	Connecting the replacing vertices to the roadmap.
32%	Connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ to the roadmap.

The percentage of collision checks spent on generating replacing vertices and connecting them to the roadmap increased compared to the collision checks spent on initially checking all vertices and connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ to the roadmap.

5.9.3. Discussion

We compared these results of adding two obstacles to adding one obstacle in Experiment 6 (Section 5.8). The number of collision checks and the planning time of RRT and PRM changed slightly, as adding two obstacles simultaneously did not initiate more graph reconstructions. The path lengths of RRT and PRM were increased as the extra obstacle initiates an additional narrow passage, which is often not covered with vertices by random sampling [23], resulting in an alternative path that does not include the narrow passage.

The success rate of sARM dropped compared to the environment where one obstacle was added as more vertex adaptations were performed, resulting in the roadmap getting disconnected as we discussed in Experiment 5 (Section 5.7). Additionally, the disconnected roadmap caused the increase in the path length as the connected component that $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ connected to does not cover the entire space. The increased number of collision checks and planning time of sARM can be explained by the increased number of adaptations sARM performs due to the additional obstacle. The increase of the collision checks spent on generating a replacing vertex and connecting it to the roadmap can be explained by the increased number of adaptations sARM performed.

When adding two obstacles, the number of collision checks, and thus the planning time, increased for sARM while it remained similar for RRT, resulting in the loss of the speedup from sARM compared to RRT. However, the speedup compared to PRM remained. It must be noted that the limit on the incremental changes that ensures sARM adapts the roadmap faster than RRT and PRM reconstruct the graph depends the planning problem, for instance, on the environment and the initial roadmap of sARM. Therefore, we can not determine a limit on the incremental changes to maintain the speedup of sARM compared to conventional algorithms.

5.9.4. Conclusion

An increase of the workspace area affected by incremental obstacles in the environment invalidates more vertices, resulting in more adaptations performed by sARM. This may cause losing the speedup compared to RRT and PRM.

5.10. Experiment 8: Real-world implementation of sARM

We performed real-world experiments to assess whether the algorithm can be applied on a real mobile manipulator and not merely in simulation.

5.10.1. Setup

We performed the real-world experiments in the corridor at RoboHouse [45], where the AIRLab Delft is located. We set the planning problems in EXOTica [19]. We present the details on the problem and planner settings in Table 5.9. A path was planned from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$ whereafter an obstacle was added on the shortest path. The appearing obstacle was $0.6 \times 0.3 \times 0.3$m and the 3D grid cell size was $0.5 \times 0.5 \times 0.25$m, approximately the size of the appearing obstacle, and the z-value of 0.25 to allow the robot to go under the obstacle. The roadmap adaptation by sARM must ensure a safe path from $\mathbf{q}_{\text{goal}}$ back to $\mathbf{q}_{\text{start}}$. We performed this experiment with the obstacle placed on the ground, and with the obstacle 1.5m above the ground, where the robot must bend its arm, or navigate its base around, to pass. All other problem and planner settings are identical.

We evaluated whether sARM successfully adapted the roadmap to a change and ensured a safe path for the real robot.

5.10.2. Results

Figure 5.12 visualises the roadmaps and paths of the robot before and after the obstacles are added to the environment. The roadmap was successfully adapted by sARM for both appearing obstacles and the path from $\mathbf{q}_{\text{goal}}$ to $\mathbf{q}_{\text{start}}$ took the obstacle into account. For the incremental change by an obstacle on the ground, three vertices were adapted and for the incremental change by an obstacle in the air, two vertices. The videos of the real-world experiments can be downloaded from the doi: 10.4121/14912766.v1 [17].

5.10.3. Discussion

sARM successfully performed the roadmap adaptation in real-world experiments. If the roadmap adaptation performed by sARM was not successful, the robot would have gone through the appeared obstacle, as this still would have been the shortest path A* found. sARM ensures the roadmap is applicable for multiple queries, despite the incremental changes in the environment.

The extensive arm movement of the robot when the obstacle appeared on the ground was due to the random sampling to construct the roadmap: any valid configuration can be included in the roadmap and thus be part of the planned path. This extensive arm movement does not occur when planning for the base and the arm separately, as in these situations, the arm remains static during locomotion.

The success for the obstacle that appeared in the air uncovers the advantage of coupled planning for the base and arm, where the robot can bend its arm to pass the obstacle. If the planning for the base and the arm would have been decoupled, the robot would have had to navigate around it.

5.10.4. Conclusion

sARM successfully adapted the roadmap to an appearing obstacle on the ground and in the air in the real-world experiments. The path is planned by performing graph search on the adapted roadmap, which ensures that it takes the incremental change into acount.

