UNIVERSITÉ LIBRE DE BRUXELLES

BRUFACE BRUSSELS FACULTY OF ENGINEERING

VRIJE UNIVERSITEIT BRUSSEL

Graduation thesis submitted for acquiring the degree of Master of Science in Electromechanical Engineering: Robotics & Mechanical Construction

Multi-Robot Task and Motion Planning

Conflict-Based Search for Sampling-Based Motion Planning of Multi-Manipulator Systems

Viktor Laurens De Groote

Master thesis submitted under the supervision of
Prof. ir. Bram Vanderborght

The co-supervision of
PhD ir. Gaoyuan Liu and Prof. ir. Ilias El Makrini

Academic year
2023-2024

In order to be awarded the Master's degree in
Electromechanical Engineering: Robotics & Mechanical Construction

Abstract

As robotic systems become increasingly integral across various industries, the complexities of coordinating multiple autonomous robots in shared environments have drawn significant research attention. Multi-Robot Systems (MRS) promise enhanced efficiency, robustness, and new possibilities, but they also introduce new challenges in motion planning. Particularly for coordinating multiple manipulators,[1] the need for advanced motion planning strategies becomes critical due to the high dimensionality and increased complexity of the problem.

Traditionally, coupled and decoupled approaches are applied to address the motion planning problem for multiple robots. Coupled approaches demonstrate substantial scalability issues due to the exponential growth in computation time and memory requirements with each additional Degree Of Freedom (DOF) in the Multi-Robot System (MRS). Conversely, decoupled approaches, while more scalable, often fall short in terms of the quality of the solution and are not able to provide any guarantees on whether or not a solution will be found when one exists.

This work explores a hybrid motion planning framework, which uses Conflict-Based Search (CBS) as the low-level search algorithm for Probabilistic Roadmaps (PRM). This hybrid approach aims to combine the benefits of both coupled and decoupled methodologies. To evaluate its effectiveness, the hybrid approach is tested and compared against both coupled and decoupled methods through Python-based simulations in two experimental settings: one involving planar manipulators in close proximity and the other featuring Franka Emika Panda manipulators in close proximity in PyBullet. The experimental findings indicate that CBS-PRM scales better with increasing dimensionality of the motion planning problem compared to coupled approaches but that it also shows improved success rates and yields paths with lower cost compared to decoupled methods.

The development of this hybrid approach was facilitated by the creation of a multi-robot motion planning library, which includes implementations of the discussed algorithms and supports benchmarking across various scenarios. While the hybrid approach offers significant improvements, there are still numerous opportunities for further refinement and improvements.

Keywords

Multi-Manipulator Motion Planning, Sampling-Based Motion Planning, Conflict-Based Search and Probabilistic Roadmaps.

Viktor Laurens De Groote

Master of Science in Electromechanical Engineering: Robotics and Mechanical Construction

2023-2024

Multi-Robot Task and Motion Planning

[1] In the context of this thesis, the term manipulator refers to a robotic arm, functioning independently rather than as a component of a more complex robot, with an end effector used to manipulate objects. This includes articulated industrial robots, cobots, and SCARA robots.

Acknowledgements

I want to express my sincere gratitude to all those who have supported me throughout the course of my research and the writing of this thesis.

First and foremost, I extend my sincere thanks to my supervisor and co-supervisor, Prof. ir. Ilias El Makrini and PhD ir. Gaoyuan Liu, whose guidance and feedback have contributed to the success of this work.

I would also like to thank my promotor Prof. ir. Bram Vanderborght and Prof. ir. Emanuele Garone for their input and support during the critical stages of this project.

I am also grateful to the BruBotics department for providing me with the resources to support my research.

I am forever thankful to my family and friends for their support and encouragement throughout my studies and life in general. To my parents, Marc and Kathleen, and to Ronja - thank you for your unconditional love and understanding.

Contents

List of Abbreviations

APF Artificial Potential Fields

BiRRT Bi-directional Rapidly-exploring Random Tree

CBS Conflict-Based Search

CT Conflict Tree

DH Denavit-Hartenberg

DL Deep Learning

DOFs Degrees Of Freedom

DRL Deep Reinforcement Learning

ICT Increasing-Cost Tree

ICTS Increasing-Cost Tree Search

IL Imitation Learning

LSPB Linear Segment with Parabolic Blends

LSTM Long Short-Term Memory

MAGS Multi-Agent Graph Search

MPNet Motion Planning Network

MRS Multi-Robot System

PRM Probabilistic Roadmaps

RL Reinforcement Learning

RRT Rapidly-exploring Random Tree

SAC Soft Actor-Critic

SOC Sum Of Costs

Chapter 1

Introduction

This introduction starts with the motivation for studying motion planning for an MRS consisting of several manipulators in Section 1.1. In Section 1.2, an overview is given of advancements in the field of motion planning for manipulators. From this, the objectives of the thesis and research questions are derived. These are presented in Section 1.3. Finally, an overview is given of how the thesis is organized in Section 1.4.

1.1 Motivation

For centuries, humanity has strived to create intelligent machines that can assist us in a variety of ways. As robotic systems become increasingly sophisticated, they are playing a more important role in society than ever before. From manufacturing and healthcare to exploration and entertainment, the introduction of robotics has revolutionized many industries. As we continue to push the boundaries of what robots can do, the interest in multiple manipulators working in collaboration has garnered significant interest. Large-scale tasks can be broken down into smaller subtasks, each handled by a different manipulator. This approach allows to perform subtasks concurrently, significantly reducing the time required to complete the overall task. Moreover, the demands of highly complex tasks can often surpass the capabilities of a single manipulator. By decomposing such tasks into more manageable subtasks and distributing them among multiple manipulators, these tasks can be accomplished through the cooperative efforts of the robots (Khamis et al., 2015). MRS are also more robust. If any individual manipulator fails, other units within the system can reconfigure to compensate for the loss, thereby maintaining progress on the task and minimizing disruptions (Reig et al., 2021). These are just a few of the many exciting advantages that MRS offer over a single manipulator, highlighting the significant potential of this field.

Collaboration and cooperation among multiple manipulators introduce significant challenges in motion planning, a critical component for operational autonomy and efficiency. Motion planning enables manipulators to navigate from an initial to a desired configuration, avoiding collisions with each other and environmental obstacles. As the number of manipulators (with partly overlapping workspaces) increases, it becomes increasingly complex to find collision-free paths for all. Each additional manipulator introduces extra Degrees Of Freedom (DOFs) and constraints to the motion planning problem, making scalability with the number of DOFs an important characteristic of an efficient multi-robot motion planning algorithm. When obstacles and task parameters can unpredictably change, the ability of motion planning algorithms to quickly adjust

paths is crucial, making computation time a vital factor. The space required for solving the motion planning problem can become the limiting factor when the available memory storage capacity is limited. In general, reliability is also a desired attribute of an algorithm, meaning a high success rate in finding a solution when one exists. Therefore, ideally, the algorithm provides some completeness guarantees. Lastly, the quality of the solution can be important as optimal solutions can minimize travel times or conserve energy. Scalability, computation time, space requirements, success rate, and quality of the solution are the metrics that characterize and define what will be called the efficiency of a motion planning algorithm in this work (see Section 5.4).

Improving the efficiency of motion planning algorithms is important for several reasons. Firstly, more efficient algorithms can substantially enhance the performance of existing systems. Additionally, they make it possible to deploy more sophisticated and larger-scale multi-manipulator setups, expanding the potential of what manipulators can achieve in collaboration. Furthermore, efficiency improvements can lead to significant reductions in operational costs. For example, by identifying more energy-efficient paths for manipulators, or by minimizing downtime during task transitions through faster computations and higher success rates.

1.2 State of the art

In Section 1.2.1 and Section 1.2.2, an overview is given of achievements in the field of motion planning for manipulators.

1.2.1 Motion planning for manipulators

Motion planning is concerned with autonomously determining the control input to generate the desired motion that satisfies certain constraints. In its most elementary form, a motion planning problem is issued with finding a collision-free motion among a set of static obstacles in the environment from an initial pose to a specified desired pose.

Motion planning involves the subproblems of path planning and trajectory generation. Path planning focuses on navigating through space within geometric constraints. The path provided by the path planner is a sequence of connected waypoints. Trajectory generation then adds a temporal dimension to this path. This process can be, but is not necessarily, carried out concurrently with path planning. Once the entire trajectory is planned, a control system must ensure that the manipulator meticulously follows the trajectory.

The recognition of the general motion planning problem as PSPACE-complete, as demonstrated by J.H. Reif (1979) and further substantiated by J.F. Canny (1988), underscores the inherent complexity of this domain. For manipulators with a high number of DOFs, the optimization-based (Ratliff et al., 2009; Schulman et al., 2014b) and sampling-based (L. Kavraki et al., 1996; S. LaValle & Kuffner, 2001) path planning approaches are regarded as the most applicable classical methods. In recent years, the research focus has shifted towards learning-based methods, relying on machine learning techniques, that aim to overcome limitations of the classical approaches and that show increased efficiency at solving motion planning problems compared to the classical methods. Three methods in this category that have had a lot of attention in the last years in particular are methods based on Reinforcement Learning (RL), Deep Learning (DL), or Imitation Learning (IL). This classification of motion planning methods is depicted in Figure 1.1.

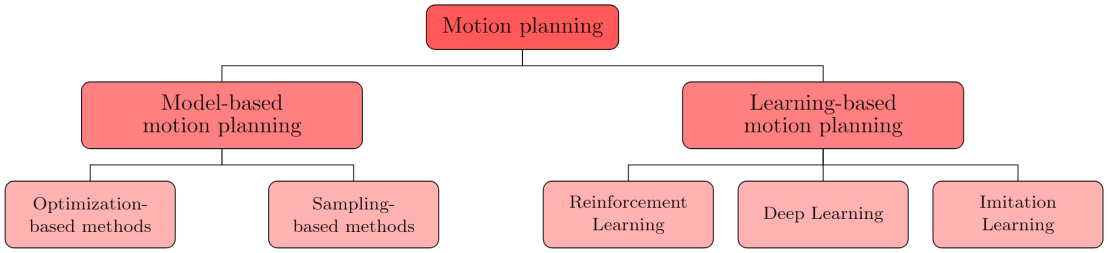


Figure 1.1: Classification of the most popular motion planning approaches for manipulators with a high number of DOFs. Recent advances in artificial intelligence have led to a significant increase in research and publications on learning-based approaches, which often build upon optimization- or sampling-based methods and demonstrate promising performance.

Combinatorial approaches

A first approach to motion planning is to find paths in the continuous configuration space without resorting to approximations (S. M. LaValle, 2006). Combinatorial motion planners do just that by using a complete algorithm for constructing a roadmap. Cell decomposition is a well-known combinatorial approach that, as the name suggests, decomposes the configuration space in cells. For two points within each cell, it is trivial to construct a path from one point to the other which is for instance the case when all the cells are convex. Knowing which cells are adjacent to one another and knowing which cells contain the initial and goal configuration of the robot, the motion planning problem is reduced to a graph search problem for which complete and optimal search algorithms exist. However, the solution is *resolution-complete*, implying that the discretization of the search space directly impacts the solution quality. More importantly, the computational cost of these search algorithms scales poorly with the dimension of the search space rendering combinatorial approaches impractical for high-dimensional configuration spaces.

Sampling-based approaches

Instead sampling-based approaches are widely used for motion planning of manipulators due to their effectiveness in navigating high-dimensional configuration spaces. These methods construct a path by performing random sampling within the configuration space and avoid an explicit construction of the free space within the configuration space. However, this computational advantage comes with the cost of giving up completeness for *probabilistic completeness*. Sampling-based methods are significantly influenced by the sampling distribution, casting uncertainty on both the initial solution quality and the convergence time to an optimal solution. Moreover, as these algorithms are probabilistically complete, there is no guarantee that a solution will be found within a finite amount of time. With increasing computation time, the likelihood of these algorithms failing to find a feasible solution, should one exist, diminishes asymptotically. Therefore, decision problem versions of these algorithms are semi-deciding (S. M. LaValle, 2006). The two most commonly applied sampling-based methods are versions of Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Tree (RRT). Because of the probabilistic nature, heuristics are often used to guide the search and decrease the amount of time needed to find a feasible solution, improving the efficiency of these algorithms (Cohen et al., 2014).

Optimization-based approaches

Optimization-based planners have also demonstrated efficacy in navigating high-dimensional spaces, as evidenced by the work of Marcucci et al. (2023). Their GCS planner, named after the underlying optimization framework of finding shortest paths in Graphs of Convex Sets, shows that convex optimization can be used to find higher-quality trajectories in less time than sampling-based planners. Another powerful method in terms of practical application is Artificial Potential Fields (APF). In this model, an attractive artificial potential field is defined around the goal, and repulsive potential fields around the obstacles. The negative gradient of this potential field is a resultant force applied on the robot that attracts the robot towards the goal and pushes it away from obstacles. The force points in the desired direction of motion and modulates the speed based on its magnitude. The potential energy can be minimized using a gradient descent algorithm and in doing so the robot reaches the goal configuration, that is if it does not get stuck in local minima. This concept of utilizing potential field functions for obstacle avoidance in robotic manipulators and mobile robots was introduced as early as 1989 (Warren, 1989).

Learning-based approaches

However, because of the high amount of computations for increasing dimension of the search space, these classical methods are outperformed by learning-based methods in many ways including memory usage and runtime. In recent years the development of artificial intelligence has become significant and the focus of research has shifted towards developing learning-based methods for addressing the drawbacks of the classical approaches (Tamizi et al., 2023). In these conventional approaches, the sampling of nodes is probabilistic or deterministic to ensure a uniform distribution across the state or configuration space. However, due to differential or collision avoidance constraints, the motion is often confined to certain regions within the state space. By devising non-uniform sampling strategies that favor sampling in those regions where an optimal solution might lie, the planning process can be accelerated. Thus, by biasing the sampling of sampling-based methods intelligently, learning-based methods can find feasible paths faster and solve the long runtime problem of sampling-based methods (Cheng et al., 2020; Ichter et al., 2018; Wang et al., 2020).

Deep Learning (DL) DL is a subset of artificial intelligence that mimics the brain's way of processing data and creating patterns for decision-making. It is mainly combined with sampling-based methods to improve their performance in motion planning. One of the drawbacks of sampling-based methods is the low convergence rate in the high-dimensional environment due to random sampling. Deep learning aims to solve this issue by sampling nodes in a way that decreases the planning steps. How to bias the sampling is learned by using data generated using conventional sampling-based methods.

Ying et al. (2021) developed a new method for planning the movement of a dual-arm robot, combining deep learning with a sampling-based planning strategy known as Bi-directional Rapidly-exploring Random Tree (BiRRT). Instead of randomly sampling nodes as traditional RRT does, they introduced the Long Short-Term Memory (LSTM) sampler. This technique uses an LSTM neural network to intelligently bias the sampling of nodes. The idea is that the LSTM, by learning from sequences of successful paths (trained using the RRT* algorithm, an optimized version of RRT), can quickly predict nodes that are likely to lead to a successful path. The LSTM needs to understand the environment to generate useful nodes. An autoencoder neural network is used to encode the environment (including obstacles) into a format the LSTM can use. Autoencoders compress data into a more manageable form while retaining its essential features.

The result of this work shows a significant improvement in convergence rate in comparison to classical approaches.

Another approach that shows strong performance in unseen environments is Motion Planning Network (MPNet) (Qureshi et al., 2020). It learns the path directly by combining two specialized neural networks. The first neural network, Enet, is responsible for encoding the environment. The idea is to convert complex environmental data into a more manageable format that still contains crucial information but that the second network can effectively use. The second neural network, Pnet, takes the encoded environment from Enet and uses it to generate a path from the starting node to the goal. MPNet has shown strong performance in complex and cluttered environments.

Creating a good data set is the main challenge for learning-based motion planners. The primary goal of such planners is to mimic the behavior of an oracle motion planner—that is, the algorithm responsible for data set creation. Therefore, selecting the right method for data set generation is crucial. Choosing an algorithm that generates the optimal path with respect to a certain metric, for instance, path length, leads to a better-performing learning-based planner. While most studies in the field have utilized RRT-based motion planners for data set generation due to their ability to explore space quickly, these methods often fall short in cluttered and complex environments in terms of planning time or path optimality.

Reinforcement Learning (RL) RL methods stand out as being particularly suited for planning in unknown environments. In RL-based methods, the robot learns to make decisions by interacting with the environment and learning from these experiences. In every attempt, the robot performs actions, observes the outcomes, and receives feedback in the form of rewards or penalties. This feedback guides the agent to understand which actions are beneficial and which are not, enabling it to optimize its behavior over time. The combination with deep-learning networks and their ability to process vast amounts of data and extract important features enables so-called Deep Reinforcement Learning (DRL) algorithms to handle high-dimensional continuous configuration spaces. The higher the dimensionality, the more difficult it is for regular DRL approaches to have sufficiently efficient exploration which is crucial for successful training. Soft Actor-Critic (SAC)-based planners explore the high-dimensional configuration space more quickly thanks to the use of an entropy term in the objective function. In the works of Prianto et al. (2020) and Ha et al. (2020), the SAC approach is leveraged for the motion planning of the multi-arm manipulator.

1.2.2 Motion planning for multiple manipulators

When regarding motion planning of multiple manipulators things become even more interesting as challenges arise associated with coordinating multiple autonomous agents[1] in a shared environment. During the previous discussion in Section 1.2.1, some examples of motion planning for multiple manipulators were already given. The optimization-based GCS planner proposed by Marcucci et al. (2023) has been demonstrated on a bimanual manipulation problem involving two KUKA LBR iiwa with seven degrees of freedom each, yielding an overall configuration space $\mathcal{Q}$ of $n = 14$ dimensions. Another example is the work of Schulman et al. (2014a) on motion planning with sequential convex optimization and convex collision checking. In their study, they considered motion planning for a PR2 robot consisting of two arms, a torso, and a base which equals 18 DOFs.

[1] Throughout this work, the terms robot and agent are used interchangeably.

The dependence of learning on sampling

Most of the attention in research nowadays goes to learning-based motion planning. Different planners of this type have shown to be applicable to multi-manipulator systems. Notable examples include the work of Ying et al. (2021), Ha et al. (2020), and Prianto et al. (2020), which are highlighted in the previous section. To recall, the work of Ying et al. (2021) explores DL techniques to learn a sampling distribution for a dual-arm robot. The sampling distribution is learned using the LSTM approach and this sampler is integrated in a centralized BiRRT algorithm. The studies by Prianto et al. (2020) and Ha et al. (2020) focus on decentralized path planning for multi-manipulator systems using DRL. In many cases, learning-based approaches are built upon sampling-based approaches, or sampling-based approaches are used to increase the performance of the learning-based approach. In the work of Ying et al. (2021), the actual planner is the BiRRT algorithm which plans paths in the composite configuration space of the dual-arm robot. Also in the work of Ha et al. (2020), BiRRT is used in a centralized way. For learning a decentralized motion planning policy, using multi-agent RL, BiRRT is used as an oracle planner on failed tasks, alongside the policy's own failed attempt, to solve the sparse reward problem. Therefore it is apparent that the performance of the overall planner also depends on the performance or efficiency of the sampling-based motion planner it builds upon. This sparks interest in improving the efficiency of sampling-based algorithms as well.