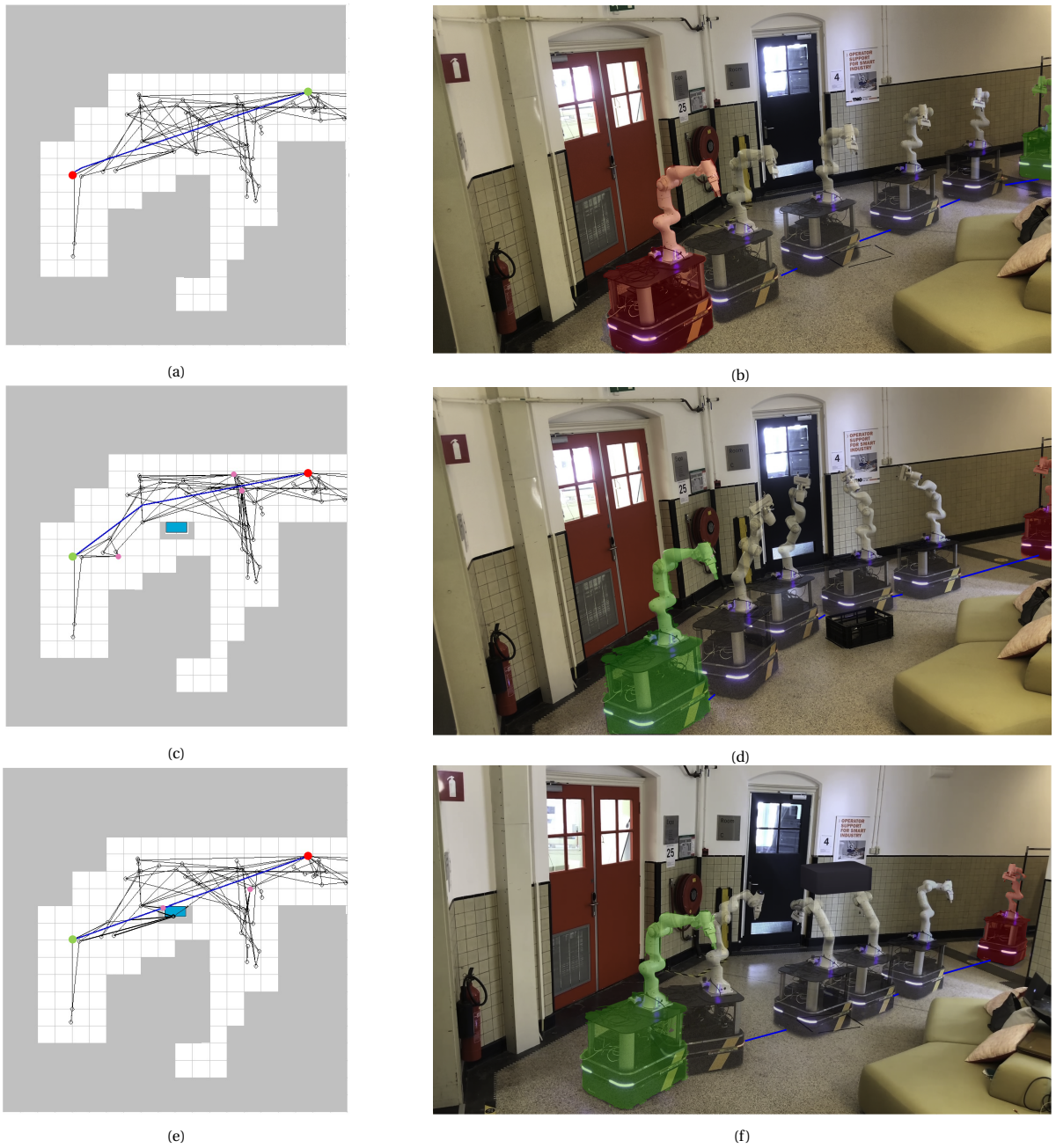


Figure 5.12: These figures present the results of real-world Experiment 8 (Section 5.10): roadmap and real-world paths of (a,b) the 10 degrees of freedom (DOF) robot from start to goal and after the adaptation (c,d) from goal to start after an obstacle is added on the floor or (e,f) in the air. We present simplified roadmaps where the base coordinates of the vertices and edges of the roadmap are visualised in black. Additionally, the occupied grid cells (grey), the appearing obstacle (light blue), start (green), goal (red), adapted vertices (pink) and path (blue) are visualised. We do not visualise the roadmap of the remainder of the corridor (cut off on the right) since this does not change. The photos contain configurations representing the robot's path with the start (green) and goal (red) configuration. Due to the obstacle, the roadmap was adapted by the proposed simplified adaptive roadmap algorithm (sARM) and the graph search algorithm (A*) returned a collision-free path. Note that all experiments were performed in 3D, taking all dimensions of the robot and obstacles into account; however, the visualisation of the (x, y) DOFs in the roadmap allows an intuitive interpretation. We present the experiment setup in Table 5.9.

			Experiment 1			Experiment 2		Experiment 3	Experiment 4	Experiment 5	Experiment 6			Experiment 7			Experiment 8
			RRT	PRM (MQ)	PRM (SQ)	ARM*	sARM*	sARM	sARM	sARM	RRT	PRM (MQ)	sARM	RRT	PRM (MQ)	sARM	sARM
Problem settings	Test	Simulation	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
		Real-world															x
	Robot	3-DOF	x	x	x												
		10-DOF	x	x	x			x	x	x	x	x	x	x	x	x	x
	Initial environment	0 obstacles (20x20x2m)	x	x	x												
		3 obstacles (20x20x2m)	x	x	x												
		10 obstacles (20x20x2m)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
		8 obstacles (20x20x2m)						x			x	x	x				
		RoboHouse corridor															x
	Changing environment	Static	x	x	x												
		Incremental changes #(size [m])				1(1,1,1)	1(1,1,1)	1(1,1,1)	1(1,1,1)	1(1,1,1)	1(1,1,1)	1(1,1,1)	1(1,1,1)	2(1,1,1)	2(1,1,1)	2(1,1,1)	1(0.3x0.6x0.3)
	Queries		100	100	100	n/a	n/a	50	50	200	50	50	50	50	50	50	2
Planner settings	Collision checker	Δq	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
		Sphere-based	x	x	x												
		Occupancy 3D grid (size [m]) + safety [m]				(1,1,1)+0.1	(1,1,1)+0.1	10 obs: (1,1,0.25)+0.1 8 obs: (1,1,0.25)+0.1	[(0.1,0.1,0.1)+0.1 (0.25,0.25,0.25)+0.1, (0.5,0.5,0.5) +0.1, (1,1,1)+0.1]	(0.5,0.5,0.25)+0.1	10 obs: (1,1,0.25)+0.1 8 obs: (1,1,0.25)+0.1	10 obs: (1,1,0.25)+0.1 8 obs: (1,1,0.25)+0.1	10 obs: (1,1,0.25)+0.1 8 obs: (1,1,0.25)+0.1	(1,1,0.25)+0.1	(1,1,0.25)+0.1	(1,1,0.25)+0.1	(0.5,0.5,0.25)+0.1
	Fail criterium	Maximum time [s]	120	120	120	n/a	n/a	120	120	120	120	120	120	120	120	120	120
	RRT	Range	0.05								0.05			0.05			
		Goal bias	15								15			15			
	PRM	Nearest neighbors		10	10							10			10		
		Graph search		A*	A*							A*			A*		
	ARM/sARM	Nearest neighbors				10	10	10	10	10			10			10	10
		d [m]				2	2	[1,6]	2	2			2			2	2
		Graph search				n/a	n/a	A*	A*	A*			A*			A*	A*
		Initial roadmap (vertices,edges)				(168,348)	(168,348)	10 obs: (168,348) 8 obs: (237,480)	(160,322)	(160,322)			10 obs: (111,230) 8 obs: (237,480)			(111,230)	(81,324)

Table 5.9: This table presents the problem and planner settings for Experiments 1-8 (Section 5.3-Section 5.10). PRM (SQ) refers to an implementation of the probabilistic roadmap algorithm (PRM) that clears the roadmap after every run, and PRM (MQ) to the conventional implementation of PRM. ARM* and sARM* refer to implementations of the proposed algorithm and its simplified implementation that merely perform the roadmap adaptation and can not be used as a planner. The incremental changes are denoted by #(size), where # refers to the number of incremental changes of rectangles of size, in (x, y, z). Details on the 3D occupancy grid collision checking are provided by the size, in (x, y, z), of the grid cells and the safety margin used to inflate the bounding box representing the robot in $\mathcal{W}$. We discuss the problem and planner settings more extensively in Section 5.2.

6

Conclusion & Future Work

This chapter provides the conclusion of the conducted research by answering the research question from Chapter 1 and evaluating whether the proposed algorithm meets the requirements. It also provides proposals for future work.

6.1. Conclusion

This thesis answers the research question:

How can local roadmap adaptation allow fast planning for mobile manipulators assuming the environment changes incrementally?

We conclude that local roadmap adaptation allows fast planning for mobile manipulators in incrementally changing environments by using our proposed algorithm: the adaptive roadmap algorithm (ARM). A benchmarking experiment with the simplified implementation of ARM (sARM) confirms that local roadmap adaptation allows fast planning in the incrementally changing environment. This experiment reported speedups of $35-40\%$ and $65-75\%$ compared to the state-of-the-art algorithms RRT (Section 2.5) and PRM (Section 2.6), respectively. The speedup sARM gained compared to existing planners will be magnified for ARM as the roadmap adaptation by ARM is approximately 10% faster than by sARM. However, future work is necessary to improve the algorithm, which we discuss later in this chapter (Section 6.2).

Following the discussions of the experiments (Chapter 5), we can now evaluate to what extend ARM (Section 4.4) and sARM (Section 5.1.4) fulfill the requirements R1-R5, defined in Section 1.4. We conclude that ARM and sARM can:

R1 reuse its roadmap for multiple queries.

- ARM: the roadmap is applicable for multiple queries by design. Note that in the current version of the algorithm, the number of incremental changes must be limited to maintain a connected roadmap and allow planning for multiple queries, as we discussed in Experiment 5 (Section 5.7).
- sARM: Experiment 6 (Section 5.8) presented a success rate of the algorithm of $98-100\%$ without performing reconstruction of the roadmap, which verifies that it is applicable for multiple queries. As with ARM, the number of incremental changes must be limited.

R2 locally adapt the roadmap to changes; these changes can be due to appearing or disappearing obstacles.

- ARM: by design, the algorithm can locally adapt its roadmap due to an appearing obstacle by updating vertices and removing edges to ensure $V \subset \mathcal{C}_{\text{free}}$ and $E \subset \mathcal{C}_{\text{free}}$ if these are associated with a grid cell occupied due to an obstacle. However, the algorithm does not deal with the disappearance of obstacles since it does not generate new vertices in grid cells associated with disappearing obstacles. ARM sets the grid cells from which obstacles have disappeared to free, enabling the generation of new vertices and edges in these grid cells due to an incremental change close by. Experiment 2 (Section 5.4) verified this with the implementation of the roadmap adaptation of ARM.

- sARM: the algorithm can locally adapt its roadmap due to an appearing obstacle by updating the vertices to ensure $V \subset \mathcal{C}_{\text{free}}$. However, the simplifications prevent the removal of edges, which results in $E \not\subset \mathcal{C}_{\text{free}}$. Additionally, the algorithm does not deal with the disappearance of obstacles since it does not generate new vertices in grid cells associated with disappearing obstacles. sARM sets these grid cells to free, enabling the generation of new vertices and edges in these grid cells due to an incremental change close by. Experiments 3-8 (Section 5.5-Section 5.10) verified this.

R3 be applicable for a high-DOF robot (high dimensional configuration space).

- ARM: the dimension of the robot is not limited due to the sampling in $\mathcal{C}$ to generate replacing vertices. Experiment 2 (Section 5.4) verified this with the implementation of the roadmap adaptation of ARM for a 10-DOF mobile manipulator.
- sARM: the simplifications do not affect the applicability for a high-DOF robot; therefore, sARM meets this requirement. Experiments 3-8 (Section 5.5-Section 5.10) with a 10-DOF mobile manipulator verified this.

R4 be applicable in a 3D workspace.