Sampling: coupled or decoupled?

The strategies for sampling-based motion planning for multiple robots can be categorized into coupled and decoupled approaches. Coupled planning approaches consider all robots as one system and planning happens in the composite configuration space of the robots. This allows for notions of completeness but suffers from scalability issues due to the significant growth of the dimension of the composite space with each additional robot. Decoupled approaches, on the other hand, plan for each robot independently or in smaller subgroups. They explore individual configuration spaces in isolation before combining them in a separate next phase. This improves scalability at the potential cost of suboptimality and loss of completeness guarantees. A comparison of the pros and cons of coupled and decoupled approaches in sampling-based motion planning for multiple robots is given in Table 1.1.

Table 1.1: Comparison of the pros and cons of coupled versus decoupled approaches in sampling-based motion planning for multiple robots. Coupled approaches plan in the composite configuration space and offer completeness guarantees but struggle with scalability. Decoupled approaches plan independently for each robot, improving scalability at the cost of potential suboptimality and loss of completeness guarantees.

	Coupled	**Decoupled**
Pros	• Probabilistically complete. • Suited for tightly coupled systems with significant dependencies.	• More scalable, less expensive. • Suitable for systems with less interaction between robots.
Cons	• Suffers from scalability issues due to the high-dimensional composite space.	• Lacks completeness guarantees. • Results in suboptimal paths.

About combining the best of both worlds

In the field of Multi-Agent Graph Search (MAGS), hybrid planners have been proposed which aim to combine the benefits of coupled and decoupled approaches. In a MAGS problem, the multiple agents operate in a discrete space, as opposed to the continuous space considered in the motion planning problem. Generally, these hybrid approaches address the motion planning problem in a decoupled manner whenever possible, only coupling the problem when necessary. Several algorithms have been proposed in the literature to solve MAGS problems completely and even optimally such as Increasing-Cost Tree Search (ICTS) (Sharon et al., 2013), CBS (Sharon et al., 2015), variants of A* (Standley & Korf, 2011) and M* (Wagner & Choset, 2015). However, an open research gap remains in applying hybrid approaches to sampling-based motion planners.

1.3 Objectives

The main objective of this work is to improve the performance and efficiency of sampling-based motion planners in multi-manipulator systems, particularly in scenarios where manipulators operate in close proximity. As highlighted in Section 1.1, efficiency includes various aspects, including success rate, computation time, memory usage, and the quality of the generated paths. Additionally, scalability with respect to the problem's dimensionality is a key consideration for high-dimensional motion planning problems, which is typical for multi-manipulator setups.

This study focuses on scenarios in which multiple manipulators perform pick-and-place tasks[2] in close proximity, such as in a large automated warehouse. In this context, the combination of scalability and completeness guarantees is essential. Scalability allows the system to manage the high-dimensional problem efficiently, enabling the deployment of numerous manipulators without overwhelming computational resources. Completeness guarantees ensure that, as the problem becomes increasingly complex and constrained with the addition of more manipulators, the system can still reliably find collision-free paths, even if this requires greater computational effort. This prevents bottlenecks or failures in the pick-and-place operations. Sampling-based motion planners can be broadly categorized into coupled and decoupled approaches. Coupled approaches generally provide higher success rates and better path quality due to their probabilistic completeness guarantees, but they often struggle with scalability. On the other hand, decoupled approaches are more scalable, requiring less computation time and memory, but they typically lack completeness guarantees and may result in suboptimal paths.

From this main objective, the following scientific questions arise:

- *Can a hybrid sampling-based motion planning approach combine the benefits of coupled and decoupled planning approaches?*
- *Can this hybrid approach achieve improved performance and efficiency compared to traditional coupled and decoupled sampling-based motion planning algorithms for multi-manipulator systems performing pick-and-place tasks in close proximity?*

To address these scientific questions, the following sub-objectives are identified:

1. **Develop a hybrid approach for motion planning:** Hybrid approaches in multi-agent graph search have shown promise in balancing the strengths of coupled and decoupled

[2]The focus is specifically on the motion from the pick position to the place position, rather than on the grasping of objects.

methods. This work aims to extend these hybrid strategies to the continuous domain of motion planning for multi-manipulator systems. The objective is to develop a hybrid algorithm that combines the scalability of decoupled approaches with the success rate and path quality benefits of coupled approaches.

2. **Develop a multi-robot motion planning library:** To facilitate the comparison of different motion planning strategies, a comprehensive multi-robot motion planning library is developed. This library includes algorithms tailored for both planar manipulators and robots simulated in PyBullet, providing a platform for benchmarking and evaluation of motion planning algorithms written in Python.

3. **Benchmark and assess the performance gains of the hybrid approach:** The final objective is to evaluate the performance improvements offered by the hybrid approach, particularly in terms of scalability, success rate, computation time, and path quality. This assessment will involve comparing the hybrid approach against both coupled and decoupled methods across a variety of setups.

1.4 Outline of the thesis

In Chapter 2, the motion planning problem is formulated. Additionally, this chapter describes how the motion problem can be solved using coupled and decoupled sampling-based planners. The hybrid approach will be built based on PRM (see Section 4.1) and also the coupled and decoupled approaches used for benchmarking the hybrid approach. Chapter 3 serves as our starting point for researching what the principles are behind hybrid search algorithms for the coordination of multiple agents on a discrete graph. In Chapter 4, the hybrid approach for addressing the continuous motion planning problem is discussed. In Chapter 5, the implementation of the algorithms is detailed as well as the tools used for benchmarking. In Chapter 6, the results are presented and discussed. Finally, Chapter 7 concludes this work.

Chapter 2

Motion planning

As already explained before, motion planning encompasses the subproblems of path planning and trajectory generation. The goal of motion planning is to find a feasible motion to let the robot perform a certain task. When at a higher level also tasks are planned, we speak of task and motion planning (Liu et al., 2023). In this chapter, basic concepts of motion planning are briefly described and the motion planning problem is formulated. The motion planning problem is first formulated for a single robot in Section 2.1.1 and then extended for multiple robots in Section 2.1.2. A useful concept is a configuration space that is typically used in motion planners. For multiple robots, this concept becomes the composite configuration space. This chapter concludes with how the motion planning problem for coordinating multiple robots can be addressed using sampling-based algorithms as a transition to the following chapters.

2.1 Problem formulation

First, the motion planning problem is formulated for a single robot in Section 2.1.1, followed by the reformulation of the problem for multiple robots in Section 2.1.2.

2.1.1 Planning the motion of a single robot

In motion planning, the robot moves in a workspace $\mathcal{W}$ which is most of the time in reality either a subspace of $\mathbb{R}^2$ or $\mathbb{R}^3$. The task of motion planning involves determining a motion for the robot to transition from its initial pose to a specified target pose. This motion must adhere to certain constraints, a primary one being the avoidance of collisions with any obstacles present in the robot's workspace. These constraints form the basic requirements for most motion planning problems. In addition to these constraints, also constraints regarding self-collision of robots and inter-robot collisions will be regarded in this work. For the robot, a configuration can be defined as a minimal set of parameters needed to completely specify the location in the workspace of every point of the robot. This means that each Degree Of Freedom (DOF) of the robot corresponds to one parameter of its configuration. For example, if the robot is a rigid body that moves in a 2-dimensional space, it can both translate and rotate in the plane. Therefore, the location of every point of the robot is known when three parameters are specified with respect to a frame of reference: two translational coordinates and one rotational coordinate. These three coordinates are in this case the parameters of the configuration of the robot. For a rigid-body robot that moves in a 3-dimensional workspace, the configuration consists of six parameters. Three for the position and three for the orientation of the robot.

The configuration space

The configuration space of a robot is the continuous space of all the configurations of that robot. For a robot with a configuration with d parameters, the configuration space is d-dimensional. Each translational parameter in the configuration of the robot defines a dimension in the configuration space that is an interval subset of $\mathbb{R}$. This translational DOF can take on any value within a certain range, corresponding to the physical limits of the robot's movement in that direction. On the other hand, the rotational parameters of the robot correspond to angles, which can vary from the lower to the upper joint angle limit. When no limits are specified for a joint angle, the rotational parameter is circular or periodic. For example, the configuration space of a planar manipulator with d revolute joints is a region on a d-dimensional torus. In most practical applications, the configuration space is higher dimensional than the workspace. The configuration space of a planar arm with a fixed base and two revolute joints is visualized in Figure 2.1.

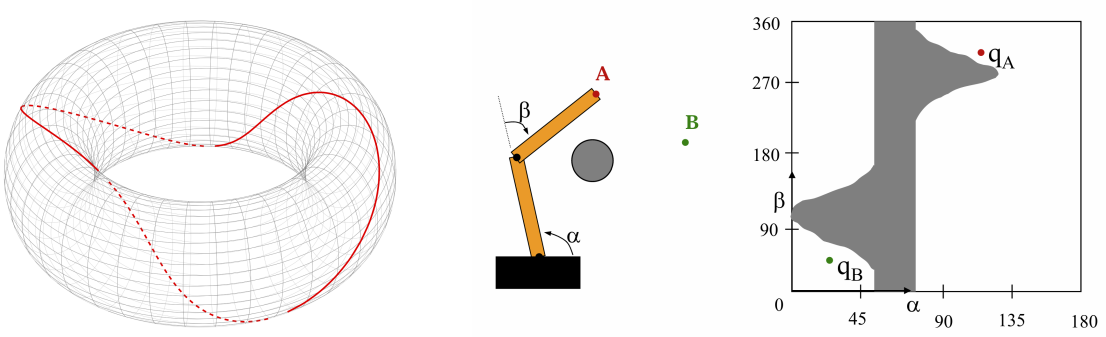


Figure 2.1: The left image shows a 2D torus representing the configuration space of the planar arm depicted in the center. On the right, the same configuration space is represented as a rectangle with wrapped edges, topologically equivalent to the torus. The grey area in the rectangle corresponds to the grey circular obstacle near the planar arm (Choset, 2000).

Instead of trying to find a collision-free motion directly in the workspace, it is possible to transform the motion planning problem to an equivalent problem in the configuration space. This allows us to treat the robot as a point in a d-dimensional space and the spatial aspect of the movement in the workspace becomes a space curve in the configuration space. This transformation of the problem to the configuration space offers a level of abstraction as different types of robots can be handled in the same way. Furthermore, as we will explore, the interaction between the robot as a point and the transformed constraints is simpler to characterize compared to the interaction within the workspace between the actual robot geometry and the constraints imposed by the obstacle geometry.

Path planning

Let us begin by formulating the path planning problem, which serves as the primary subproblem within the broader context of motion planning. The path planning problem is defined in the configuration space, starting with the transformation of constraints. For each configuration q, the robot occupies a set of points $\mathcal{R}(q)$ in a workspace $\mathcal{W}$. In the workspace $\mathcal{W}$, there are k static obstacles denoted by $\mathcal{O}_i$ with $1 \leq i \leq k$. For each $\mathcal{O}_i$, there exists a counterpart $\mathcal{O}'_i$ in the configuration space $\mathcal{Q}$. This static configuration space obstacle $\mathcal{O}'_i$ is defined as the set of all

configurations for which the robot and the obstacle $\mathcal{O}_i$ collide, i.e.

$$\mathcal{O}'_i = \{q \in \mathcal{Q} \mid \mathcal{R}(q) \cap \mathcal{O}_i \neq \emptyset\} \tag{2.1}$$

The *feasible* or *free* configuration space, $\mathcal{Q}_F$, is the subset of $\mathcal{Q}$ for which the robot does not collide with any of the static obstacles. If our robot consists of one rigid body, the free configuration space $\mathcal{Q}_F$ is given by

$$\mathcal{Q}_F = \mathcal{Q} - \bigcup_{i=1}^{k} \mathcal{O}'_i \tag{2.2}$$

The path planning problem can now be translated to finding a continuous path π from an initial configuration q_{init} to a goal configuration q_{goal} that lies completely in $\mathcal{Q}_F$, i.e. $\pi : [0,1] \to \mathcal{Q}_F$, where $\pi(0) = q_{\text{init}}$ and $\pi(1) = q_{\text{goal}}$. It must be noted that Equation (2.2) is no longer sufficient when our robot consists of multiple rigid bodies and collisions between the different bodies also have to be taken into consideration. When one of the obstacles $\mathcal{O}_j$, with $1 \leq j \leq k$, is dynamic and moves, then $\mathcal{O}_j$, $\mathcal{O}'_j$, and $\mathcal{Q}_F$ all become functions of time.[1] The path planning problem, also called the piano mover's problem, is visualized in Figure 2.2.

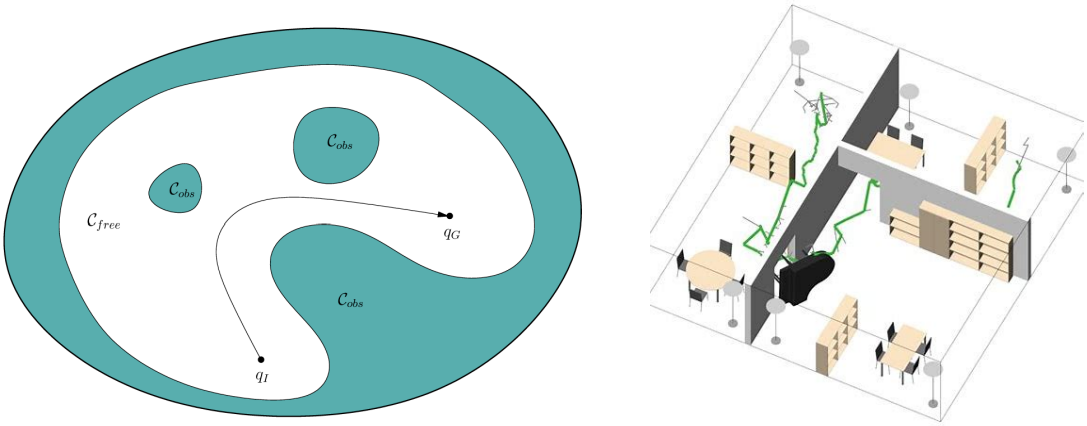


Figure 2.2: On the left, the path planning problem in the configuration space is conceptually visualized (S. M. LaValle, 2006). On the right, the position of a potential path is depicted for a piano moving through different rooms (Rickert et al., 2014).

This general formulation of the path planning problem has been recognized as PSPACE-complete, as demonstrated by Reif (1979) and further substantiated by Canny (1988). The best deterministic path planning algorithm known requires *exponential* time in the dimension of the configuration space $\mathcal{Q}$. This dimension increases fast even for relatively simple problems. The dimension is already equal to 6 for a rigid body in a 3-dimensional workspace. Articulation, which is the main interest of this work, typically adds many more DOFs. In addition, even simple obstacles in the workspace have complex counterparts in the configuration space. For robots with many DOFs it quickly becomes impractical to compute and represent $\mathcal{Q}_F$ explicitly.

[1] Apart from other robots, no dynamic obstacles will be regarded in the context of this thesis.

Therefore, the problem is approached with approximate or randomized methods in practice which trade away completeness for gains in efficiency. Approximate methods use regular and conservative subdivision of $\mathcal{Q}_F$ to recast the complex original problem as a search within space of many, simpler paths. Randomized methods are the sampling-based methods. They sample configurations in $\mathcal{Q}$, and evaluate whether or not they are collision-free and lie in $\mathcal{Q}_F$.

For sampling-based path planning, the path planning problem can be formally defined by a configuration space $\mathcal{Q}$, a binary constraint function $F : \mathcal{Q} \to \{0, 1\}$, a designated initial configuration $q_{\text{init}} \in \mathcal{Q}_F$ and a goal configuration $q_{\text{goal}} \in \mathcal{Q}_F$. This last constraint is often relaxed to a set of goal configurations $\mathcal{Q}_G \subset \mathcal{Q}_F$. The free configuration space $\mathcal{Q}_F$, is a subset of $\mathcal{Q}$ consisting of the configurations that satisfy the binary constraint:

$$\mathcal{Q}_F = \{q \in \mathcal{Q} \mid F(q) = 1\} \tag{2.3}$$

The aim is to find a continuous path $\pi : [0, 1] \to \mathcal{Q}$ such that $\pi(0) = q_{\text{init}}$, $\pi(1) \in \mathcal{Q}_G$ and for every $\lambda \in [0, 1]$, the condition $\pi(\lambda) \in \mathcal{Q}_F$ holds.

Trajectory generation

In the next phase of the motion planning problem, the path obtained from the path planner is transformed into a trajectory by generating a time schedule $\tau : [0, t] \to [0, 1]$. The time schedule τ dictates how to follow the path, considering constraints such as position, velocity, and acceleration. The motion μ is then obtained by composing the time schedule τ with the path π: $\mu = \pi \circ \tau : [0, t] \to \mathcal{Q}$, ensuring that for every $t' \in [0, t]$, the condition $\mu(t') \in \mathcal{Q}_F$ holds. Alternatively, trajectory generation can be performed concurrently with path planning. For example, the configuration space $\mathcal{Q}$ can be extended to include a time dimension. Consequently, the computed path must satisfy the additional constraint of being monotonically increasing with respect to this time dimension.

2.1.2 Towards coordinating multiple robots

Let us now consider the scenario in which a collection of robots share the same workspace $\mathcal{W}$. In the context of robot manipulators, each robot is a single rigid entity, denoted with the letter $\mathcal{R}$, composed of k interconnected links, $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k$. To clearly distinguish among the robots, we employ indices, such that the i-th robot is identified as $\mathcal{R}_i$. Assuming a total of m robots $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$, each one is associated with a configuration space $\mathcal{Q}_i$, an initial configuration $q_{\text{init},i}$, and a set of goal states $\mathcal{Q}_{G,i}$.

When regarding multiple robots, a *composite* configuration space X is defined that considers the configurations of all robots simultaneously. This space is the cartesian product of the configuration spaces of all the robots:

$$X = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \cdots \times \mathcal{Q}_m \tag{2.4}$$

A composite configuration $x = (q_1, q_2, \ldots, q_m) \in X$ specifies all robot configurations and the dimension of X is $N = \sum_{i=1}^{m} \dim(\mathcal{Q}_i)$.

A robot $\mathcal{R}_i$ can either collide with a static obstacle or with another robot $\mathcal{R}_j$. In the case of robot-obstacle collision, we have that $F(q_i) = 0$, independent of all other $m - 1$ robots, and in

the case of inter-robot collision, we have that $F(q_i, q_j) = 0$, independent of all other $m-2$ robots. The motion planning problem is now to determine a trajectory $\mu_i : [0, t] \to \mathcal{Q}_i$ with $\mu_i(0) = q_{\text{init},i}$ and $\mu_i(t) \in \mathcal{Q}_{G,i}$ for each robot $\mathcal{R}_i$ that successfully navigates around static obstacles and avoids inter-robot collisions:

$$\forall \lambda \in [0, t], \forall (i, j) \in \{1, \ldots, m\}^2 \mid i < j : \mu_i(\lambda) \in \mathcal{Q}_{F,i} = \{q_i \in \mathcal{Q}_i \mid F(q_1, \ldots, q_m) = 1\} \quad (2.5)$$

With the composite configuration $x = (q_1, q_2, \ldots, q_m)$ and the trajectories $\mu = (\mu_1, \mu_2, \ldots, \mu_m)$ the motion planning problem in the case of multiple robots can be equivalently stated as finding one continuous trajectory $\mu : [0, t] \to X$ such that $\mu(0) = x_{\text{init}} = (q_{\text{init},1}, q_{\text{init},2}, \ldots, q_{\text{init},m})$, $\mu(t) \in X_G$ and $\forall \lambda \in [0, t] : \mu(\lambda) \in X_F = \{x \in X \mid F(x) = 1\}$.