- ARM: by design, the algorithm is applicable in $\mathcal{W} = \mathbb{R}^3$, by generating a 3D bounding box around the entire robot as workspace representation of a configuration. The implementation of the roadmap adaptation of ARM in Experiment 2 (Section 5.4) successfully represented configurations in $\mathcal{W}$.
- sARM: the algorithm is applicable in $\mathcal{W} = \mathbb{R}^3$ as the workspace representation of configurations is equivalent to ARM. The Experiments 3-8 (Section 5.5-Section 5.10) successfully represented configurations in $\mathcal{W}$.

R5 adapt the roadmap faster than replanning.

- ARM: we did not present comparisons concerning the planning times as we implemented the roadmap adaptation of this algorithm and not the entire planner. However, we expect the speedup sARM gained compared to existing planners will be magnified for ARM as the roadmap adaptation by ARM is faster than by sARM.
- sARM: the results of the benchmarking Experiment 6 (Section 5.8) for coupled motion planning for a mobile manipulator in different incrementally changing environments reported a $30-40\%$ reduction in performed collision checks, which resulted in a $35-40\%$ speedup of the planning time, compared to the single-query algorithm rapidly-exploring random tree (RRT) that reconstructs its graph for every new query or change in the environment. Compared to the multi-query algorithm probabilistic roadmap (PRM) the decrease in collision checks was $60-70\%$, which resulted in a $65-75\%$ speedup of the planning time. As the roadmap reconstruction is more time-consuming for PRM, sARM achieved more reduction in the planning time than compared to RRT. The number of incremental changes appearing simultaneously must be limited to maintain the advantage of sARM, as we concluded from the results of Experiment 7 (Section 5.9).

sARM meets all requirements but one (R2). ARM ensures that the entire roadmap is in $\mathcal{C}_{\text{free}}$, and not merely the edges; however, it does not meet R2 since it does not deal with disappearing obstacles. ARM must be implemented as a planner to assess whether it meets R5.

6.2. Future work

This section identifies directions for future work, based on limitations that arose in our work, as described in earlier chapters. In Section 6.2.1 and Section 6.2.2 we suggest future work for a more extensive evaluation of the proposed algorithm, in Section 6.2.3 we propose an approach that maintains a high success rate if the algorithm performs many adaptations on a single roadmap, and in Section 6.2.4, Section 6.2.5, and Section 6.2.6 we propose extensions to the algorithm to improve the performance even further.

6.2.1. Implementation of non-simplified ARM as planner

Future work is needed to implement ARM as a planner, without simplifications as in sARM, to assess whether it meets all requirements from Section 1.4 and to implement it in simulation and on the real robot. The simplified implementation does not perform the quick lookup of vertices and edges if the occupancy of grid cells

is updated due to the lack of communication between OMPL and EXOTica, as was discussed in Section 5.1. For future research, we suggest implementing ARM as a planner within OMPL. ARM not alone ensures all edges in the roadmap are in $\mathcal{C}_{\text{free}}$, the roadmap adaptation by ARM is faster than by sARM. The speedup sARM gained compared to existing planners will be magnified for ARM.

To enable the implementation of ARM in simulation and on the real robot in ROS using EXOTica, additional communication between OMPL and EXOTica is necessary to allow assigning vertices and edges. We suggest implementing a learned lookup table, which allows us to find the associated grid cells for a configuration. We suggest learning the table to ensure the most informative grid cell-configuration pairs are included. The lookup table is generated before the algorithm execution, and the planner loads it during the planner initialisation. Additionally, the planner needs to receive data concerning the updated occupancy of the 3D grid.

6.2.2. Benchmarking to more sophisticated algorithms

For a complete assessment of the performance of ARM, we suggest performing benchmarking experiments with more sophisticated algorithms than RRT and PRM. Algorithms we suggest to include in the benchmarking are RRT^X [37], which is a single-query planner that locally adapts its graph to changes, and the multi-query algorithms that locally adapt their roadmap that are suitable for mobile manipulators in Chapter 3 [34, 51, 58, 59]. We did not compare our algorithm to RRT^X, as existing implementations are merely suitable for mobile robots in 2D workspaces [38], or are applicable for high-DOF robots but can not deal with changes in the environment [28]. No implementations for the algorithms from Chapter 3 are available. We did not create implementations of the more sophisticated algorithms in this thesis due to time constraints.

6.2.3. Roadmap enhancement

We suggest roadmap enhancement by locally adding vertices or edges to prevent the roadmap from splitting into multiple components after many adaptations and to enable dealing with disappearing obstacles. If a roadmap is adapted many times, the roadmap may become disconnected, which means that the roadmap will not consist of one connected component, but of multiple; which causes failure if $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ connect to a different component. The algorithm must evaluate whether the roadmap consists of multiple components to unite the components. We suggest counting the total number of vertices and the vertices of the connected component of a randomly chosen vertex. If these values differ, the roadmap enhancement must be initiated. We suggest to generate new vertices in areas affected by incremental changes. The new vertices try to connect to the neighbouring vertices, and if connecting to vertices of different connected components is possible, these two components are united. An alternative to sampling in affected areas is reinstating the invalid vertices due to appeared obstacles that have disappeared. Additionally, this approach enables the algorithm to deal with disappearing obstacles as areas where once an obstacle appeared will be accessible if the obstacle is gone.

6.2.4. 3D grid improvement

To enhance the performance of ARM, we suggest improving the way the, currently simplistic, 3D grid structure represents the environment. We suggest altering the 3D grid to allow varying resolutions. Smaller grid cells can represent areas where small obstacles cause incremental changes to limit the number of adaptations due to a tiny obstacle to the vertices that are actually in $\mathcal{CO}$. Additionally, smaller grid cells ensure the passages in narrow corridors are accessible for the robot due to the less conservative occupancy of the grid cells. Lastly, smaller grid cells allow a less conservative representation of areas occupied by non-rectangular obstacles. Larger grid cells can represent regions where larger obstacles appear to allow quicker initiation of the roadmap adjustment because the occupancy of fewer grid cells needs to be updated.

6.2.5. Less conservative workspace representation of configurations

We suggest an alternative representation of the robot in $\mathcal{W}$ for a more accurate assignment of configurations to grid cells, resulting in improved roadmap adaptation and more flexibility. We suggest representing the robot with two bounding boxes: one for the base and one for the arm. This increases the flexibility because it enables extending the arm above an obstacle while the base is next to the obstacle.

6.2.6. Biased resampling

We suggest guiding the resampling of invalid vertices to speed up finding a replacing vertex and connecting it to the roadmap. The narrow passages induced by the incremental changes in the corridors cause more attempted replacing vertices to be invalid. As the 3D occupancy grid informs us on the narrow passages in $\mathcal{W}$, we suggest to bias the resampling to $\mathcal{C}_{\text{free}}$, for instance by potential field techniques [4] to attract sampling in the narrow passage.

We suggest an additional approach to speed up finding a replacing vertex: by predefining meaningful arm configurations to draw a configuration from, while randomly sampling to obtain a base configuration. For instance, these meaningful arm configurations could consist of a configuration in which the arm is entirely above the base and configurations where the arm is bend and rotated to the right, left, front and back of the base. We are confident these configurations are not in self-collision or violate the joint limits. We expect that in many situations, at least one of these arm configurations ensures assignment of the configuration to solely free grid cells. However, if all predefined arm configurations result in assigning the configuration to occupied grid cells, we suggest random sampling for the entire robot. Additionally, if the robot is represented by multiple bounding boxes as suggested in Section 6.2.5 and merely the box representing the arm is associated with occupied grid cells, we suggest attempting to find a replacing vertex by merely adapting the arm by drawing a configuration from the predefined arm configurations while maintaining the pose of the base. If this does not result in a valid replacing vertex, we suggest resampling the entire robot configuration.

For nonholonomic robots in environments with narrow corridors, such as supermarkets, we suggest guiding the resampling of the invalid vertex by the orientation of the base to be aligned with the passage. Aligning configurations with the corridors makes connecting to neighbours more straightforward for nonholonomic robots, as this prevents driving along arcs within a narrow space. To enable this guided sampling, areas containing corridors in the initial environment must be marked, and the base orientation to align it with the passage must be specified.

Acknowledgements

I would like to thank Dr. J. Alonso-Mora for getting me interested in motion planning during one of my first MSc courses, for being my supervisor, and thereby providing me the opportunity to graduate within the AMR group at the TU Delft.

Furthermore, I would like to thank Max Spahn for being my daily supervisor. I could always count on your thoughts during the weekly catch-ups and you always responded within minutes to my questions on Teams. You taught me a lot this year about robotics as well as academic work.

I would like to thank Robbert and my roommates, as you were always there to celebrate the ups or cheer me up during the downs when writing a thesis from home in the COVID-19 pandemic.

Last, but not least, I would like to thank my parents and sisters for always supporting me and believing in me during the journey from elementary school to a Master of Science.

E.J. Heerkens
Delft, July 9, 2021

Bibliography

[1] AIRLab Delft. https://icai.ai/airlab-delft/. Accessed on: 28-09-2020.

[2] ROS. https://www.ros.org. Accessed on: 10-05-2021.

[3] Ali Akbar Agha-mohammadi, Suman Chakravorty, and Nancy M. Amato. FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *International Journal of Robotics Research*, 33(2):268–304, 2014. ISSN 17413176. doi: 10.1177/0278364913501564.

[4] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach.

[5] Amy J. Briggs, Carrick Detweiler, Daniel Scharstein, and Alexander Vandenberg-Rodes. Expected shortest paths for landmark-based robot navigation. *The International Journal of Robotics Research*, 23(7): 717–728, 2004. doi: 10.1007/978-3-540-45058-0_23.

[6] Oliver Brock and Lydia E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. *IEEE Conference on Robotics and Automation (ICRA)*, 2:1469–1474, 2001. doi: 10.1109/ROBOT.2001.932817.

[7] Andrej Brodnik, Svante Carlsson, Robert Sedgewick, J.I. Munro, and E.D. Demaine. Resizable arrays in optimal time and space. *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, pages 10–22, 1999. doi: 10.1007/3-540-48447-7_4.

[8] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion-Theory, Algorithms and Implementation*. The MIT Press, London, England, 2005.

[9] Clearpath. Boxer Indoor Robotic Platform: Datasheet. https://go.pardot.com/l/92812/2018-10-18/5p7x5n/92812/113199/Boxer_DataSheet_2020.pdf. Accessed on: 20-05-2021.

[10] David Coleman, Sachin Chitta, and Ioan A. Şucan. Reducing the barrier to entry of complex robotic software: a moveit! case study. *Journal of Software Engineering for Robotics*, 5(1):3–16, 2014. doi: 10.6092/JOSER_2014_05_01_p3. Software available at http://github.com/ros-planning/moveit.

[11] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. *Proceedings of the Seventh National Conference on Artificial Intelligence*, page 49–54, 1988. doi: 10.5555/2887965.

[12] Lester Eli Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957. doi: 10.2307/2372560.

[13] Franka Emika. Panda: Datasheet. https://wiredworkers.io/wp-content/uploads/2019/12/Panda_FrankaEmika_ENG.pdf. Accessed on: 20-05-2021.

[14] Russell Gayle, Avneesh Sud, Ming C. Lin, and Dinesh Manocha. Reactive deformation roadmaps: Motion planning of multiple robots in dynamic environments. *IEEE International Conference on Intelligent Robots and Systems*, pages 3777–3783, 2007. doi: 10.1109/IROS.2007.4399287.