2.2 About solving the problem

With this last formulation, it is evident that the motion planning problem for multiple robots is in fact an ordinary motion planning problem. X can be regarded as an ordinary configuration space and motion planning algorithms established for a single robot can be applied without adaptation. The first motion planners were deterministic in nature (see Section 2.2.1). The problem with these approaches, whether exact or approximate, is their need for a large number of cells or points in the discretization. Additionally, complete algorithms require time that is at least exponential in dimension. This becomes a real issue with increasing dimension of $\mathcal{Q}$, rendering them impractical for high-dimensional spaces. Sampling-based methods were developed to alleviate this issue by exchanging (resolution-) completeness for probabilistic completeness (see Section 2.2.2). Sampling-based algorithms are found to scale better in practice with the dimension of the configuration space, however, when the dimension of X is excessively high, applying the algorithm in a coupled fashion (i.e. applying it directly on X) might still be an approach with no practical relevance.

2.2.1 Deterministic motion planners

For deterministic planners a distinction can be made between combinatorial motion planners and approximate motion planners. Combinatorial motion planners are exact algorithms that find paths through the continuous configuration space without resorting to approximations (S. M. LaValle, 2006). They are complete, meaning that for any problem instance, the algorithm will either find a solution or will correctly report that no solution exists. On the other hand, approximate motion planners do make certain approximations to trade off exactness for computational efficiency in some way.

2.2.2 Sampling-based motion planning

In search of more efficient algorithms to solve the motion planning problem, sampling-based algorithms have been developed. One of the first works in motion planning incorporating sampling was the works of Barraquand and Latombe (1990). Their motion planner is an improved version of the well-known potential field approach. This approach has the disadvantage of getting stuck in local minima. In order to address this problem, the algorithm progressively constructs a graph that links the local minima of the potential function. This graph is searched until a goal configuration is reached. One local minimum is linked to another by performing a random motion

that allows to escape from the first minimum's well, followed by a negative gradient motion of the potential function towards the other minimum. The graph of local minima is explored in a depth-first approach with random backtracking. Subsequent to this research, various other sampling-based motion planning algorithms have been created, all sharing the common characteristic of estimating the connectivity of the free configuration space by means of sampling. Many of these algorithms possess the property of probabilistic completeness, signifying that given infinite time, they are guaranteed to find a solution. An overview of important sampling-based path planners that have been developed over the years, is given in Table 2.1.

Table 2.1: Overview of relevant and state-of-the-art sampling-based path planners, organized by year of publication and categorized by their underlying structure. This overview highlights that sampling-based path planners are either graph-based or tree-based.

Planner	**Year**	**Structure**	**Description**
RPP	1990	APF	Randomly escapes local minima of APF using random walks.
PRM	1996	Graph	Constructs a graph connecting randomly sampled configurations.
EST	1997	Tree	Constructs trees by connecting nodes to maximize exploration of the search space.
RRT	1998	Tree	Grows a tree connecting randomly sampled configurations.
Ariadne's Clew	1998	Tree	Combines minimization for finding a path and maximization for local exploration and escaping local minima.
Lazy PRM	2000	Graph	Creates a graph, deferring collision checking.
RRT-Connect	2000	Tree	Connects two trees grown from both start and goal configurations.
PRM*	2011	Graph	Asymptotically optimal variant of PRM that improves path quality.
RRT*	2011	Tree	Asymptotically optimal variant of RRT that improves path quality.
SPARS	2014	Graph	Similar to PRM* but is asymptotically near-optimal and has a higher convergence rate.
FMT*	2015	Tree	Grows a tree using lazy dynamic programming, blending features of RRT and PRM.
BIT*	2015	Tree	Heuristic search on increasingly dense implicit graphs, reusing information.
AIT*	2020	Tree	Adaptive bidirectional search using problem-specific heuristics.

As can be seen from Table 2.1, there are two main categories when it comes to sampling-based motion planning. One category is graph-based planners, like the Probabilistic Roadmaps (PRM) (L. E. Kavraki et al., 1996), which builds a graph connecting randomly sampled configurations

(see Section 4.1). Several planning queries can then be answered using the same graph which makes them suited for multi-query problems. The other category is tree-based planners for which the Rapidly-exploring Random Tree (RRT) algorithm is the representative algorithm (S. LaValle, 1998). It grows a tree from a start configuration by randomly sampling configurations and connecting them to the tree until the goal is reached. The tree structure only allows the specific query for which it is grown to be solved. Given that we focus on pick-and-place tasks where start and goal poses can vary between tasks and roadmaps constructed for a specific manipulator can be reused multiple times, graph-based planners like PRM are more advantageous. Many state-of-the-art sampling-based motion planners are available in the Open Motion Planning Library (OMPL) (Şucan et al., 2012). The PRM and RRT algorithms are visualized in 2D in Figure 2.3.

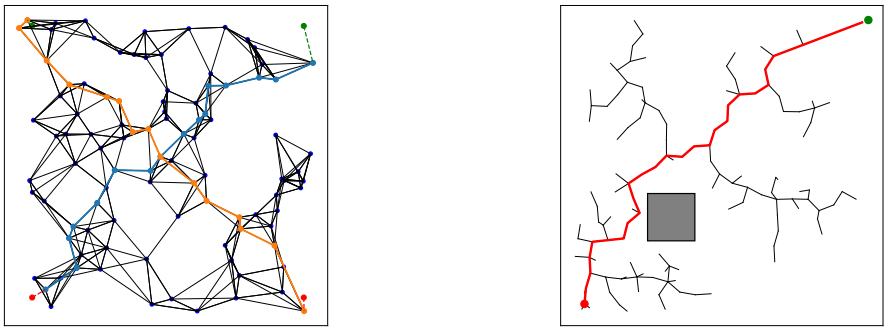


Figure 2.3: On the left, the Probabilistic Roadmaps (PRM) algorithm builds a graph by connecting randomly sampled configurations in the configuration space. This graph supports multiple planning queries. On the right, the Rapidly-exploring Random Tree (RRT) algorithm incrementally grows a tree from the start configuration by connecting random samples until the goal is reached. This approach does not support multiple queries with the same tree.

2.2.3 Multi-robot sampling-based approaches

For solving the motion planning problem including the coordination of the movements of multiple robots, as formulated in Section 2.1.2, sampling-based algorithms can be applied to the composite configuration space without adaptation. In this work, this is called the coupled approach. While the motion planning of a Multi-Robot System (MRS) can be addressed similarly to that of a single robot, there exist notable differences that have to be discussed. The motions of the robots can be decoupled, which allows us to initially plan for each robot independently and subsequently consider the interactions between these partial plans to generate the solution. A comparison of the main advantages and disadvantages of coupled and decoupled approaches is given in Table 1.1. Two decoupled approaches will now be discussed.

Decoupled sampling-based approaches

Decoupled approaches design movements in environments where multiple robots operate simultaneously by splitting the problem in two stages. In the first stage, they focus on planning the path of each individual robot without considering potential interactions or collisions with other robots.

This initial phase allows for simplification of the planning process since each robot's movement is planned (and possibly optimized) in isolation, which is computationally less expensive and simpler to design and implement.

The coordination of the robots, which is the actual challenge of the multi-robot motion planning problem, takes place in the subsequent phase. Since the initial planning was done in isolation, the options available to coordinate the motions of the robot are already constrained by the paths designed in the first stage. By constraining the motion of each robot to a certain path in its joint space, the dimension of the search space is reduced to one. If these constraints do not admit a solution, these approaches are unable to reverse their commitments. In other words, decoupled approaches are generally incomplete.

Despite these challenges, decoupled approaches remain valuable in practical applications (Čáp et al., 2015; Zhang et al., 2022). This is because of the significant reduction of the computational complexity and computation time required to plan motions in multi-robot systems and because of the fact that completeness can in some cases be recovered by careful design of the motion planner and planning problem. For instance, completeness can be fully recovered in the case of fixed-roadmap coordination when each robot is given a *garage* configuration and the planner is complete (S. M. LaValle, 2006). For a configuration to be a garage, the robot should be impossible to collide with in this configuration and the garage configuration should be reachable from any other configuration.

Prioritized planning A first decoupled approach to motion planning for multiple robots is prioritized planning (Erdmann & Lozano-Perez, 1987; S. M. LaValle, 2006; Van Den Berg & Overmars, 2005). It was introduced by Erdmann and Lozano-Perez in 1987. In this approach, each of the robots is assigned a priority and the planning of the robots is done sequentially in order of decreasing priority. Each trajectory is planned so as to avoid collisions with the static obstacles as well as the higher priority robots which are in this sense regarded as moving obstacles. So prioritized planning requires both a motion planner for a single robot in a known dynamic environment and a prioritization scheme. There are numerous methods to assign priorities among robots, guided by various logical justifications. For example, the knowledge of the importance of the tasks each robot is performing or the knowledge of their respective starting times can be used to base the prioritization on. With n robots in the system, it's possible to create $n!$ distinct sequences of priorities. The way in which these priorities are assigned can greatly affect how optimal the resulting path is, as illustrated in Figure 2.4.

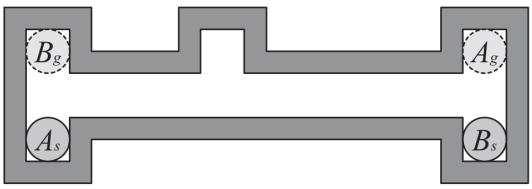


Figure 2.4: With robot A having the highest priority, robot B can only start moving after A has reached its goal. With robot B having the highest priority, robot A can use the cavity in the passageway to let robot B pass, giving a much better result. This example has been adapted from the works of Van Den Berg and Overmars (2005).

This demonstrates that the prioritization process can be a critical factor for the overall performance and optimality of the robotic system's path planning. This same figure also shows how completeness is lost. If in this figure the goals for robots A and B would coincide with the starting position of the other robot then prioritized planning would yield no solution even though there are solutions. Ways of prioritization will however not be further discussed and the priorities of robots will assumed to be sorted in a randomized way. Assume the n robots are sorted as $\mathcal{R}_1, \ldots, \mathcal{R}_n$, with $\mathcal{R}_1$ the robot assigned with the highest priority. The prioritized planning framework works inductively by computing a collision-free path, $\pi_1 : [0,1] \to \mathcal{Q}_{F,1}$ for $\mathcal{R}_1$. Next, a timing function is computed, $\tau_1 : [0,t] \to [0,1]$, for π_1, to yield $\mu_1 = \pi_1 \circ \tau_1 : [0,t] \to \mathcal{Q}_{F,1}$. Now suppose that the collision-free motions $\mu_i : [0,t] \to \mathcal{Q}_{F,i}$ have been computed for the first k robots, i.e. i from 1 to k. The prioritized planning framework continues inductively by formulating the k higher priority robots as moving obstacles in the workspace $\mathcal{W}$ with their motions being $\mu_1, \ldots, \mu_k$. Now again, a path π_{k+1} and a timing function τ_{k+1} are computed to design a motion $\mu_{k+1} = \pi_{k+1} \circ \tau_{k+1}$ using a time-varying motion planner. An important characteristic of prioritized planning is that the timing function τ_{k+1} is designed concurrently with the spatial path π_{k+1}, essentially planning a spatio-temporal path at once. In other words, path planning and coordination happen together. This approach differs significantly from treating the path planning and designing the timing function as distinct stages within the planning process. Viewing them as separate steps implies that during the spatial path's design, the robot's timing or temporal location is unspecified. Consequently, coordination, which involves assigning a time function, occurs only after the spatial path is defined. Essentially, the coordination becomes a velocity-tuning problem. This method is known as fixed-path coordination. This approach constrains the problem even more than prioritized planning. The idea of pre-planned paths or roadmaps for all robots independently will be the main idea behind the sections on fixed-path coordination and the hybrid approach using Conflict-Based Search (CBS) (see Chapter 4).

Fixed-path coordination As already touched upon in the last paragraph, the idea behind fixed-path coordination is to constrain each robot $\mathcal{R}_i$ to follow a predetermined path $\pi_i : [0,1] \to \mathcal{Q}_{F,i}$ and coordinate the robots in a following phase by tuning their velocities (S. M. LaValle, 2006). The paths are computed using any ordinary path planning algorithm to only avoid collisions with static obstacles. Next, the timing functions need to be determined to coordinate the motion of the robots. This can also be defined as a search problem in a n-dimensional space where n equals the number of robots. This space is called the *coordination space*, denoted with the letter $\mathcal{S}$. This space represents time implicitly and once a path is computed in this space, explicit timing functions are derived from this computed path. For n robots, $\mathcal{S}$ is an n-dimensional unit cube $\mathcal{S} = [0,1]^n$. At state $(0, \ldots, 0) \in \mathcal{S}$, every robot $\mathcal{R}_i$ is in its initial configuration $\pi_i(0)$ and at state $(1, \ldots, 1) \in \mathcal{S}$, every robot $\mathcal{R}_i$ is in its goal configuration, $\pi_i(1)$. The obstacle space $\mathcal{S}_{\text{obs}}$ in the coordination space $\mathcal{S}$ is defined by the robot-robot collisions when the robots move along their paths. The objective is thus to find a continuous path $\sigma : [0,1] \to \mathcal{S}_F$, in which $\mathcal{S}_F = \mathcal{S} \setminus \mathcal{S}_{\text{obs}}$, for which $\sigma(0) = (0, \ldots, 0)$ and $\sigma(1) = (1, \ldots, 1)$. The path σ does not need to be monotonic, in contrast to prioritized planning, and ordinary path planning algorithms can be applied to the coordination space. The easiest approach is to place a grid over $\mathcal{S}$ and use classical search algorithms to find a collision-free path. If the dimension of $\mathcal{S}$ is very high, sampling-based approaches can be used instead. When a path in the coordination space $\mathcal{S}$ is found, the relative movement of all the robots along their paths is fixed. Now, the time interval in which all motions are executed has to be assigned. The final motions of the different robots are found by applying a time mapping $\tau : [0,t] \to [0,1]$ to the path $\sigma : [0,1] \to \mathcal{S}_F$. For a certain time $t' \in [0,t]$ the configuration of robot $\mathcal{R}_i$ is given by the projection of σ on the domain $\mathcal{S}_i = [0,1]$ corresponding to that robot.

2.3 Conclusion

Motion planning involves path planning and trajectory generation. The challenge of finding feasible and collision-free motions for m robots navigating among k static obstacles can be reduced to finding a single feasible and collision-free motion for the MRS as a unified entity. This approach, referred to as the coupled approach, requires performing path planning in the composite configuration space of the robots, where the dimensionality equals the total number of DOFs of the MRS. However, the size of the composite configuration space increases exponentially with its dimension. Therefore, it quickly becomes intractable to use conventional sampling-based path planners to plan paths in the composite configuration space. To address the scalability issues associated with coupled approaches, it is crucial to decouple the motion planning problem by independently planning the paths for each robot and considering interactions between the robots at a later stage. Decoupled methods offer significantly better scalability with increasing problem complexity, but they sacrifice (probabilistic) completeness guarantees, which means they may not always find a solution when one exists. Since our focus is on pick-and-place tasks where we can reuse roadmaps constructed for a specific manipulator, graph-based planners like PRM are preferred over tree-based planners. This, combined with the importance of scalability in MRS applications and the need for completeness guarantees, underscores the relevance of the next chapter, which delves into the field of multi-agent graph search. Recent advancements in this area include hybrid approaches that address the motion planning problem in a decoupled manner whenever possible, only coupling the problem when necessary.

Chapter 3

Multi-agent graph search

In this chapter, multi-agent graph search is explored with the aim of integrating such an algorithm with a graph-based motion planner like PRM.

3.1 Problem formulation

A Multi-Agent pathfinding (MAPF) or Multi-Agent Graph Search (MAGS) problem involves identifying paths for multiple agents within a shared graph, connecting start and goal nodes. The objective is to ensure that the agents do not collide when traversing these paths. In essence, the underlying idea of the multi-agent graph search problem aligns with that of the multi-agent motion planning problem. The key distinction between both lies in the nature of the spaces they operate in. The graph search problem exists within a discrete space, while the motion planning problem unfolds within a continuous configuration space. By constructing a graph through sampling, Probabilistic Roadmaps (PRM) bridges this gap, essentially transforming the motion planning problem into a discrete graph search problem. This approach allows for the application of graph search techniques to solve the motion planning problem.

In the graph search problem, the undirected graph $G = (N, E)$ consists of nodes N connected to each other by edges E. Every node in N is a location that can be occupied by an agent traversing the graph and every edge in E connects two nodes for which an agent can move from one to the other. A solution to the multi-agent graph search problem is a plan that successfully navigates all agents from starting nodes to their goal nodes without any collisions occurring. In general, more than one valid solution can be found. In some cases, we are interested in finding a valid solution that optimizes a secondary objective. This secondary objective is either to minimize the *makespan* of the solution or the *sum of costs*. The makespan of a solution μ is the time required for all agents to reach their goal nodes

$$MS(\mu) = \max_{1 \leq i \leq k} |\mu^i| \tag{3.1}$$

The sum of costs of a solution μ is the sum of the costs of the actions taken by each agent as they traverse their respective paths

$$SOC(\mu) = \sum_{1 \leq i \leq k} |\mu^i| \tag{3.2}$$

In a state graph, the cost of actions typically represents the effort or resources required to transition from one state to another. In the pathfinding problem, this is typically the duration of

an action. Similar to what is discussed in Section 2.1.2, the multi-agent graph search is equivalent to a single-agent graph search in the composite graph of the k agents, which we will also denote by X. A node in X now represents the vector of k nodes occupied by the agents at a particular time step. An edge now also represents a vector of k actions performed by the agents to move to an adjacent node in X. Thus once again, single-agent search algorithms can be leveraged to solve the multi-agent problem.

3.2 The computational issue of the coupled approach

The computational expense of solving the shortest path problem, such as with the A* algorithm, is largely dependent on two factors: the *size* of the search space and its *degree* or *branching factor*. The size of the search space corresponds to the number of nodes in the graph. In the composite graph X, each node corresponds to a vector of k nodes in the shared graph G. Therefore, with $|N|$ the size of G, the size of X is $|N|^k$. The branching factor of G is $\frac{|E|}{|N|}$ and the branching factor of X is $\left(\frac{|E|}{|N|}\right)^k$. Both these values increase exponentially with the number of agents, making the performance of the search algorithm scale poorly with the number of agents to coordinate.