[15] Roland Geraerts and Mark H. Overmars. A comparative study of probabilistic roadmap planners. *Springer Tracts in Advanced Robotics*, 7 STAR:43–57, 2004. ISSN 16107438. doi: 10.1007/978-3-540-45058-0_4.

[16] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):2951–2957, 2015. ISSN 10504729. doi: 10.1109/ICRA.2015.7139603.

[17] Evelien Heerkens. Experiment data and videos, underlying the msc thesis: Local roadmap adaptation for mobile manipulators in incrementally changing environments, Jul 2021. URL https://data.4tu.nl/articles/dataset/Experiment_data_and_videos_underlying_the_MSc_thesis_Local_roadmap_adaptation_for_mobile_manipulators_in_incrementally_changing_environments/14912766/1.

[18] D. Hsu, L. E Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Robotics: The Algorithmic Perspective*, 1998.

[19] Vladimir Ivan, Yiming Yang, Wolfgang Merkt, Michael Camilleri, and Sethu Vijayakumar. *EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control*, pages 211–240. 2019. ISBN 978-3-319-91589-0. doi: 10.1007/978-3-319-91590-6_7. Software available at http://github.com/ipab-slmc/exotica,.

[20] Edwin A. Jackson. *Equilibrium Statistical Mechanics*. Prentice-Hall, Upper Saddle River, USA, 1968. doi: 10.1063/1.3035486.

[21] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011. ISSN 02783649. doi: 10.1177/0278364911406761.

[22] Lydia E. Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. *IEEE International Conference on Robotics and Automation*, 1994. doi: 10.1109/IROS.1994.407619.

[23] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996. doi: 10.1109/70.508439.

[24] Weria Khaksar, Md Zia Uddin, and Jim Torresen. Self-Adjusting Roadmaps: A Fast Sampling-Based Path Planning Algorithm for Navigation in Unknown Environments. *IEEE International Conference on Robotics and Biomimetics*, pages 1094–1101, 2018. doi: 10.1109/ROBIO.2018.8665326.

[25] Weria Khaksar, Md Zia Uddin, and Jim Torresen. Incremental Adaptive Probabilistic Roadmaps for Mobile Robot Navigation under Uncertain Condition. *15th International Conference on Electrical Engineering, Computing Science and Automatic Control*, 2018. doi: 10.1109/ICEEE.2018.8533989.

[26] Weria Khaksar, Md Zia Uddin, and Jim Torresen. Multiquery Motion Planning in Uncertain Spaces: Incremental Adaptive Randomized Roadmaps. *International Journal of Applied Mathematics and Computer Science*, 29(4):641–654, 2020. ISSN 20838492. doi: 10.2478/amcs-2019-0047.

[27] Oussama Khatib. Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26(2):175–183, 1999. doi: 10.1016/S0921-8890(98)00067-0.

[28] Kavraki Lab. Ompl ompl::geometric::rrtxstatic class reference. https://ompl.kavrakilab.org/classompl_1_1geometric_1_1RRTXstatic.html, 2019. Accessed on: 01-07-2021.

[29] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Press, 1991. ISBN 978-0-7923-9129-6. doi: 10.1007/978-1-4615-4022-9.

[30] Jean-Paul Laumond. *Robot motion planning and control*. Springer, Toulouse, 1988. ISBN 3-540-76219-1,.

[31] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[32] Steven M. LaValle. Planning algorithms. *Planning Algorithms*, 2006. doi: 10.1017/CBO9780511546877.

[33] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20:378–400, 2001. doi: 10.1177/02783640122067453.

[34] Peter Lehner, Arne Sieverling, and Oliver Brock. Incremental, sensor-based motion generation for mobile manipulators in unknown, dynamic environments. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4761–4767, 2015. ISSN 10504729. doi: 10.1109/ICRA.2015.7139861.

[35] Qinghua Li, Yaqi Mu, Yue You, Zhao Zhang, and Chao Feng. A Hierarchical Motion Planning for Mobile Manipulator. *IEEJ Transactions on Electrical and Electronic Engineering*, pages 1390–1399, 2020. ISSN 19314981. doi: 10.1002/tee.23206.

[36] Tomas Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983. doi: 10.1109/TC.1983.1676196.

[37] Michael Otte and Emilio Frazzoli. RRT X: Asymptotically Optimal Single-Query Sampling-Based Motion Planning with Quick Replanning. *The International Journal of Robotics Research*, 2016. doi: 10.1177/0278364915594679.

[38] Michael W. Otte. Ottelab code rrt-x. `http://ottelab.com/html_stuff/code.html`, 2015. Accessed on: 01-07-2021.

[39] Jia Pan and Dinesh Manocha. Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *International Journal of Robotics Research*, 35(12):1477–1496, 2016. ISSN 17413176. doi: 10.1177/0278364916640908.

[40] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012. doi: 10.1109/ICRA.2012.6225337.

[41] Jae Han Park, Ji Hun Bae, and Moon Hong Baeg. Adaptation algorithm of geometric graphs for robot motion planning in dynamic environments. *Mathematical Problems in Engineering*, 2016, 2016. ISSN 15635147. doi: 10.1155/2016/3973467.

[42] Vinay Pilania and Kamal Gupta. A localization aware sampling strategy for motion planning under uncertainty. *IEEE International Conference on Intelligent Robots and Systems*, pages 6093–6099, 2015. ISSN 21530866. doi: 10.1109/IROS.2015.7354245.

[43] Vinay Pilania and Kamal Gupta. Mobile manipulator planning under uncertainty in unknown environments. *The International Journal of Robotics Research*, 37(2):316–339, 2018. doi: 10.1177/0278364918754677.

[44] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes bothforwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990. doi: 10.2140/pjm.1990.145.367.

[45] RoboValley. Robohouse. `https://robovalley.com/robohouse/`, 2021. Accessed on: 17-06-2021.

[46] Stuart J. Russell. *Artificial intelligence a modern approach*. Norvig, Peter, Boston, 2018. ISBN 978-0134610993.

[47] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. Efficient autonomous exploration planning of large-scale 3-d environments. *IEEE Robotics and Automation Letters*, 4(2):1699–1706, 2019. doi: 10.1109/LRA.2019.2897343.

[48] Andrew Short, Zengxi Pan, Nathan Larkin, and Stephen van Duin. Recent progress on sampling based dynamic motion planning algorithms. *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1305–1311, 2016. doi: 10.1109/AIM.2016.7576950.

[49] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. Boost graph library. `http://www.boost.org/libs/graph/doc/index.html`, 2001. Accessed on: 24-05-2021.

[50] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co, Boston, MA, USA, 2002.

[51] Arne Sieverling, Nicolas Kuhnen, and Oliver Brock. Sensor-based, task-constrained motion generation under uncertainty. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4348–4355, 2014. ISSN 10504729. doi: 10.1109/ICRA.2014.6907492.

[52] Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, 2006. ISBN 0-619-21764-2.

[53] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. doi: 10.1109/MRA.2012.2205651. Software available at `http://github.com/ompl/ompl`,.

[54] Shantanu Thakar, Liwei Fang, Brual Shah, and Satyandra Gupta. Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 981–987, 2018. doi: 10.1109/COASE.2018.8560446.

[55] Shantanu Thakar, Pradeep Rajendran, Ariyan M. Kabir, and Satyandra K. Gupta. Manipulator motion planning for part pickup and transport operations from a moving base. *IEEE Transactions on Automation Science and Engineering*, 2020. doi: 10.1109/TASE.2020.3020050.

[56] R.L. Wilder. Evolution of the topological concept of "connected". *American Mathematical Monthly*, 85 (9):720–726, 1978. doi: 10.2307/2321676.

[57] S. A. Wilmarth, N. M. Amato, , and P. F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. *Proceedings on the fifteenth annual symposium on computational geometry*, 1999.

[58] Yuandong Yang and Oliver Brock. Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. *Robotics: Science and Systems*, 2:279–286, 2007. ISSN 2330765X. doi: 10.15607/rss.2006.ii.036.

[59] Yuandong Yang and Oliver Brock. Elastic roadmaps-motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28(1):113–130, 2010. ISSN 09295593. doi: 10.1007/s10514-009-9151-x.

[60] Eiichi Yoshida and Fumio Kanehiro. Reactive robot motion using path replanning and deformation. *Proceedings - IEEE International Conference on Robotics and Automation*, (December 2013):5456–5462, 2011. ISSN 10504729. doi: 10.1109/ICRA.2011.5980361.

[61] Eiichi Yoshida, Kazuhito Yokoi, and Pierre Gergondet. Online replanning for reactive robot motion: Practical aspects. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 5927–5933, 2010. doi: 10.1109/IROS.2010.5649645.

[62] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. CHOMP: Covariant Hamiltonian optimization for motion planning. *International Journal of Robotics Research*, 32(9-10):1164–1193, 2013. ISSN 02783649. doi: 10.1177/0278364913488805.

A

Parameter selection OMPL planners

In this appendix, we present performed experiments for selecting the parameters for RRT and PRM in the Open Motion Planning Library (OMPL) [53] for the experiments in Chapter 5. We benchmarked the planners with different values for the parameters based on the default value set for the planner in OMPL.

A.1. Range parameter RRT

For RRT, we aim to select a value for the range parameter, which is the maximum length of an edge to be added to the graph.

A.1.1. Experiment setup

We carried out experiments in simulation to compare the effect of changing the range parameter of RRT on the performance. We set the planning problems in EXOTica [19]. We performed the parameter selection for the 10-DOF mobile manipulator in Section 5.2.2 in the static ten-obstacle environment in Section 5.2.3 because we aimed to deploy our proposed algorithm in obstacle-cluttered environments and therefore perform parameter selection for RRT for benchmarking. We evaluated the planner for 50 runs to get more substantiated results. We set the fail criterion to a maximum planning time of 120s. We set the goal bias, introduced in Section 2.5, to 0.05 as is suggested by OMPL. For collision checking, we used the collision checking based on the occupancy of a 3D grid from Section 5.2.5 and an interpolation resolution for checking the validity of the edges of 0.01. The range parameter RRT in OMPL estimates for this environment in the configuration of the planner is 10m. We compared the performance of RRT with range parameters of values 5m, 10m, 15m and 20m. We evaluated the planning time, path length, and success rate of the results.

A.1.2. Results

We present the results of RRT for the different range parameters in Table A.1. The planning time was lowest for the range parameter of 15m. The path length was similar for the range parameters of 10m and 15m and slightly lower for 5m and slightly higher for 20m. The success rate was highest for a range of 20m.

Range [m]	**Planning time [s]**	**Number of collision checks**	**Path length [m]**	**Success**
	Mean ± STD	Mean ± STD	Mean ± STD	
5	19.73 ± 25.64	5993.5 ± 7780.6	46.4 ± 21.9	86%
10	19.61 ± 23.80	6039.4 ± 7320.8	49.8 ± 19.6	82%
15	13.49 ± 18.37	3786.7 ± 5127.9	49.9 ± 19.3	88%
20	22.14 ± 29.00	6597.5 ± 8652.3	53.1 ± 23.9	92%

Table A.1: This table presents the results of the Experiment to select the parameter for the range, which is the maximum length of an edge to be added to the graph, of the rapidly-exploring random tree (RRT) algorithm in the Open Motion Planning Library (OMPL) [53]. We discuss the experiment setup in Appendix A.1.1.