To have an idea of the significance of the scalability issue, let's regard the simple case of a 4-connected grid with 100 by 100 nodes. The size of the shared graph is thus 10000 and the branching factor is approximately 4. Say we have 5 agents we wish to coordinate on the shared graph. The size of X is then 10^{20} and the branching factor becomes approximately $4^5 = 1024$. Especially the exponential branching factor poses a significant challenge for A*. This is because at least all nodes along an optimal path have to be expanded. Moreover, the expense of the expansion of a node is at least linear in relation to the branching factor. Therefore, the application of the coupled approach, in which A* searches the shortest path on the composite graph, is limited to a low number of agents. Thus the A* algorithm search on the composite graph is not a suited approach for coordinating multiple agents on a shared graph. By analogy, this discussion also illustrates the scalability issue with the coupled approach of applying motion planners directly in the composite configuration space.

3.3 Decoupling to solve the scalability issue

The classical approach of dealing with the scalability issue is to decouple the multi-agent graph search into single-agent pathfinding problems. Undoubtedly the most used approach to do so is the prioritized approach because of its simplicity and efficiency. For this approach, each agent is assigned a priority and subsequently shortest paths are found for each agent in order of priority avoiding conflict with higher-priority robots. However, what is fundamentally different from finding the shortest path on a graph is that for coordinating the agents time is explicitly taken into account and next to moving to an adjacent node, agents are also allowed to wait for a time step at the current node. The optimal path for an agent is no longer the shortest path in length but in duration. This is however equivalent to the shortest path in a time-expansion graph (Stern, 2019). To transform the graph G in the time-expansion graph G_t, every node n is augmented with a time dimension, resulting in nodes (n, t) where, without loss of generality, t takes on non-negative integer values, starting from 0 and increasing indefinitely. A node n_2, t_2 is a neighbor of node n_1, t_1 if $n_1 = n_2$ and $t_2 = t_1 + 1$. The size of G_t is equal to the number of nodes multiplied by the upper bound T of the makespan of the solution: $|N| \times T$. The branching factor of G_t is $\left|\frac{|E|}{|N|}\right|^{1/T}$. To see how prioritization reduces the computational expense, let's regard our

100 by 100 4-connected grid again. With 5 agents traversing the graph and assuming $T = 1000$, the size of the search space is 10^7. The branching factor is equal to 5 as 4 edges connect the node to its adjacent nodes at the next time step and one edge for a wait action connecting the node with itself at the next time step. This is well within the capabilities of A* to solve as it has been applied to much larger search spaces. The main drawback of decoupled approaches is that they are neither optimal nor complete.

3.4 Combining completeness and scalability

Recent research focuses on developing algorithms combining the completeness guarantees of coupled approach with the decrease in computational expense, or equivalently the increase in scalability, of decoupled approaches (Stern, 2019). This leads to a class of hybrid approaches. In this section, some of these hybrid approaches will be discussed further.

3.4.1 Operator Decomposition

The Operator Decomposition (OD) method, proposed by Standley (2010), is a variant of the A* algorithm. When A* is applied directly to the k-agent composite graph, every pair of neighboring nodes can differ in more than one action. In other words, multiple agents may move simultaneously from one node to another in the composite graph, leading to an exponential branching factor. The OD algorithm mitigates this issue by evaluating the actions of the agents one at a time, in an a priori defined arbitrary order, rather than simultaneously. The search begins by expanding the source node, considering only the action of the first agent. This results in a set of neighboring nodes where all other agents remain at their locations at time step 0, and only the location of the first agent potentially changes at time step 1. The search process continues by expanding these new nodes, considering only the actions of the second agent, and so on. Once this process is completed for all agents, the nodes representing the potential location for all agents at time step 1 are obtained. These nodes are classified as *full* nodes, as opposed to *intermediate* nodes that exist between two time steps. In this manner, each k^{th} descendant of a full node is another full node at the next time step. Through this process, the OD algorithm effectively reduces the exponential branching factor to the value of the branching factor of a single agent. However, the solution becomes k times deeper as there are k intermediate nodes between any pair of full nodes. This trade-off is typically beneficial as a high heuristic value for an intermediate node can prevent the expansion of the entire tree from one full node to the next.

3.4.2 Simple Independence Detection

Another variant of the A* algorithm, also proposed by Standley (2010) in the same work, is the Simple Independence Detection algorithm. In the initial phase of this algorithm, each agent independently formulates an optimal path for itself, completely ignoring the presence of other agents. This is followed by a check for conflicts on the paths found. If a conflict arises between agents, these agents are combined into a single entity. This single entity is called a *group* in this work, which is equivalent with the term *meta-agent* used later on. The A* algorithm, enhanced with Operator Decomposition (OD), is then used to cooperatively plan a solution for this group, while still ignoring the presence of all other agents. This process is iterative, with each iteration identifying the first conflict that arises in time, merging the conflicting (meta-)agents, and then solving them using OD enhanced A*. The process continues until there are no conflicts left between the paths of the agents. The worst-case scenario would occur if all the agents are

merged into one single meta-agent for the whole duration of the search. This is the case when for the initial solution all the agents are in conflict with each other at the first time step.

3.4.3 M*

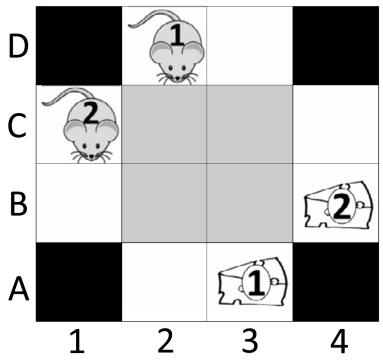
The idea behind the M* algorithm for solving conflicts is similar to the principle behind Simple Independence Detection. M* addresses the exponential branching factor of the k-agent search space in which it operates by dynamically merging agents into meta-agents (Wagner & Choset, 2011). Initially, the agents follow their individually optimal paths determined with A*. During the search of these individual paths, it may happen that multiple agents are in collision at a certain time step. This state is discarded and the colliding agents are merged into a meta-agent for which the coupled A* search will find a valid state for the colliding agents at the next time step. When all agents do not collide anymore the search is fully decoupled again and each agent follows their individually optimal path. The search space of M* can therefore be thought of as being one-dimensional in the composite graph, but with a dynamically growing dimensionality around states for which the agents are in collision.

3.4.4 Increasing Cost Tree Search

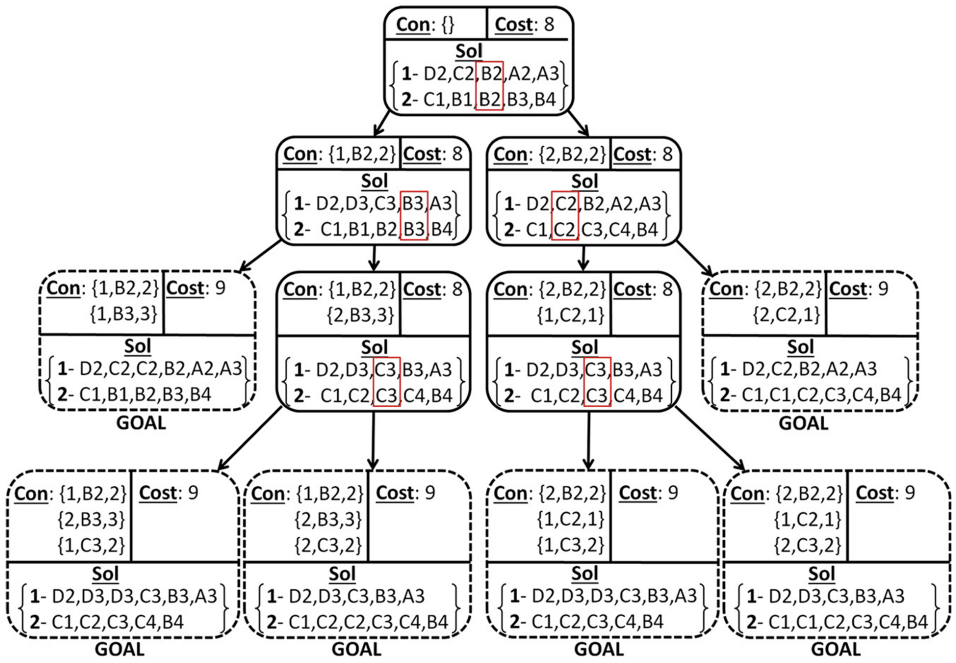
The Increasing-Cost Tree Search (ICTS) operates on two levels for finding an optimal path that minimizes total travel cost for multiple agents and avoids collisions (Sharon et al., 2013). In the ICTS approach, a high-level search happens on an Increasing-Cost Tree (ICT). Each node in the ICT consists of a k-vector of individual path costs, representing all possible solutions where the cost of the individual path of each agent is exactly a certain value. The low-level search for an ICT node essentially performs a goal test to determine if there is a valid solution that is represented by this ICT node. This goal test comprises of first enumerating all the possible individual paths with a cost equal to the cost in the ICT node and secondly iterating over all possible combinations of the individual paths until a solution is found. In the work of Sharon et al. (2013), they also present a low-level search algorithm for doing this. Nodes in the ICT on the same level have the same total cost and by performing a breadth-first search on the ICT the solution found is ensured to be an optimal one.

3.4.5 Conflict-Based Search

Conflict-Based Search (CBS) is a highly scalable and recently developed optimal multi-agent pathfinding algorithm. It is notable for its application in graph search scenarios involving high numbers of agents. Given the relevance and underlying principles of CBS to our proposed hybrid sampling-based motion planner, it deserves detailed consideration. For a more in-depth exploration of CBS and its algorithmic foundation, readers are encouraged to consult the original work by Sharon et al. (2015). CBS is similar to ICTS in that it also formulates the problem into a search process with two levels. High-level, a Conflict Tree (CT), which is a binary tree, is grown and searched. Each of the CT nodes consists of three things: a set of constraints, a solution, and a total cost. For a given CT node, the low-level search algorithm computes the solution and total cost based on the set of constraints of the nodes. This can be any search algorithm that returns the optimal path for an agent a_i such as A*. An example of the working principle behind CBS is illustrated in Figure 3.1.



(a) A simple example for showcasing the principles behind CBS.



(b) The CT constructed by CBS for solving the traversal problem shown in Figure 3.1a.

Figure 3.1: Illustration of the working principle behind the Conflict-Based Search (CBS) algorithm, adapted from the original paper on CBS by Sharon et al. (2015). The top figure shows the initial setup with agents (the mice), obstacles (the black squares), and the goal nodes (the cheese). The bottom figure depicts the Conflict Tree (CT) used by CBS. The root node of the CT contains no constraints, allowing the low-level search to compute individually optimal paths in a decoupled manner. Conflicts between agents' paths are highlighted in red. The algorithm resolves these conflicts by creating child nodes with additional constraints, ensuring that agents do not occupy the same node or swap nodes at the same time. This process continues until a conflict-free solution is found, as indicated by the GOAL statement. By only considering interactions at the higher level of the CT, CBS balances decoupling and coupling of the graph traversal problem, providing a scalable and complete solution for Multi-Agent Graph Search (MAGS).

The search starts with the root node of the CT. This node contains no constraints. The low-level search thus computes the individually optimal paths in a completely decoupled way. The total cost is simply the sum of the individual path costs. The algorithm proceeds with checking the solution for the first conflict that occurs in time between the individual paths of agents, which are highlighted in red in the CT of Figure 3.1. There are two types of conflicts. Node conflicts $C = (a_i, a_j, n, t)$ signify that at time t the agents a_i and a_j occupy the same node n. Edge conflicts $C = (a_i, a_j, n_1, n_2, t)$ signify that the two agents a_i and a_j swap nodes. For time step t to time step $t + 1$, agent a_i moves from node n_1 to node n_2 while agent a_j moves from node n_2 to node n_1. In the high-level CT, edge conflicts can be treated in the same way as node conflicts and therefore the general term conflicts will be used from now on to address both. If no conflicts are found in the solution of the CT node, we know that this is a solution to the multi-agent graph search problem. This results in a GOAL statement underneath the CT node in Figure 3.1. If a first conflict $C = (a_i, a_j, n, t)$ has been found, we know that a valid solution requires that at most one of the agents a_i and a_j occupies node n at time t. Therefore two child nodes are created that both inherit the constraints of the parent node and take one additional constraint into account to specifically resolve the conflict C. One child node takes into account the constraint (a_i, n, t) and the other one the constraint (a_j, n, t). A constraint (a_i, n, t) implies that agent a_i is prohibited from occupying node n at time t. But what if for a given time step more than two agents collide? There are two ways to handle such k-agent conflicts with $k > 2$. The straightforward option is to create k child nodes for the CT node, each of which adds a constraint to $k-1$ agents so that only one agent is allowed to occupy the vertex n at time t. The second option is to only focus on two agents that are found to conflict and apply a constraint accordingly, leaving further conflicts for deeper levels of the tree. This way the binary nature of the tree is maintained. These two options are further equivalent and will lead to the same solution. By taking into account all possibilities of resolving a conflict in separate CT nodes, representational optimality is guaranteed. Regarding the implementation, rather than storing all the constraints for each node, only the latest constraint has to be stored for a node. The other constraints can be extracted by traversing the path to the root via its ancestors. Additionally, taking a new constraint into account will only influence the path of the agent to which the newly added constraint applies. To all the other paths the same constraints apply as specified by the parent node. Therefore the low-level search should only recompute the path for the agent a_i to which the constraint applies and all the other paths can be taken over from the solution computed for the parent node.

3.5 Conclusion

With the focus on combining scalability and completeness guarantees into a single motion planner, CBS emerges as a promising MAGS algorithm to extend to the continuous motion planning problem. The most natural way to achieve this is by transforming the continuous motion planning problem into a discrete one using a graph-based sampling-based motion planner like PRM, and then applying CBS to this discrete problem. The next chapter will elaborate on how to combine PRM with CBS to develop a hybrid coupled-decoupled motion planner.

Chapter 4

Conflict-Based Search on Probabilistic Roadmaps

Conflict-Based Search (CBS) is a Multi-Agent Graph Search (MAGS) algorithm, known for its effectiveness in handling dense scenarios of multiple agents navigating within a shared graph. It coordinates agents by computing paths independently, detecting conflicts, and resolving them with constraints, as discussed in Section 3.4.5. Each new constraint requires re-planning the affected agent's path. Given the iterative nature of conflict detection and the subsequent need for frequent re-planning, graph-based motion planners like Probabilistic Roadmaps (PRM) are particularly compatible with CBS. PRM constructs a roadmap of potential paths through the configuration space, effectively estimating its connectivity. The methodology for constructing meaningful roadmaps is discussed in Section 4.1. The integration of CBS with PRM for sampling-based motion planning is detailed in Section 4.2. Although this approach was previously explored by Solis et al. (2021), it hasn't been extensively tested for multi-manipulator setups performing independent pick-and-place tasks. After planning the paths, they need to be converted into trajectories that the manipulators can effectively follow, as detailed in Section 4.3.

4.1 Probabilistic Roadmaps

The first step in solving the path planning problem for multiple manipulators with CBS is to recast the continuous problem into a discrete one using a graph-based path planner like PRM. The PRM planner is visualized on the left in Figure 2.3. It works in two phases. The first phase is called the learning phase. During this phase, the planner tries to learn the connectivity of the free configuration space $\mathcal{Q}_F$. It does this by constructing a data structure called the *roadmap* in a probabilistic way for a given scene. The roadmap $R = (N, E)$ is an undirected graph consisting of a set of nodes and edges connecting them. The nodes in N are a set of configurations of the robot over $\mathcal{Q}_F$. The edges (q_1, q_2) in E are valid paths of movement connecting the configurations q_1 and q_2. These local paths are computed by the fast and computationally less expensive local planner. Recomputing the local paths is inexpensive, therefore local paths are not explicitly stored in the roadmap to save a considerable amount of space. The learning phase is entirely performed before any path planning query. The logic behind constructing the roadmap for a single robot is detailed in Algorithm 1.

Algorithm 1 PRM

$R = \emptyset$
while *Termination Condition* is false **do**
  $q_{\text{new}} \leftarrow$ valid *sample* from $\mathcal{Q}_F$
  Add q_{new} to nodes N of R
  $N_c(q_{\text{new}}) \leftarrow$ closest *candidate neighbors* of q_{new}
  for each $q_{\text{near}} \in N_c(q_{\text{new}})$ **do**
    $e \leftarrow$ *local path* q_{new} to q_{near}
    if $e \in \mathcal{Q}_F$ and $e \notin$ edges E of R **then**
      Add e to edges E of R
    end if
  end for
end while
return R

This general PRM implementation allows the user to make certain design choices. The termination condition is typically chosen to be a metric that represents the computational effort of the algorithm such as a maximum computation time or a maximum number of nodes. The most straightforward local planner, which is also most frequently used, checks the linear segment in the joint space connecting two configurations for collisions at points separated by a predefined local step size. These two aspects will be the same throughout this work. More interesting is how to sample nodes from $\mathcal{Q}_F$ and how to connect them to form a roadmap that estimates the connectivity of $\mathcal{Q}_F$ properly. It is these two aspects that have the most influence on the performance of PRM. Therefore these two aspects will be further analyzed in the next two sections, Section 4.1.1 and Section 4.1.2.

4.1.1 Sampling of nodes

Node sampling can be performed in various ways, ranging from random to quasi-random and deterministic methods. It becomes more advantageous to guide the sampling process when additional information is available, such as task-specific details or workspace characteristics. The goal of an effective sampler is to distribute samples evenly across the relevant subspace of the free configuration space $\mathcal{Q}_F$. This involves preventing the formation of sample clusters and avoiding large, unoccupied areas that are important for motion planning. Several sampling methods are now discussed.

Random sampling Initially, the roadmap $R = (N, E)$ is empty. To approximate the connectivity of $\mathcal{Q}_F$, the classical standard and naive way of sampling the nodes of R is to sample them randomly according to a uniform distribution over $\mathcal{Q}$. This is known as unbiased sampling. On the other hand, Biased sampling alters the probability distribution to favor sampling of interesting regions of the composite configuration space (Orthey et al., 2023). Various biases can be employed including obstacle-based and clearance-based sampling (Amato et al., 1998; Wilmarth et al., 1999). Obstacle-based sampling biases the sampling towards obstacles, improving the planning in narrow passages. Examples are the Gaussian sampling method (Boor et al., 1999) and the bridge-based sampling method (Hsu et al., 2003). Clearance-based sampling aims to increase the distance between robots and obstacles. For instance, first a feasible configuration can be sampled and next random steps can be taken to increase the distance to obstacles (Verginis et al., 2022).

Deterministic and quasi-random sampling Sampling in a deterministic way generates each time the same samples (Janson et al., 2018). The most straightforward way of sampling nodes in a deterministic way is to generate samples according to a grid or lattice. Quasi-random sampling is deterministic but makes samples appear random. Grid or lattice sampling can be transformed into quasi-random sampling by adding Gaussian noise to the samples. Samplers based on low-discrepancy sequences, like van der Corput, Halton, or Sobol sequences, generate samples in a deterministic way that are of low discrepancy and thus appear to be random for many purposes (S. M. LaValle, 2006). Another quasi-random sampling method is called Poisson-disk sampling. This is a sampling process in which samples are independently drawn from a certain random distribution while ensuring that samples are separated from each other by a minimum distance. Deterministic and quasi-random sampling prevent clusters of samples from being generated and minimize the largest unsampled region in $\mathcal{Q}$ achieving better coverage of $\mathcal{Q}$.