A.1.3. Discussion

None of the range parameters ensured the best performance on planning time, path length and success rate. The range of 15m performed best on planning time and second-best on path length and success rate. The planning time of the range parameter of 20m was nearly doubled compared to the value of 15m, while the success rate was merely increased by 4%. This increased adaptation time is because longer edges are more likely to be invalid due to an obstacle, resulting in more attempts necessary to find a valid edge and construct the tree. The increased planning times for the range values of 5 and 10, compared to a range of 15, were because more edges are generated to reach $\mathbf{q}_{\text{goal}}$ as the edges are shorter. Additionally, more edges that would not be part of the graph were generated before the goal was reached, increasing the planning time. The range parameter of 15m ensures the best performance of RRT in this environment based on its lowest planning time and second-best success rate.

For environments of different sizes or with different obstacles, this experiment must be repeated. All experiments RRT solves a planning problem in, in this thesis, have the same size. The eight-obstacle environment in Section 5.2.3 consists of a similar geometry as the ten-obstacle environment in this experiment. If the range parameter results in a high success rate for this obstacle-cluttered environment, it will also result in a high success rate for environments with fewer obstacles, such as the zero-obstacle and three-obstacle environments in Section 5.2.3. As we do not perform parameter selection for all environments separately, we suggest setting the value to 15m, which results in a good performance for all environments.

If one desires to find the parameter that ensures the best performance for RRT, the range parameter selection must be performed for more range values between 5 and 20. Additionally, one can perform the parameter selection specifically for every environment.

A.1.4. Conclusion

The range parameter of 15m ensured the best performance of RRT in this environment, and we assume that it is also suitable for the remaining environments RRT solves planning problems in, in this thesis.

A.2. Nearest neighbour parameter PRM

For PRM we aimed to select a value for the number of nearest neighbours a new vertex attempts to connect to.

A.2.1. Experiment setup

We carried out experiments in simulation to compare the effect of changing the nearest neighbours of PRM on the performance. We set the planning problems in EXOTica [19]. We performed the parameter selection for the 10-DOF mobile manipulator in Section 5.2.2 in the static ten-obstacle environment in Section 5.2.3 because we aimed to deploy our proposed algorithm in obstacle-cluttered environments and therefore performed parameter selection for PRM for benchmarking. We evaluated the planner for 50 runs to get more substantiated results. We set the fail criterion to a maximum planning time of 120s. For collision checking, we used the collision checking based on the occupancy of a 3D grid from Section 5.2.5 and an interpolation resolution for checking the validity of the edges of 0.01. The default nearest neighbour parameter in OMPL is 10. We compared the performance of PRM with 5, 10, 15 and 20 as the number of nearest neighbours. We evaluated the planning time, path length, and success rate of the results.

A.2.2. Results

We present the results of PRM for the different number of nearest neighbours in Table A.2. If the number of nearest neighbours was 5, the success rate was 64 %. The remaining values for the number of nearest neighbours ensured a higher success rate. The planning time was highest if the parameter was set to 20. When the parameter was 10, the planning time was slightly higher than when it was 15; however, the success rate was 10 % higher.

A.2.3. Discussion

None of the nearest neighbour parameters ensured the best performance on planning time, path length and success rate. The values 5 and 20 are unsuitable due to their low success rate and high planning time, respectively. The low success rate of 5 nearest neighbours is due to the inability to connect a new sample to the roadmap resulting in drawing more samples that would not be part of the roadmap. This increases the planning time and results more often in not finding a solution within the maximum allocated time. The increased

Nearest neighbours	**Planning time [s]**	**Number of collision checks**	**Path length [m]**	**Success**
	Mean ± STD	Mean ± STD	Mean ± STD	
5	3.95 ± 7.52	18743.2 ± 10027.0	43.9 ± 15.0	64%
10	5.85 ± 11.56	1904.3 ± 3798.6	36.0 ± 12.2	98%
15	4.02 ± 3.65	1218.7 ± 1127.0	38.1 ± 13.2	88%
20	11.45 ± 26.19	3647.7 ± 8340.6	47.3 ± 16.9	92%

Table A.2: This table presents the results of the experiment that performs the parameter selection for the number of nearest neighbours a new vertex attempts to connect to in the probabilistic roadmap algorithm (PRM) in the Open Motion Planning Library (OMPL) [53]. We discuss the experiment setup in Appendix A.2.1.

planning time of 20 nearest neighbours is due to the more time-consuming collision checks are performed on attempted connections. This increased time-consuming collision checks caused the success rate of 15 nearest neighbours to decrease, as the planning time is increased above the maximum allowed time. The success rate of 98% makes the parameter 10 most suitable for planning with PRM in this environment.

For environments of different sizes or with different obstacles, this experiment must be repeated. All experiments PRM solves a planning problem in, in this thesis, have the same size. The eight-obstacle environment in Section 5.2.3 consists of a similar geometry as the ten-obstacle environment in this experiment. Therefore, we assume the number of nearest neighbour parameter of 10 is also suitable for this environment. If the nearest neighbour parameter results in a high success rate for this obstacle-cluttered environment, it will also result in a high success rate for environments with fewer obstacles, such as the zero-obstacle and three-obstacle environments in Section 5.2.3. However, in environments with fewer obstacles, a lower number of nearest neighbours will result in a higher success rate than in an obstacle-cluttered environment since fewer attempted connections are invalid. As we do not perform parameter selection for all environments separately, we suggest setting the value to 10, which results in a good performance for all environments.

If the goal is to find the optimal number of nearest neighbours parameter for PRM, we suggest performing the parameter selection for more values between 5 and 20. Additionally, we suggest performing the parameter selection specifically for every environment.

A.2.4. Conclusion

The number of nearest neighbours parameter of 10 ensured the best performance of PRM in this environment, and we assume that it is also suitable for the remaining environments PRM solves planning problems in, in this thesis.