A visualization of uniform random sampling, grid sampling with Gaussian noise, and (2, 3)-Halton sequence sampling is given in Figure 4.1.

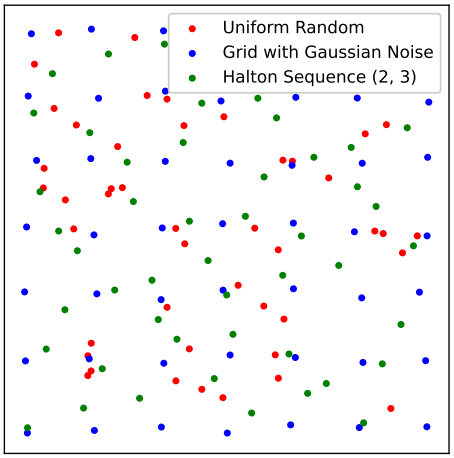


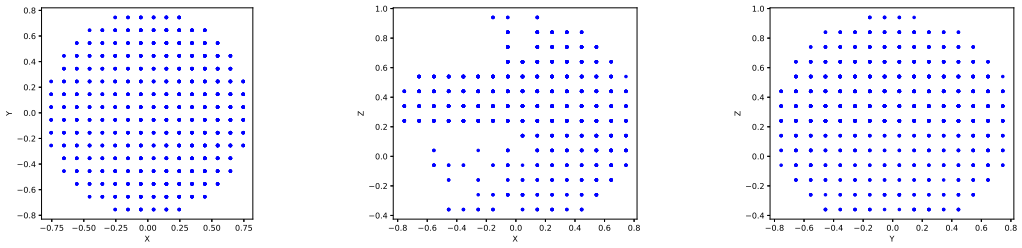
Figure 4.1: Illustration of different sampling strategies. The red samples are drawn from a uniform distribution over the black square. The blue samples are generated from a grid, with Gaussian noise added. The green samples form a (2, 3)-Halton sequence. It can be seen that the grid and Halton sequence achieve better coverage over the square, with points sufficiently distant from one another. In contrast, the random red samples can cluster together, making samples redundant for effective estimation of the connectivity of $\mathcal{Q}$.

Validation of nodes After sampling a new node, it must be checked for collisions[1]. If collision-free, it is added to the set of nodes N used to construct the roadmap R. Collision checking includes self-collisions within the manipulator and collisions with static obstacles or other manipulators. Nodes leading to self-collision are discarded. Checking for collisions with other

[1]Collision checking can be done using various techniques such as the Gilbert-Johnson-Keerthi algorithm (Coulombe & Lin, 2020) and Bounding Volume Hierarchies such as Bounding Spheres or Axis-Aligned Bounding Boxes (Klosowski et al., 1998).

manipulators is only relevant during the path search phase. Collision checking with static obstacles can be done either during node validation or deferred to the query phase. Discarding nodes that collide with static obstacles refines the roadmap for the specific workspace, reducing computational effort during path search but limiting the roadmap's reusability in different scenes. Therefore, in this work, only nodes causing self-collision are discarded.

Task space sampling Up to this point, nodes have been sampled directly in $\mathcal{Q}$, the joint or configuration space. However, sampling can also be performed in the task space of the robot. This approach is more suitable for manipulators focused on controlling the movement of the end effector to manipulate objects. In this work, Panda manipulators (see Section 5.2) perform pick-and-place tasks. Not all samples in $\mathcal{Q}$ are relevant for this purpose. Relevant samples are determined by sampling positions in the task space and using the inverse kinematics solver detailed in Section 5.2.2 to map them to $\mathcal{Q}$. The sampled positions obtained by placing a grid over the workspace of the Franka Emika Panda are shown in Figure 4.2.



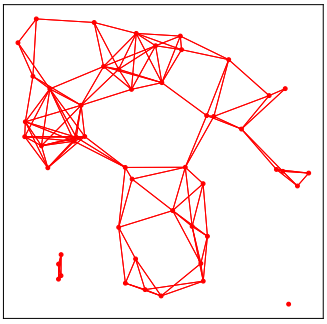
(a) Projection on XY-plane. (b) Projection on XZ-plane. (c) Projection on YZ-plane.

Figure 4.2: Task space sampling for the Franka Emika Panda manipulator. The sampled positions, visualized as projections on the (a) XY-plane, (b) XZ-plane, and (c) YZ-plane, represent possible end effector locations within the robot's workspace. These positions are mapped to the joint configuration space $\mathcal{Q}$ using the inverse kinematics solver detailed in Section 5.2.2. In this way, mainly relevant configurations for pick-and-place tasks are considered.

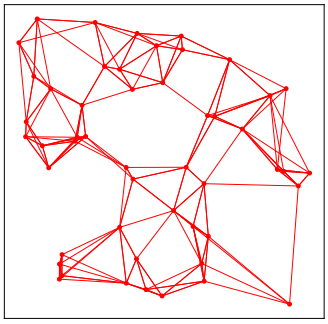
Two points need further clarification. First, for inverse kinematics, the orientation of the end effector can also be specified. The solution must achieve the desired position and orient the end effector to point downwards, fixed by the quaternion $(1, 0, 0, 0)$. Other end effector orientations with the same axis can be obtained by rotating the 7^{th} joint of the Panda manipulator (see Figure 5.1) within its joint limits. Secondly, since the manipulator is redundant there are multiple solutions to this inverse kinematics problem and it becomes an optimization problem. The inverse kinematics solver calculates the pose that realizes a certain end effector position and orientation closest to another specified pose. This pose, called the default pose, is defined as $(0, -\frac{\pi}{4}, 0, -\frac{3\pi}{4}, 0, \frac{\pi}{2}, \frac{\pi}{4})$ for the Panda manipulator.

4.1.2 Constructing the roadmap

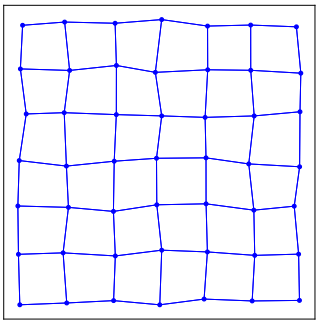
With the nodes sampled, the next step is to connect them to construct the roadmap. Nodes are connected to their nearest neighbors. These nearest neighbors are typically within a predefined distance from the node or are the k-nearest neighbors, where k is a predefined integer. Different roadmaps obtained from the samples in Figure 4.1 are shown in Figure 4.3.



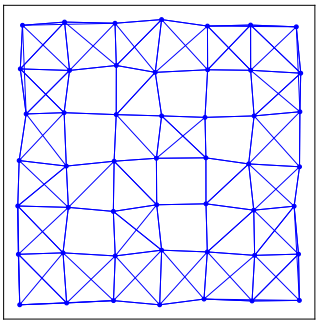
Uniform Random - Distance.



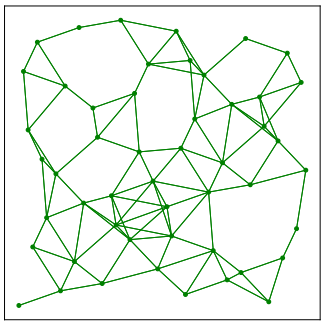
Uniform Random - kNN.



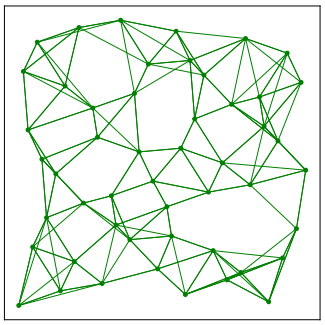
Grid with Noise - Distance.



Grid with Noise - kNN.



Halton Sequence - Distance.



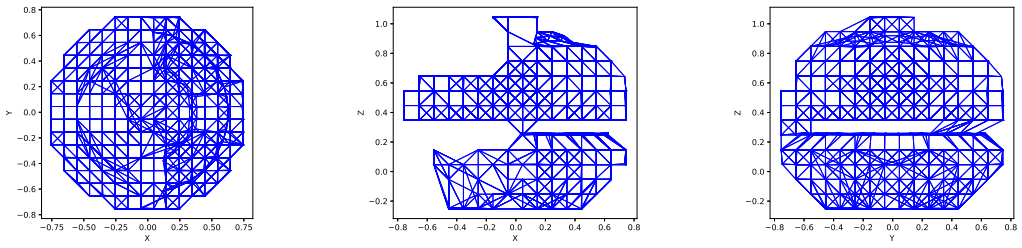
Halton Sequence - kNN.

Figure 4.3: Illustration of the different roadmaps obtained from the samples in Figure 4.1. On the left neighbors are within a predefined distance from each other and on the right each node is connected to its k-nearest neighbors. Compared to random sampling, Halton sampling achieves better spread and less clustering and the largest unsampled region is the smallest for a grid.

To connect nodes to their nearest neighbors with valid edges, two components are required: a distance metric and a local planner.

Distance metric The choice of distance metric significantly impacts the roadmap construction, as it determines which nodes are considered neighbors and subsequently connected. A naive approach would be to use the Euclidean distance in the joint space. However, this is not ideal, as illustrated with the Panda manipulator (see Figure 5.1). For example, if the 1^{th} joint rotates by $\pi/2$ or the 6^{th} joint rotates by the same angle, both new poses would be considered equally distant from the initial pose, which contradicts our intuition. Instead, a good distance metric should reflect how far two robot poses are from each other in the workspace. Therefore, the distance metric chosen for the Panda manipulator is the sum of Euclidean distances between the origins of the Denavit-Hartenberg (DH) frames defining the two poses. Since the poses are sampled from the end effector positions in task space and inverse kinematics is used to find a pose closest to the default pose, one could also simply use the Euclidean distance between the end effector positions as the distance metric.

Local planner The local planner assesses whether a direct path between two nodes is feasible and collision-free. It generates a trajectory that connects the nodes while adhering to the manipulator's kinematic and dynamic constraints. In this work, the local planner evaluates the straight-line motion between nodes at regular intervals. This can be done in either joint space or task space. When performed in task space, inverse kinematics is used to determine the manipulator's pose for collision checking. Because the local planner needs to be fast, local planning is done along the line segment connecting two nodes in joint space in this work. In Figure 4.4, the roadmap constructed for the samples in Figure 4.2 is shown.



(a) Projection on XY-plane. (b) Projection on XZ-plane. (c) Projection on YZ-plane.

Figure 4.4: Roadmap constructed from task space samples in Figure 4.2, shown as projections on the (a) XY-plane, (b) XZ-plane, and (c) YZ-plane. Samples are mapped to joint space, selecting poses closest to a default pose with the end effector oriented downward. Nodes are then connected to their 6 nearest neighbors using a distance metric based on the origins of the DH frames.

Adding start and goal to the roadmap Once the roadmap is constructed, the manipulator is ready to receive a task that specifies a start and goal pose for the manipulator. The next step in the path planning process is to incorporate the start and goal configurations into the roadmap. This involves connecting the start and goal nodes to the nearest nodes in the existing roadmap using the same distance metric and local planner employed during the roadmap construction. By doing so, the start and goal nodes become part of the network of nodes and edges, enabling the planner to find a feasible path from the start to the goal. Once the roadmap is constructed,

we go on to the second phase which is called the query phase. In the query phase, the roadmap is searched to solve path planning problem. In this phase, an optimal search algorithm like A* is typically used to find the path. However, in this case for multiple manipulators, CBS will be used instead. How this is done, is covered in the next section (see Section 4.2).

4.2 Conflict-Based Search to query roadmaps

For using CBS for the continuous motion planning problem for robots in a shared environment instead of points on a shared graph, there are some essential differences to take into account.

Individual graphs Unlike the CBS framework where agents are treated as points moving from node to node on a shared graph, the motion planning challenges addressed in this thesis involve manipulators navigating within a shared workspace $\mathcal{W}$. In this workspace, valid continuous motions have to be found for each agent a_i in a set of n agents $A = \{a_1, ..., a_n\}$ from an initial configuration $q_{\text{init,i}}$ to a goal configuration $q_{\text{goal,i}}$. With $\mathcal{Q}_{F,i}$ the free configuration space of agent a_i, we have that $q_{\text{init,i}} \in \mathcal{Q}_{F,i}$ and $q_{\text{goal,i}} \in \mathcal{Q}_{\text{goal,i}} \subset \mathcal{Q}_{F,i}$ with $\mathcal{Q}_{\text{goal,i}}$ a goal set of configurations satisfying a goal condition. Each agent thus moves in its own distinct continuous configuration space. Therefore, individual graphs or roadmaps R_i for each robot $a_i \in A$ have to be generated using PRM as discussed in Section 4.1.

Conflicts On the shared roadmap in the Multi-Agent Graph Search (MAGS) problem, agents move from node to node in the same roadmap and conflicts arise from simultaneous node occupancy or when swapping node at a given time step. However, in a continuous setting, conflicts emerge primarily as agents traverse edges between nodes on their individual roadmaps. To identify these, paths are first discretized with a defined time resolution and at each time step collision checking is performed for each of the robots. To achieve this, the velocity of each of the robots when traversing the path has to be known a priori. Therefore it is assumed that each robot moves at the same predetermined velocity that remains constant over the whole trajectory until the goal configuration is reached. In the individual roadmaps, the edges are of different lengths and thus each edge takes a different duration to traverse. A conflict between robots moving along their respective paths is defined by specifying the time step t at which the collision occurs and the pair of configurations q_i, q_j for which the robots a_i, a_j are in collision at time t. The conflict is thus denoted as $c = (t, a_i, a_j, q_i, q_j)$.

Constraints Once the earliest conflict has been detected, the next step involves deriving constraints that specifically prevent the recurrence of this collision. In scenarios involving point representations on a shared roadmap, two agents are in a node conflict at time t with each other if they simultaneously occupy the same node in the shared roadmap. To resolve this conflict, constraining one of the agents to occupy a different node at time t can effectively ensure that no conflict will occur between these agents at that time during subsequent queries of the low-level search algorithm. For robots navigating on individual roadmaps and traversing its edges, conflicts are more likely to occur while traversing these edges. Initially, one might consider imposing a constraint for one of the conflicting agents to prevent the edge on which the conflict occurs is traversed at the time of conflict t. However, the configuration q_i for agent a_i at time t now impacts a *set of edges* in the roadmap R_j of robot a_j. If agent a_j traverses an edge in this set when agent a_i is at configuration q_i, a collision will occur. If the distance function for constructing the roadmap is chosen well - so as to accurately reflect the physical separation between configurations in the workspace - this set of edges will form a concentrated *conflict zone* in the configuration

space. Simply assigning a constraint to a single edge would likely trigger a cascade in the CT of CBS of conflicts within the same zone, as attempts to resolve one conflict might lead to others in closely situated edges. To address this challenge more effectively, it is advantageous to design the constraint to encompass all potential conflicts within the same conflict zone simultaneously. This can speed up the search process within the CT of CBS. The constraint for agent a_j is thus formulated as $c = (t, a_j, a_i, q_i)$, which restricts agent a_j from traversing any edge during the time t that would result in a collision with agent a_i being at configuration q_i. This constraint not only resolves the initial conflict but any conflict between the two agents at time t. Similar to the approach used in CBS with a shared roadmap, the resolution involves incorporating the two constraints, $c_1 = (t, a_j, a_i, q_i)$ and $c_2 = (t, a_i, a_j, q_j)$, into separate child nodes of a parent node within the constraint tree. Pseudocode illustrating the operation of the algorithm is shown in Algorithm 2.

Algorithm 2 CBS-PRM

Initialize roadmaps R_i for each agent a_i in the set of agents A using PRM
for each agent $a_i \in A$ **do**
    Add start node $q_{\text{init,i}}$ and goal node $q_{\text{goal,i}}$ to roadmap R_i
end for
Initialize CT with a root node *start* having no constraints
start.solution $\leftarrow$ initial paths for each agent computed without constraints
if not *start.solution* **then**
    return {} ▷ Exit if no initial solution
end if
start.discretized_solution $\leftarrow$ discretized initial paths using predetermined time step
start.cost $\leftarrow$ sum of costs of the initial paths
Initialize *open_set* $\leftarrow$ {*start*} and *closed_set* $\leftarrow$ {}
while *open_set* is not empty **do**
    $P \leftarrow$ CT node in *open_set* with minimum cost
    Remove P from *open_set* and add to *closed_set*
    $C \leftarrow$ earliest conflict detected in *P.discretized_solution*
    if C is None **then**
        return *P.discretized_solution* ▷ Solution found
    end if
    Generate constraints c_i, c_j for agents involved in C
    for $k \in \{i, j\}$ **do**
        $P_{\text{new}} \leftarrow$ inherit constraints from P and add c_k for a_k
        P_{new}*.solution* $\leftarrow$ paths for each agent computed with new constraints
        if not P_{new}*.solution* **then**
            Continue to the next iteration
        end if
        P_{new}*.discretized_solution* $\leftarrow$ discretized paths using predetermined time step
        P_{new}*.cost* $\leftarrow$ sum of costs of the paths
        if P_{new} not in *closed_set* **then**
            open_set.remove(P_{new})
        end if
    end for
end while
return {} ▷ No solution found

4.3 Trajectory generation

Once the path has been planned, the next step is to generate a trajectory that specifies the position, velocity, and acceleration profiles for the robot's joints or end effector over time. This ensures smooth movement from the start to the goal position. This process involves time parameterization of the path provided by the path planner. There are various methods for generating trajectories, both in task space and joint space. Since the path planner assumes linear interpolation between poses in joint space for collision-avoidance calculations, it is crucial that the generated trajectory closely follows this path. However, the trajectory should be smoother and must adhere to the kinematic and dynamic constraints of the manipulators. The path planner does not impose a specific velocity for each robot, it only dictates the relative motion between manipulators. This allows for velocity tuning during the trajectory generation process. An example of the path generated by the path planner is shown in Figure 4.5.

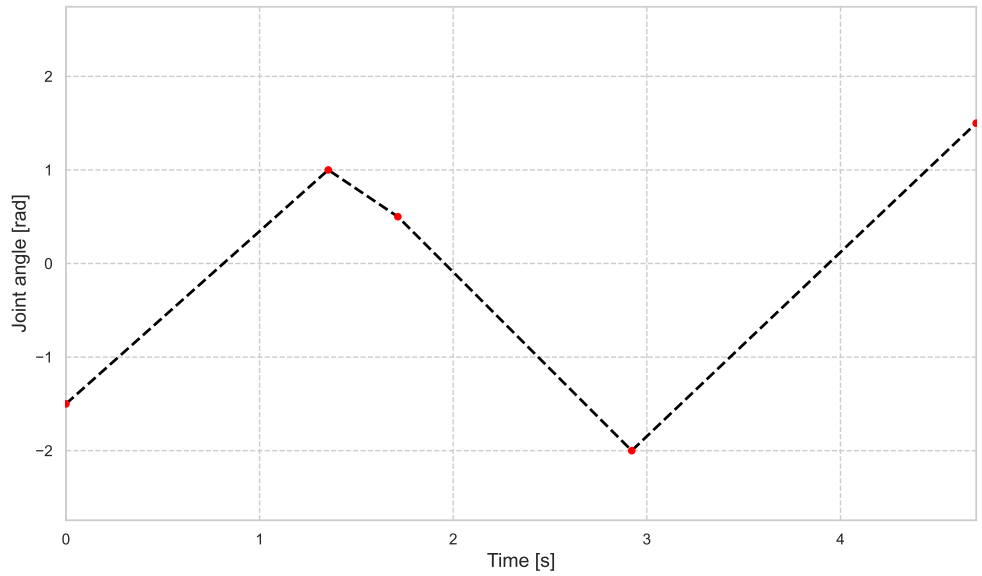


Figure 4.5: Example of the assumed linear interpolated path between waypoints in the joint space. In this figure, the waypoints for the first joint angle of the Panda manipulator are shown in red. The joint angles for this joint are equal to -2.7437 rad and 2.7437 rad. This path cannot be given directly to the controller because of jerky movements and too abrupt changes in velocity and acceleration.

The purpose of the trajectory generator is to convert the planned path into a trajectory that ensures collision-free motion while adhering to kinematic and dynamic constraints, making it suitable for direct input into the manipulator's controller. Various methods exist for trajectory generation, such as defining polynomials through waypoints or using splines, which are piecewise combinations of lower-order polynomials. One of the most widely used techniques in the industry is the Linear Segment with Parabolic Blends (LSPB) trajectory, which is also discussed here to generate a smooth trajectory. This method features phases of constant acceleration, zero acceleration, and constant deceleration, resulting in a trapezoidal velocity profile. The position profile consists of linear segments with parabolic blends, resembling an S-curve. To calculate the sequence of LSPBs, the first step is to allocate time to the path. This is done by arbitrarily

choosing an initial value for the time interval within which the movement must be executed. From this time allocation, the average speeds for each linearly interpolated segment are calculated and compared to the velocity limits of the manipulator. The velocity limits for the Franka Panda are given by:

$$\text{Velocity limits} = \begin{bmatrix} 2.62 & 2.62 & 2.62 & 2.62 & 5.26 & 4.18 & 5.26 \end{bmatrix} \frac{\text{rad}}{\text{s}} \tag{4.1}$$

Next, the time allocation is scaled so that all the velocities fall within a certain margin of the velocity limits. For each joint, the LSPB trajectory is computed iteratively over the waypoints. Given two waypoints p_1 and p_2 with average velocity v_{12}, the initial velocity v_1 and final velocity v_2 for each segment are determined as follows:

$$v_1 = 0 \quad \text{(initial velocity)} \quad \text{or} \quad v_1 = \frac{v_{01} + v_{12}}{2} \quad \text{(average velocity)} \tag{4.2}$$

$$v_2 = 0 \quad \text{(final velocity)} \quad \text{or} \quad v_2 = \frac{v_{12} + v_{23}}{2} \quad \text{(average velocity)} \tag{4.3}$$

where v_{01} and v_{23} are the average velocities of the previous and next segments respectively. The accelerations for the parabolic blends are:

$$a_1 = \begin{cases} a & \text{if } v_{12} > v_1 \\ -a & \text{if } v_{12} < v_1 \\ 0 & \text{if } v_{12} = v_1 \end{cases} \qquad \text{and} \qquad a_2 = \begin{cases} -a & \text{if } v_{12} > v_2 \\ a & \text{if } v_{12} < v_2 \\ 0 & \text{if } v_{12} = v_2 \end{cases} \tag{4.4}$$

The constant a is selected to ensure it remains within a specified margin of each acceleration limit. The acceleration limits for the Franka Panda are given by:

$$\text{Acceleration limits} = \begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{bmatrix} \frac{\text{rad}}{\text{s}^2} \tag{4.5}$$

The time t_1 for the start of each segment is either $t_1 = 0$ for the first segment or the end time of the previous segment. Given t_1, p_1, v_1, and a_1, it is straightforward to calculate the time t_1' at which the first parabolic blend ends and the linear segment begins:

$$t_1' = \frac{v_{12} - v_1}{a_1} + t_1 \tag{4.6}$$

Given $a_1 = a$, $b_1 = v_1$, and $c_1 = p_1$, we can determine p_1' (the position at t_1') as follows:

$$p_1' = \frac{a_1}{2}(t_1' - t_1)^2 + b_1(t_1' - t_1) + c_1 \tag{4.7}$$

For the second parabolic blend, we can find:

$$\Delta t_2 = t_2 - t_2' = \frac{v_{12} - v_2}{a_2} \tag{4.8}$$

where t_2' is the time at which the linear segment ends and the second parabolic blend begins. With $a_2 = a$, $b_2 = v_{12} = v_2 - a_2\Delta t_2$, and $c_2 = p_2' = v_{12}(t_2' - t_1') + p_1'$, the equation for the second parabolic blend yields:

$$t_2' = \frac{p_2 - p_1' - \frac{a_2}{2}\Delta t_2^2 - v_{12}\Delta t_2 + v_{12}t_1'}{v_{12}} \tag{4.9}$$

The position p_2' can now be calculated using the equation $p_2' = v_{12}(t_2' - t_1') + p_1'$. With all the times, positions, velocities, and accelerations known, the LSPB trajectory is uniquely determined. In particular, t_2 is now known and the next LSPB can be calculated. This process is repeated for each joint and segment to form the entire LSPB trajectory for each joint. An example of the path and LSPB trajectory for the first joint of the Franka Panda is shown in Figure 4.6.

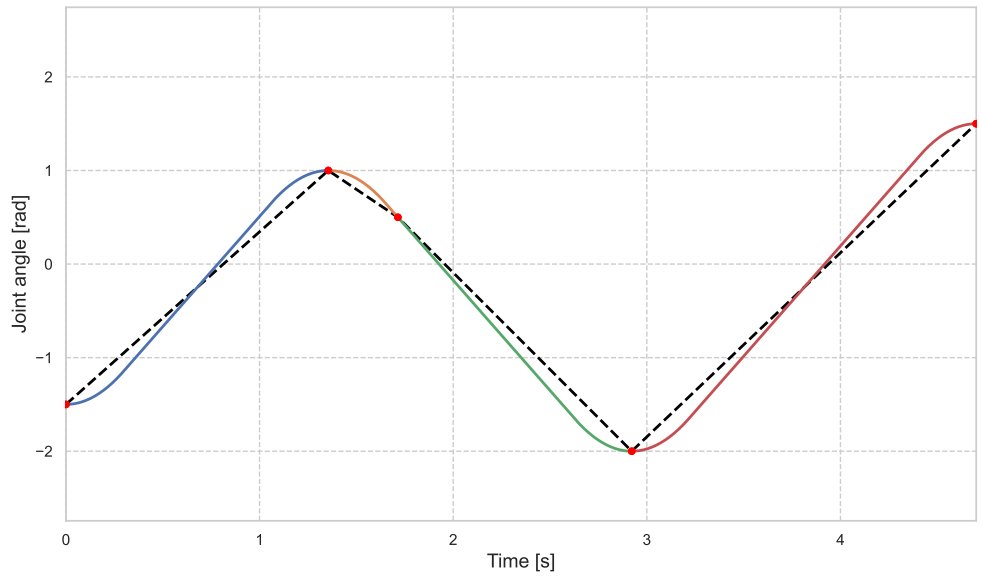


Figure 4.6: An example of the path and corresponding LSPB trajectory for the first joint of the Franka Panda. The waypoints are depicted as red dots and the linear interpolated motion as black dotted lines. The trajectory is a sequence of blue, orange, green, and red LSPB trajectories, ensuring a smooth motion that satisfies both kinematic and dynamic constraints while closely following the linear interpolated motion in joint space.

4.4 Conclusion

In conclusion, this chapter presented a novel approach for collision-free motion planning of multiple manipulators with overlapping workspaces. The proposed motion planning framework integrates the Conflict-Based Search (CBS) search algorithm with the Probabilistic Roadmaps (PRM) path planner, followed by a trajectory generation phase. The formation of a roadmap that estimates the joint space well is essential for finding satisfactory solutions, with both node sampling and roadmap construction significantly influencing its quality. By leveraging individual roadmaps for each manipulator, CBS effectively finds collision-free paths, with necessary adjustments in conflict detection and constraint assignment as detailed in Section 4.2. Finally, the calculated paths are transformed into smoother trajectories that adhere to kinematic and dynamic constraints using Linear Segment with Parabolic Blends (LSPB), as discussed in Section 4.3.

Chapter 5

Experimental setup

This chapter presents the methods, frameworks, and tools used to evaluate the performance of the motion planning algorithms, with a particular emphasis on the approach described in Chapter 4. Section 5.1 outlines the tested algorithms and the software and hardware utilized in this research. Section 5.2 provides an overview of the Franka Emika Panda manipulator, focusing on forward kinematics and numerical inverse kinematics solvers. The independent pick-and-place tasks performed by these manipulators are detailed in Section 5.3. Finally, Section 5.4 discusses the metrics chosen for evaluating the algorithms.

5.1 Implementation details

5.1.1 Algorithms

This study primarily examines the performance of a novel hybrid motion planning approach, which merges Conflict-Based Search (CBS) with Probabilistic Roadmaps (PRM). To establish benchmarks, this hybrid method is compared against traditional PRM in a coupled approach and prioritized PRM as a decoupled approach. Next to PRM also Rapidly-exploring Random Tree (RRT) variants were implemented and tested. Specifically, as for PRM, also the traditional coupled approach and decoupled prioritized versions of RRT were implemented. The implementation of the algorithms is available on GitHub.

5.1.2 Software

The algorithms were developed in Python, chosen for its robust support for rapid prototyping and integration with machine learning libraries, which may be explored in future research extensions. While the Open Motion Planning Library (OMPL), primarily based in C++, was considered for its extensive range of planning algorithms, Python was retained for its flexibility and ease of use in initial testing phases. For visualization and simulation, the matplotlib library was utilized for 2D graphical outputs and Pybullet was employed for physics-based simulation in 3D environments.

5.1.3 Hardware

The simulations were conducted on a machine equipped with Windows 11 Home version 22H2, powered by a 1.8 GHz Intel Core i7-8565U processor and 16 GB of RAM.

5.2 Franka Emika Panda manipulators

Simulations were performed in 3-dimensional space using Pybullet. The simulations involve Franka Emika Panda manipulators with 7 DOFs in multi-manipulator setups performing individual pick-and-place tasks. Therefore, in this section, the Franka Emika Panda manipulator is detailed. In Section 5.2.1, the forward kinematics is discussed, and in Section 5.2.2, the inverse kinematics.

5.2.1 Forward kinematics

The forward kinematics of the Franka Emika Panda involves calculating the position and orientation of the end effector based on the given joint angles. The joint limits of the Franka Emika Panda are given by:

$$q_{\max} = \begin{bmatrix} 2.7437 & 1.7837 & 2.9007 & -0.1518 & 2.8065 & 4.5169 & 3.0159 \end{bmatrix} \text{ rad} \tag{5.1}$$

$$q_{\min} = \begin{bmatrix} -2.7437 & -1.7837 & -2.9007 & -3.0421 & -2.8065 & 0.5445 & -3.0159 \end{bmatrix} \text{ rad} \tag{5.2}$$

Determining the forward kinematics of the Panda manipulator is done using the modified DH convention. Following this convention, reference frames are assigned to each of the links of the robot. The reference frames of the Panda manipulator are defined as shown in Figure 5.1.

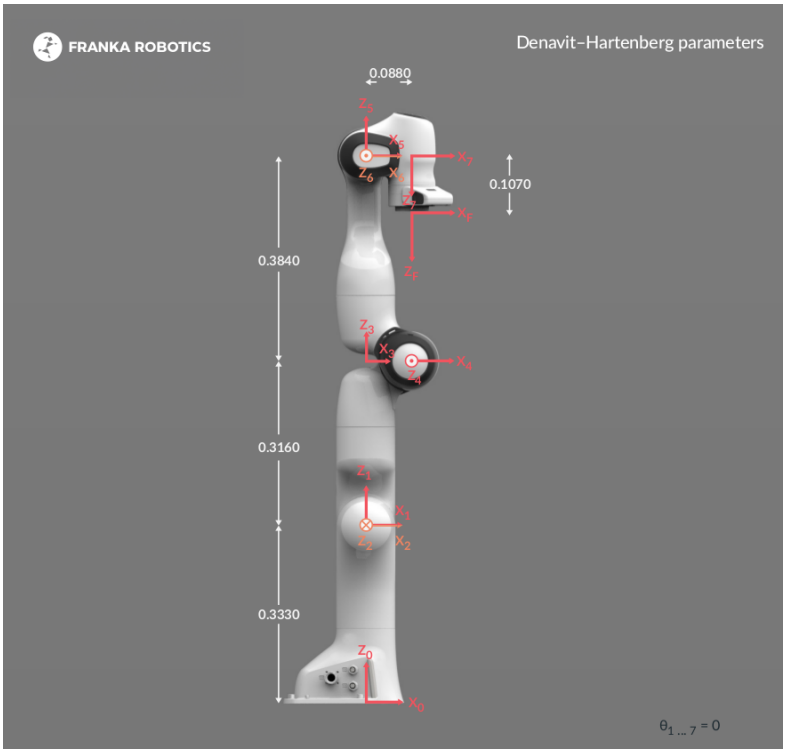


Figure 5.1: Franka Emika Panda with Denavit-Hartenberg (DH) reference frames, adapted from Franka Emika GmbH (2024).

Based on these frames, the DH parameters α_i, a_i, d_i and θ_i are determined. α_i is the twist angle between z_{i-1} and z_i around x_i. a_i is the distance along the common normal between the joint axes i and i+1. d_i is the distance, along the axis of joint i, between the intersection of joint axis i with the common normal coming from joint axes i+1 and i-1. θ_i is the angle between x_{i-1} and x_i around z_{i-1}. The DH parameters corresponding to Figure 5.1 are given in Table 5.1.

Table 5.1: Denavit-Hartenberg (DH) parameters for the Franka Emika Panda manipulator.

Joint	**a (m)**	**d (m)**	**α (rad)**	**θ (rad)**
1	0	0.333	0	θ_1
2	0	0	$-\pi/2$	θ_2
3	0	0.316	$\pi/2$	θ_3
4	0.0825	0	$\pi/2$	θ_4
5	-0.0825	0.384	$-\pi/2$	θ_5
6	0	0	$\pi/2$	θ_6
7	0.088	0	$\pi/2$	θ_7
Flange	0	0.107	0	0

The modified homogeneous transformation matrix linking frame i-1 and frame i is given by:

$$
{}^{i-1}T_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_i \\ \sin(\theta_i)\cos(\alpha_i) & \cos(\theta_i)\cos(\alpha_i) & -\sin(\alpha_i) & -d_i\sin(\alpha_i) \\ \sin(\theta_i)\sin(\alpha_i) & \cos(\theta_i)\sin(\alpha_i) & \cos(\alpha_i) & d_i\cos(\alpha_i) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}
$$

This allows us to calculate the end effector's pose with respect to the base frame as follows:

$$
\begin{aligned}
{}^0T_F(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6,\theta_7) &= {}^0T_1(\theta_1){}^1T_2(\theta_2){}^2T_3(\theta_3){}^3T_4(\theta_4){}^4T_5(\theta_5){}^5T_6(\theta_6){}^6T_7(\theta_7){}^7T_F(\theta_F) \\
&= \begin{bmatrix} {}^0R_F & {}^0\vec{p}_F \\ 0 & 1 \end{bmatrix}
\end{aligned} \tag{5.4}
$$

5.2.2 Inverse kinematics

The inverse kinematics (IK) problem for the Franka Emika Panda involves finding the joint angles that achieve a desired end effector pose. The Franka Emika Panda is a manipulator with 7 DOFs, making it a redundant robot. This redundancy implies that the IK problem can have multiple solutions, providing flexibility in achieving the desired end effector pose. By fixing one of the 7 joints of the Panda manipulator, it is possible to find solutions to the IK problem analytically. In the work of He and Liu (2021), an analytical approach to solving the inverse kinematics is discussed, in which the 7^{th} joint is fixed. A classical numerical approach that is relatively easy to implement is the Jacobian pseudoinverse method (Buss, 2004). PyBullet also has a built-in numerical IK solver based on the damped least squares method discussed by Buss (2004). Both numerical inverse kinematics solvers are now discussed.

The Jacobian pseudoinverse method

Consider a manipulator with n joint angles which govern the position function computed with forward kinematics. Since the manipulator operates in a 3-dimensional space, the position function is a mapping $p(q) : \mathbb{R}^n \to \mathbb{R}^3$. With $p_0 = p(q_0)$ the initial position of the end effector and $p_1 = p(q_0 + \Delta q)$ the goal position of the end effector, the method iteratively computes an estimate of Δq that minimizes the error given by:

$$\|p(q_0 + \Delta q_{\text{estimate}}) - p_1\| \tag{5.5}$$

For small Δq vectors, the series expansion of the position function can be approximated by:

$$p(q_1) \approx p(q_0) + J_p(q_0)\Delta q \tag{5.6}$$

where $J_p(q_0)$ is the $(3 \times n)$ Jacobian matrix of the position function at q_0. The (i, j)-th entry of the Jacobian matrix can be approximated numerically as:

$$\frac{\partial p_i}{\partial q_j} \approx \frac{p_i(q_{0,j} + h) - p_i(q_0)}{h} \tag{5.7}$$

where $p_i(q)$ is the i-th component of the position function, $q_{0,\mathrm{j}} + h$ is q_0 with a small delta added to its j-th component, and h is a reasonably small positive value.

Taking the Moore–Penrose pseudoinverse of the Jacobian and rearranging Equation (5.6) results in:

$$\Delta q \approx J_p^\dagger(q_0)\Delta p \tag{5.8}$$

where $\Delta p = p(q_0 + \Delta q) - p(q_0)$.

Applying the method once will yield a rough estimate of Δq. The estimate for Δq can be improved using the Newton–Raphson method:

$$\Delta q_{k+1} = J_p^\dagger(q_k)\Delta p_k \tag{5.9}$$

Once a Δq is computed for which the error (see Equation (5.5)) is close to zero, the algorithm terminates. The pseudoinverse has the further property that the matrix $(I - J_p^\dagger J_p)$ performs a projection onto the null space of J_p. Therefore, for all vectors φ, we have:

$$J_p(I - J_p^\dagger J_p)\varphi = 0. \tag{5.10}$$

This means that we can set Δq equal to:

$$\Delta q = J_p^\dagger \Delta p + (I - J_p^\dagger J_p)\varphi \tag{5.11}$$

for any vector φ and still obtain a value for Δq which minimizes the value $J_p\Delta q - \Delta p$. By suitably choosing φ, one can try to achieve secondary goals in addition to having the end effector track the target positions.

The damped least squares method

The damped least squares method, also called the Levenberg-Marquardt method, avoids the pseudoinverse method's problems with singularities and is a more numerically stable method for determining Δq. Instead of searching the Δq that minimizes Equation (5.5), Δq is optimized to minimize the quantity:

$$\|J_p \Delta q - \Delta p\|^2 + \lambda^2 \|\Delta q\|^2 \tag{5.12}$$

where $\lambda \in \mathbb{R}$ is a non-zero damping constant. This is equivalent to minimizing the quantity:

$$\left\| \begin{pmatrix} J_p \\ \lambda I \end{pmatrix} \Delta q - \begin{pmatrix} \Delta p \\ \vec{0} \end{pmatrix} \right\| \tag{5.13}$$

The corresponding normal equation is:

$$\left(\begin{pmatrix} J_p \\ \lambda I \end{pmatrix}^T \begin{pmatrix} J_p \\ \lambda I \end{pmatrix} \right) \Delta q = \begin{pmatrix} J_p \\ \lambda I \end{pmatrix}^T \begin{pmatrix} \Delta p \\ \vec{0} \end{pmatrix} \tag{5.14}$$

This can be rewritten as:

$$(J_p^T J_p + \lambda^2 I) \Delta q = J_p^T \Delta p \tag{5.15}$$

It can be shown that $J_p^T J_p + \lambda^2 I$ is non-singular. Thus, the damped least squares solution is equal to:

$$\Delta q = (J_p^T J_p + \lambda^2 I)^{-1} J_p^T \Delta p \tag{5.16}$$

Now $J_p^T J_p$ is an $n \times n$ matrix, where n is the number of DOFs. It is possible to show that $(J_p^T J_p + \lambda^2 I)^{-1} J_p^T = J_p^T (J_p J_p^T + \lambda^2 I)^{-1}$. Substitution in Equation (5.16) yields:

$$\Delta q = J_p^T (J_p J_p^T + \lambda^2 I)^{-1} \Delta p \tag{5.17}$$

The advantage of Equation (5.17) over Equation (5.16) is that the matrix being inverted is only $m \times m$ where m is the dimension of the space of target positions, and m is often much less than n. Additionally, Equation (5.17) can be computed without matrix inversion. Instead, row operations can find $\vec{f}$ so that $(J_p J_p^T + \lambda^2 I)\vec{f} = \Delta p$, and the solution can then be found as $\Delta q = J_p^T \vec{f}$. The damping constant must be chosen carefully to ensure numerical stability in Equation (5.17). It should be large enough to keep the solutions for Δq stable near singularities. However, if the damping constant is too large, the convergence rate may become excessively slow. It is this numerical inverse kinematics solver that is implemented in Pybullet and used throughout this work.

5.3 Pick-and-place tasks

In this work, we focus on pick-and-place tasks where manipulators pick up objects from above and place them in a similar manner. Each manipulator starts from a default or home pose. The sequence of actions for a Franka Emika Panda manipulator to perform a pick-and-place task starts with moving the manipulator to a position directly above the object to be picked up. Next, it has

to move downward until the object is positioned between the jaw grippers. Then the grippers close to securely grasp the object after which the object is lifted up. The manipulator then moves the object to the designated place position and lowers it to the ground. Finally, the grippers open to release the object and the manipulator moves back up and returns to the home pose. Multiple manipulators with overlapping workspaces perform this task simultaneously. To ensure proper execution of the pick-and-place tasks, the motion planner guarantees that the movements of all manipulators are collision-free with respect to each other. Therefore, the performance evaluation experiments will involve multiple manipulators simultaneously moving from start to goal pose instead of performing pick-and-place tasks.

5.4 Efficiency evaluation

To assess the efficiency and performance of the motion planning algorithms presented, performance metrics are recorded during experiments. The metrics used in the evaluation are the following:

Computation time The duration required by an algorithm to determine a viable path from the start to the goal configuration. Lower computational times indicate higher efficiency. Computation time includes both the learning time T_L and the query time T_Q.

Memory Usage The computational memory consumed during motion planning algorithms. Efficient memory use enhances suitability for systems with limited capacities. It includes the number of nodes, edges, and for CBS-PRM, CT nodes.

Success rate The proportion of trials where the algorithm finds a collision-free path. A high success rate indicates robustness and reliability in diverse scenarios.

Path length The total travel distance or cost of the paths found by the algorithm. Shorter paths reduce energy consumption and task execution time, enhancing operational efficiency.

Scalability The algorithm's performance as the problem scale increases, especially with the number of DOFs of the MRS. A scalable algorithm effectively handles increased complexity without significant degradation in other metrics.

5.5 Conclusion

The implementation of the algorithms, available on GitHub, was developed in Python for its flexibility and ease of use, with visualization and simulation facilitated by Matplotlib and Pybullet. The Franka Emika Panda manipulators were utilized as the primary models for testing the algorithms' performance in multi-manipulator setups performing pick-and-place tasks. The sequence of actions for these tasks was outlined, leading to the conclusion that if the motions between different poses are collision-free, the pick-and-place task is successful. Consequently, the experiments will involve manipulators moving from start pose to goal pose. Finally, several performance metrics were identified: computation time, memory usage, success rate, path length, and scalability. These metrics will be used to assess the efficiency and performance of the motion planning algorithms in the next chapter.

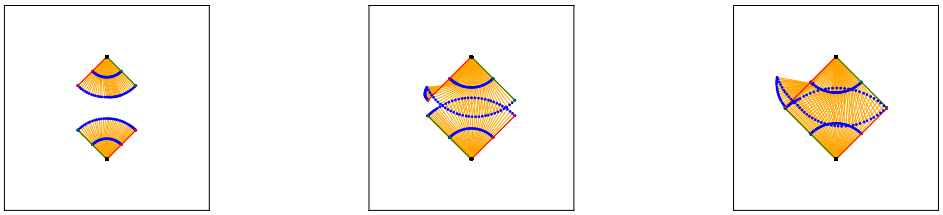
Chapter 6

Results and discussion

This chapter presents and discusses the results of experiments conducted to analyze the relative performance of the CBS-PRM algorithm compared to other algorithms. The experiments are designed to focus on scenarios where collisions between robots are likely to occur. In Section 6.1 and Section 6.2, the performance of the algorithms is investigated using planar manipulator setups. In Section 6.3 and Section 6.4, the scalability performance of the algorithms is tested and compared in different ways using setups involving multiple manipulators in Pybullet.

6.1 Proximity testing

In this experiment, two RR planar robots are placed at $(0, 5)$ and $(0, -5)$ within a 20x20 unit square. The robot lengths increase from 4 units in Figure 6.1a to 6 units in Figure 6.1b and 7 units in Figure 6.1c. Start configurations are red and goal configurations are green. As the robots progress, they create a sweep-like pattern. The robots' joints are blue dots and their links are illustrated with orange lines.



(a) Total robot's length: 4. (b) Total robot's length: 6. (c) Total robot's length: 7.

Figure 6.1: The proximity of RR articulated planar robots is changed by increasing the total length of the robots. The base positions of the robots are $(0, 5)$ and $(0, -5)$.

For each scene, all three algorithms, distance-PRM, prioritized-PRM, and CBS-PRM, are run 10 times. After each run, the performance metrics (learning time, query time, number of nodes, number of edges, degree, *maxdist* parameter, success, and sum of costs) are noted. For CBS-PRM this also includes the number of CT nodes. In order to maintain control over the graph construction during the learning phase, and thereby manage the learning time, the number of

nodes is set to a constant value of 400. The *maxdist* parameter is carefully selected to ensure a more or less consistent degree across all runs for each algorithm. The averages of the performance metrics across the 10 runs for each algorithm are listed in Table 6.1, Table 6.2, and Table 6.3.

Table 6.1: Distance-PRM performance metrics for robots of different lengths.

L	T_L	T_Q	**Nodes**	**Edges**	**Degree**	**Maxdist**	**Success**	**SOC**
4	12.67	0.22	400	915	4.57	1.5	1.0	6.72
6	14.42	0.25	400	909	4.55	1.5	1.0	9.91
7	12.23	0.25	400	933	4.66	1.5	1.0	9.75

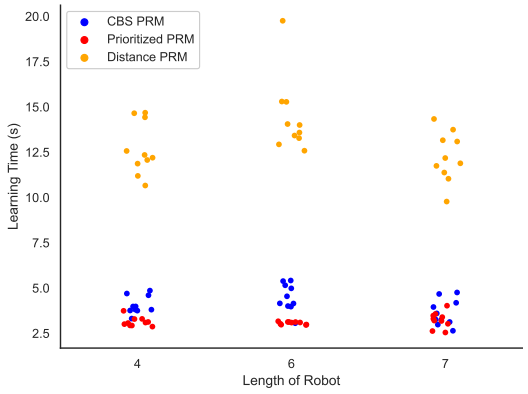
Table 6.2: Prioritized-PRM performance metrics for robots of different lengths.

L	T_L	T_Q	**Nodes**	**Edges**	**Degree**	**Maxdist**	**Success**	**SOC**
4	3.14	0.46	400	1412	7.06	0.7	1.0	5.54
6	3.07	1.30	400	1440	7.20	0.7	0.9	6.67
7	3.24	1.16	400	1487	7.44	0.7	0.9	6.85

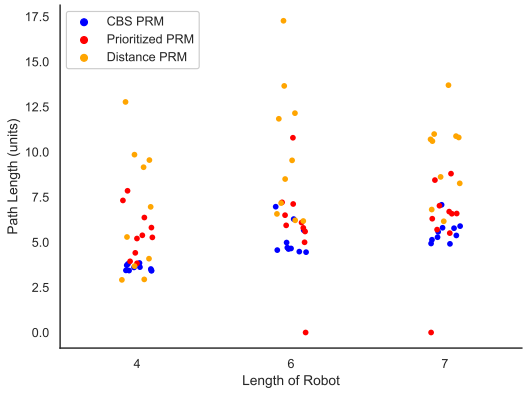
Table 6.3: CBS-PRM performance metrics for robots of different lengths.

L	T_L	T_Q	**Nodes**	**Edges**	**Degree**	**Maxdist**	**Success**	**SOC**	**CT Nodes**
4	4.84	0.20	400	1444	7.15	0.7	1.0	3.97	1
6	4.88	14.05	400	1467	7.26	0.7	1.0	5.56	41
7	4.20	14.74	400	1522	7.53	0.7	1.0	5.64	44

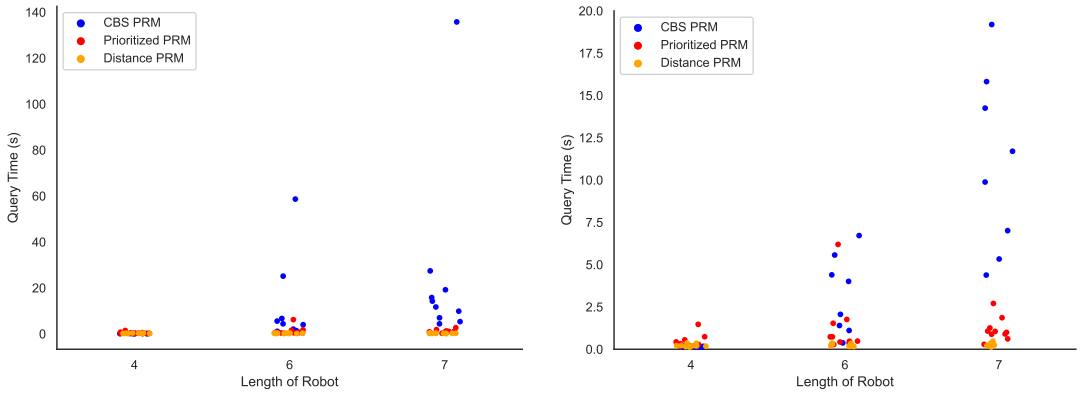
Reviewing the complete data set for performance metrics reveals their spread. Therefore Figure 6.2a, Figure 6.2b, Figure 6.2c, and Figure 6.2d show the datasets for respectively learning time, path length, and query time as a function of the total length of the manipulator.



(a)



(b)



(c) (d)

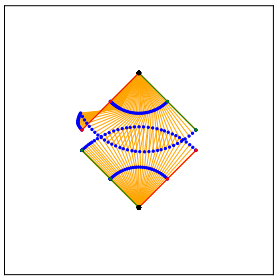
Figure 6.2: Data sets for learning time (a), sum of costs (b), and query time (c and d) for the first experiment (see Figure 6.1) in which the lengths of the manipulators is varied.

Distance-PRM builds the graph within the composite configuration space X which inherently limits scalability. The sampling density ρ in the d-dimensional space X scales proportionally with $N^{\frac{1}{d}}$, where d is the dimension of X and N represents the sample size. To maintain a constant sampling density as d increases, the number of nodes must increase exponentially. If *maxdist* remains constant, the branching factor or degree of the graph also remains constant. Consequently, the number of edges increases linearly with the number of nodes. The space needed for storing the graph thus increases exponentially with d. Additionally, the time complexity of A*, the low-level search algorithm that is used, is $\mathcal{O}(E \log N)$. This too scales exponentially with d for a constant sampling density and degree. These considerations show that the coupled approach, distance-PRM, is inherently non-scalable. Therefore, it will not be used in the following experiments regarding scalability performance.

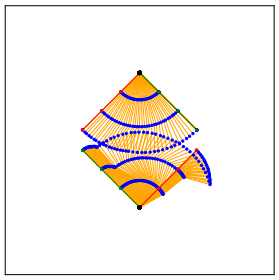
In this first experiment, 200 nodes per robot were sampled for both prioritized-PRM and CBS-PRM. To match this sampling density, distance-PRM would need 40,000 nodes. Instead, the same number of nodes was sampled across all algorithms to compare performance. The sampling density for distance-PRM is $200^{\frac{1}{2}}/400^{\frac{1}{4}} \approx 3.16$ times lower in the composite space compared to the separate $\mathcal{Q}$-spaces for prioritized-PRM and CBS-PRM, negatively impacting its performance. To achieve a similar degree and number of edges, the *maxdist* parameter must be increased, resulting in longer edges and higher learning times due to more collision checks per edge. Lower query times are obtained with fewer edges, but paths found in the query phase are generally longer for sparser roadmaps. Query times are mainly influenced by the number of edges, as reflected by the time complexity of A* ($\mathcal{O}(E \log N)$). Pathfinding failures can occur due to insufficient roadmaps or higher-priority robots blocking paths for lower-priority robots. Table 6.2 and Table 6.3 show that CBS-PRM consistently has a success rate of 100%, while prioritized-PRM sometimes fails to find paths. Additionally, CBS-PRM computes shorter paths but requires more time to do so, as seen in Figure 6.2c and Figure 6.2d. In terms of space requirements, CBS-PRM needs more memory than prioritized-PRM because it also stores nodes in the conflict tree, in addition to the roadmap.

6.2 Increasing the number of links

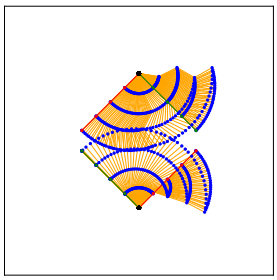
In this experiment, two planar articulated robots are once again positioned at $(0, 5)$ and $(0, -5)$ within a 20x20 square. Each robot has a length of 6, making Figure 6.3a identical to Figure 6.1b. The number of links varies from 2 to 5. The objective is to study the scalability of the algorithms with respect to the number of DOFs per robot.



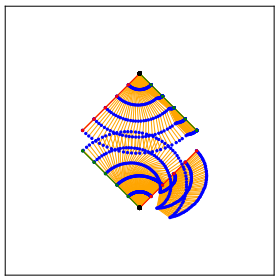
(a) Number of links: 2.



(b) Number of links: 3.



(c) Number of links: 4.



(d) Number of links: 5.

Figure 6.3: The number of links of articulated planar robots is changed from 2 to 5 while keeping the lengths of the planar robots equal. The base coordinates of the robots are $(0, 5)$ and $(0, -5)$.

In each scenario, prioritized-PRM and CBS-PRM are executed 10 times. The same performance metrics as the previous experiment are recorded. The number of nodes is set to 400, with the *maxdist* parameter calibrated to maintain a steady degree and quantity of edges. As the dimension of X increases, the sampling density decreases. The average values of the performance metrics are listed in Table 6.4 and Table 6.5. Complete data sets are shown in Figure 6.4.

Table 6.4: Prioritized-PRM performance metrics for different number of links.

N	T_L	T_Q	**Nodes**	**Edges**	**Degree**	**Maxdist**	**Success**	**SOC**
2	3.49	1.09	400	1436	7.18	0.7	0.9	7.34
3	8.04	4.99	400	1521	7.61	1.4	0.9	9.82
4	3.52	12.24	400	1503	7.52	2.1	0.6	11.63
5	6.81	18.21	400	1750	8.75	2.8	0.8	11.30

Table 6.5: CBS-PRM performance metrics for different number of links.

N	T_L	T_Q	Nodes	Edges	Degree	Maxdist	Success	SOC	CT
2	3.89	4.06	400	1508	7.46	0.7	1.0	5.14	12
3	8.04	3.65	400	1562	7.73	1.4	1.0	4.99	11
4	3.09	2.95	400	1545	7.65	2.1	1.0	6.00	9
5	6.31	1.67	400	1803	8.93	2.8	1.0	7.05	5

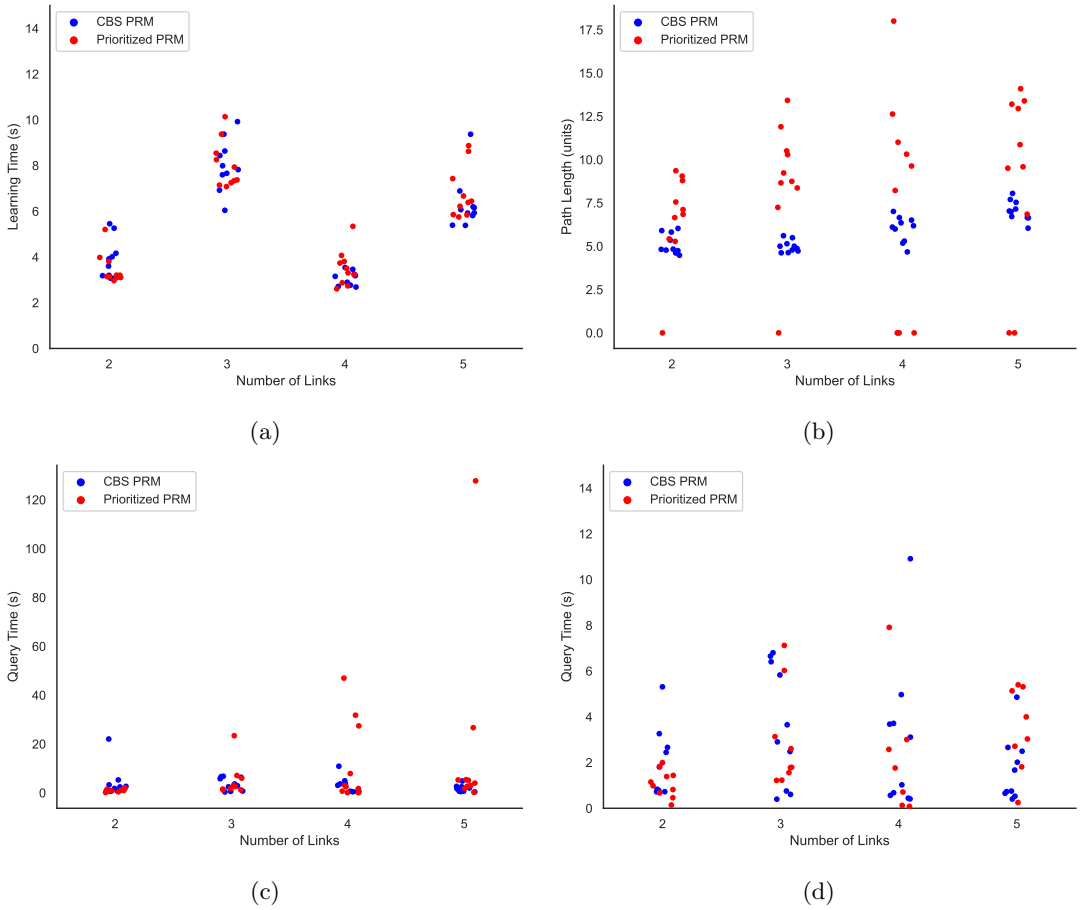
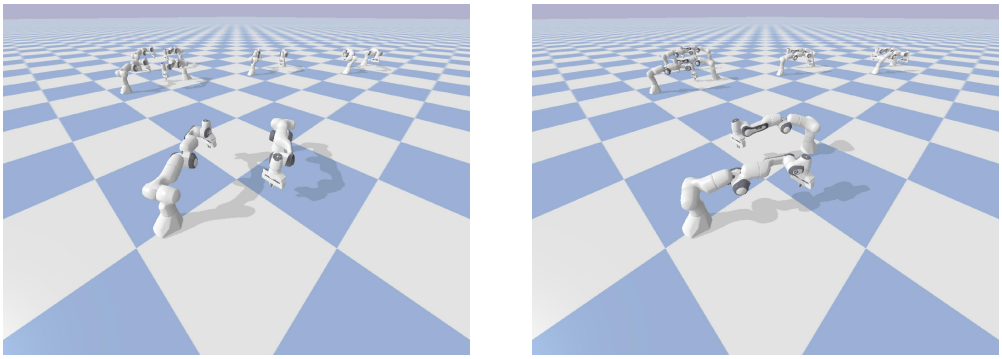


Figure 6.4: Data sets for learning time (a), sum of costs (b), and query time (c and d) for the second experiment (see Figure 6.3) in which the number of links of the manipulators is varied.

As the number of links increases, the query time for prioritized-PRM shows more outliers due to longer edges and more collision checks. Furthermore, as the number of links increases, so does the potential for collisions between robots. Conversely, CBS-PRM maintains a steady or slightly decreasing query time as the graph becomes sparser, requiring fewer constraints to find a solution. This is evidenced by the low CT node count and the linear correlation between query time and CT nodes. CBS-PRM consistently achieves a higher success rate and lower cost paths than prioritized-PRM as seen in Figure 6.4b.

6.3 Increasing the number of manipulators

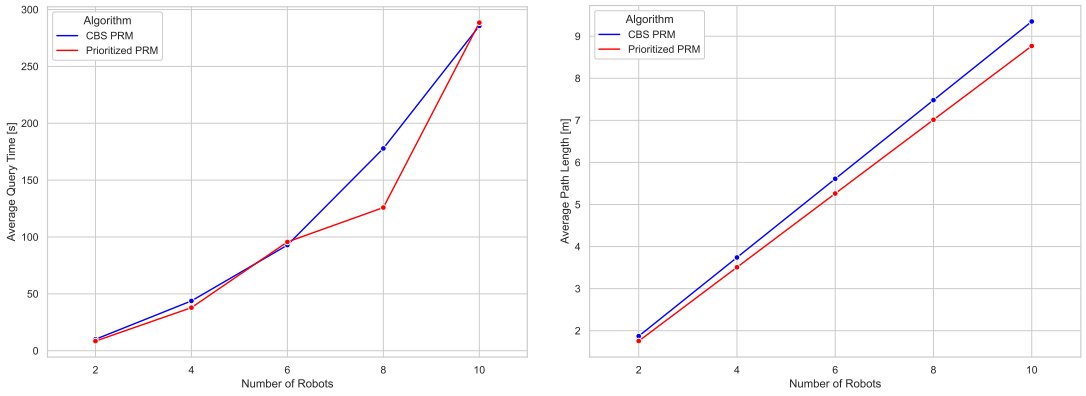
The objective of this experiment is to study the scalability of the CBS-PRM algorithm with respect to the number of robots and compare it to prioritized-PRM. Conducted in Pybullet using Franka Emika Panda manipulators arranged in pairs, setups with 2, 4, 6, 8, and 10 manipulators were tested. The setup with 10 manipulators is shown in Figure 6.5.



(a) (b)

Figure 6.5: Setup for testing the scalability of CBS-PRM in Pybullet. (a) Start poses. (b) Goal poses. The shortest path involves moving the 1^{st} joint and compensating with the 7^{th} joint to maintain the end effector orientation. This movement results in collisions between end effectors.

The same predetermined roadmap from Figure 4.4 is used for each manipulator in this and the following experiment in Section 6.4. Thus the path found for a manipulator moving from the same start to the same goal pose will be consistent across repeated experiments. This roadmap has an average connectivity of approximately 6.7 neighbors. In all experiments, the algorithm successfully identified paths for all manipulators. Figure 6.6 shows the query time and total path lengths as a function of the number of manipulators.



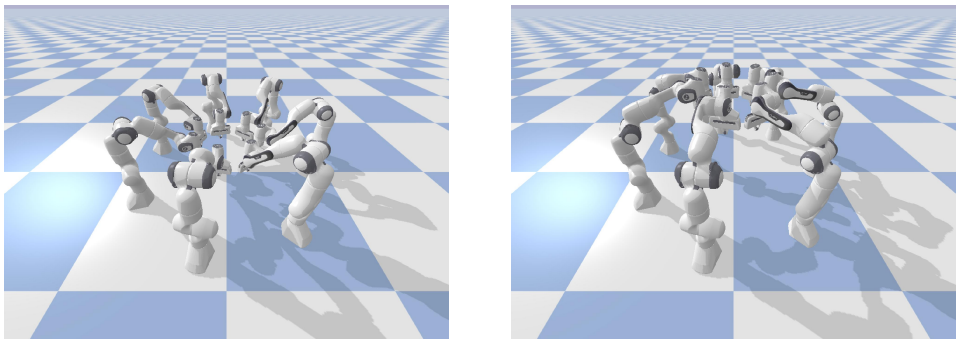
(a) (b)

Figure 6.6: (a) Query time vs. number of robots. (b) Path lengths vs. number of robots. Query times are comparable and path lengths are nearly identical. CBS-PRM paths are slightly longer.

Figure 6.6 shows that query times and path lengths are comparable for both algorithms. This signifies the similarity between the two algorithms when few conflicts occur. However, CBS-PRM paths are slightly longer than those of prioritized-PRM, likely due to the use of a simplified distance metric for path length calculation.

6.4 Stress testing

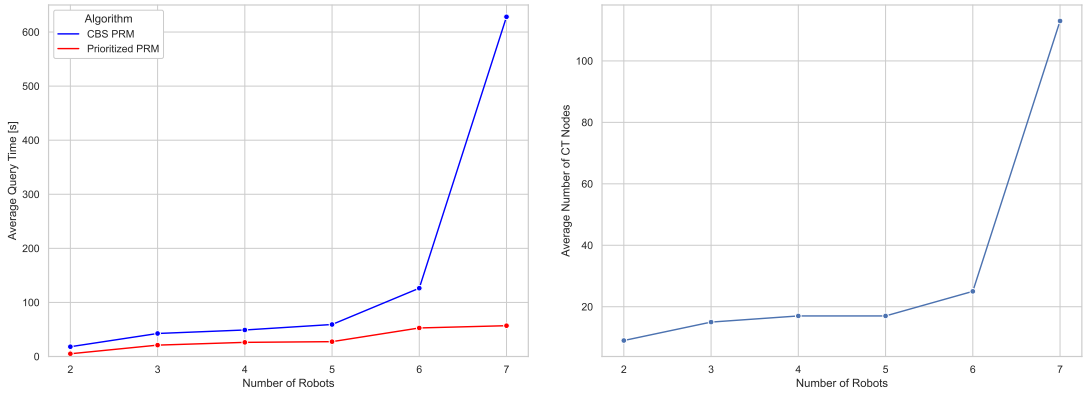
The goal of this experiment is to evaluate the performance limits of CBS-PRM. Manipulators are placed in a circle with end effectors near the center. The task requires each robot to move its end effector upward without collisions. The setup with 7 manipulators is shown in Figure 6.7.



(a) (b)

Figure 6.7: Setup for stress testing CBS-PRM in Pybullet. (a) Start poses. (b) Goal poses. The shortest path involves moving the 4^{th} joint and compensating with the 6^{th} joint to maintain the end effector orientation. This movement results in collisions between the end effectors.

Across all runs, the algorithms successfully found paths for all manipulators. Figure 6.8 shows the computational resources required as a function of the number of manipulators.



(a) (b)

Figure 6.8: (a) Query time vs. number of robots. (b) Number of CT nodes vs. number of robots. The query time for CBS-PRM is proportional to the number of CT nodes. In dense scenarios, CBS-PRM requires much more time compared to prioritized-PRM due to conflict resolution.

Figure 6.8 shows that the query time for CBS-PRM is directly proportional to the number of expanded CT nodes. This figure also demonstrates that CBS-PRM requires significantly more computation time and resources than prioritized-PRM, especially when a cascade of CT nodes must be expanded. This occurs when resolving one conflict with a new constraint likely leads to additional conflicts. In this setup, new constraints frequently lead to further conflicts because one conflicting manipulator is often surrounded by others, limiting its options for resolving the conflict. Figure 6.9 compares the path lengths computed by CBS-PRM and prioritized-PRM.

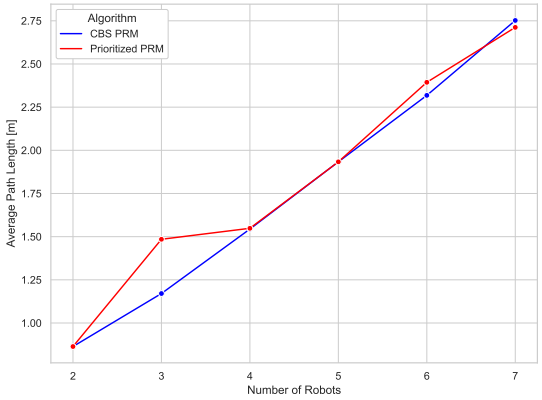


Figure 6.9: Comparison of path lengths computed by CBS-PRM and prioritized-PRM. While CBS-PRM takes longer to compute paths, it produces shorter paths and has a higher success rate compared to prioritized-PRM, demonstrating its superiority in finding efficient paths.

As illustrated in Figure 6.9, although CBS-PRM takes longer to compute paths, it produces shorter paths compared to prioritized-PRM. Combined with its higher success rate, as demonstrated in previous experiments, this highlights the superiority of CBS-PRM in computing efficient paths.

6.5 Conclusion

Overall, CBS-PRM achieves higher success rates and computes lower cost paths compared to prioritized-PRM, as evidenced by Figure 6.4b and Figure 6.9. Figure 6.8 illustrates that CBS-PRM requires significantly more computation time than prioritized-PRM, especially in increasingly dense scenarios where a cascade of CT nodes must be expanded. This occurs when resolving one conflict with a new constraint likely leads to additional conflicts, for instance when one conflicting manipulator is enclosed by others, limiting its options for resolving the conflict. In terms of space requirements, CBS-PRM needs more memory than prioritized-PRM because it also stores nodes in the conflict tree, in addition to the roadmap. This additional memory usage is justified by the algorithm's ability to maintain a high success rate and compute shorter paths. The experiments in Section 6.2, Section 6.3, and Section 6.4 demonstrate the scalability of CBS-PRM. The resources required by the algorithm scale similarly with the number of manipulators as for prioritized-PRM, as long as conflicts are easily resolved and not too many CT nodes have to be expanded. In summary, the experiments underscore the superiority of CBS-PRM over prioritized-PRM in computing shorter paths with higher success rates. While CBS-PRM generally requires more resources to compute these shorter paths, these metrics scale similarly to those of prioritized-PRM, demonstrating its good scalability performance.

Chapter 7

Conclusion

7.1 Summary

This work focused on improving the efficiency of coupled and decoupled approaches by exploring a hybrid approach for multi-manipulator motion planning, particularly for pick-and-place tasks. Efficiency was measured in terms of computation time, memory usage, success rates, Sum Of Costs (SOC) of the paths, and scalability with increasing number of DOFs. A crucial aspect of this research involved a comparative analysis of these metrics across different algorithms and scenarios to test their strengths and also limitations. Initial experiments used planar manipulators in close proximity, while later experiments tested the scalability of the algorithms with multiple Franka Emika Panda manipulators in Pybullet simulations. The coupled approach, distance-PRM, demonstrated poor scalability with increased dimensionality of the composite configuration space, suffering from exponential increases in computation time and memory usage. The decoupled approach, prioritized-PRM, had significantly reduced computation times and better scalability than coupled approaches (see Chapter 6). However, decoupled approaches are incomplete, resulting in lower success rates and generally higher SOC for the paths found, compared to the coupled approach. To address these issues, a new hybrid approach, CBS-PRM, was developed. This approach initially attempts to solve the decoupled problem and only addresses the coupled problem when necessary. It combines Conflict-Based Search (CBS) as a hybrid Multi-Agent Graph Search (MAGS) algorithm at the low level and Probabilistic Roadmaps (PRM) as a sampling-based path planner at the high level. In this way, the principles of CBS are extended from a discrete search on a graph to the continuous search space of motion planning. CBS-PRM proved more efficient than distance-PRM, the coupled approach, in terms of computational resources and outperformed prioritized-PRM, the decoupled approach, in success rates and SOC of the paths. While CBS-PRM generally required more resources, its scalability was comparable to prioritized-PRM, as long as conflicts are easily resolved and not too many CT nodes have to be expanded. When this is not the case, the performance of CBS-PRM is subject to larger variations compared to prioritized-PRM which is the main limitation of CBS-PRM. This happens, for instance, in situations in which one conflicting manipulator is enclosed by others, limiting its options for resolving the conflict. Overall, the experiments highlighted the superiority of CBS-PRM over prioritized-PRM in computing shorter paths with higher success rates. While CBS-PRM generally requires more resources to compute these shorter paths, these metrics scale similarly to those of prioritized-PRM, demonstrating its good scalability performance.

As part of this research, a multi-robot motion planning library was developed that includes all the PRM-based algorithms discussed and with which motion planning simulations can be performed in Pybullet, particularly for multi-manipulator setups and pick-and-place tasks. This multi-robot motion planning library can be found on GitHub.

7.2 Future work

Future research could explore several promising directions to improve the performance of CBS-PRM. The performance is intrinsically linked to the construction of the graph during the learning phase. One strategy could be to adopt obstacle-based sampling, which samples close to the boundary of the free configuration space, facilitating planning through narrow passages. Alternatively, clearance-based sampling could be employed to increase the distance between the robot and the environment, a particularly relevant approach given the primary goal of avoiding collisions between robots. Considering that sparser graphs reduce memory usage and lower query times, the SPARS framework (Dobson & Bekris, 2014), available in the Open Motion Planning Library (OMPL), could be used to construct the graph during the learning phase.

Further improvements could also be made to the performance of the low-level search algorithm used. The duration of the low-level search is highly dependent on the number of collision checks performed. Collision checks are performed with a predefined step size. If this step size is too large, collisions may not be detected, but a smaller step size increases computational overhead and extends path computation times. A dynamic adjustment of the granularity of time discretization based on the complexity of the robot's motion or proximity to other robots could be an interesting approach. This could reduce unnecessary calculations in lower-complexity areas while directing computational efforts toward potential conflict zones.

Instead of using a single graph, multiple layers of graphs, ranging from sparse to dense, could be used to resolve conflicts. CBS could be performed in the sparse graph, with denser graphs being used whenever no solution is found for the sparser graph. CBS would not need to restart from the beginning, but instead, constraints could be inherited from sparser graphs. This would guide the search in denser graphs in some way, intertwining CBS and graph expansion.

In many scenarios, not all robots are capable of colliding. Therefore, subgroups of robots that can collide with each other could be identified at the start of the search, and CBS could be performed for each of these subgroups independently.

Also learning from past experiences can be used in multiple ways to enhance performance. Sampling can be biased, utilizing a neural network architecture such as the LSTM sampler used in the work of Ying et al. (2021). At the low level, interesting could be to predict possible conflicts and constraints for CBS by learning from past experiences to guide the search.

Last but not least, the integration with task planning could be explored. This could involve the use of machine learning techniques to predict and optimize task sequences, ensuring that the most efficient paths are chosen while considering potential conflicts and constraints. By incorporating task planning, robots could achieve more complex objectives, leading to more robust, adaptive, and intelligent manipulator systems.

Bibliography

Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., Vallejo, D., et al. (1998). Obprm: An obstacle-based prm for 3d workspaces. *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, 155–168.

Barraquand, J., & Latombe, J.-C. (1990). A monte-carlo algorithm for path planning with many degrees of freedom. *Proceedings., IEEE International Conference on Robotics and Automation*, 1712–1717.

Bohlin, R., & Kavraki, L. E. (2000). Path planning using lazy prm. *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, *1*, 521–528.

Boor, V., Overmars, M. H., & Van Der Stappen, A. F. (1999). The gaussian sampling strategy for probabilistic roadmap planners. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, *2*, 1018–1023.

Buss, S. R. (2004). Introduction to inverse kinematics with jacobian transpose. *Pseudoinverse and Damped Least Squares methods*, 19.

Canny, J. F. (1988). *The complexity of robot motion planning*. The MIT Press.

Čáp, M., Novák, P., Kleiner, A., & Seleckỳ, M. (2015). Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering*, *12*(3), 835–849.

Cheng, R., Shankar, K., & Burdick, J. W. (2020). Learning an optimal sampling distribution for efficient motion planning. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 7485–7492.

Choset, H. (2000). *Chapter 3: Configuration space* (tech. rep.) (Accessed: 2024-08-05). Carnegie Mellon University. https://www.cs.cmu.edu/~motionplanning/lecture/Chap3-Config-Space_howie.pdf

Choudhury, S., Gammell, J. D., Barfoot, T. D., Srinivasa, S. S., & Scherer, S. (2016). Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 4207–4214.

Cohen, B., Chitta, S., & Likhachev, M. (2014). Single-and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*, *33*(2), 305–320.

Coulombe, A., & Lin, H.-C. (2020). High precision real time collision detection. *arXiv preprint arXiv:2007.12045*.

Dobson, A., & Bekris, K. E. (2014). Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research*, *33*(1), 18–47.

Erdmann, M., & Lozano-Perez, T. (1987). On multiple moving objects. *Algorithmica*, *2*, 477–521.

Faust, A., Oslund, K., Ramirez, O., Francis, A., Tapia, L., Fiser, M., & Davidson, J. (2018). Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and

sampling-based planning. *2018 IEEE international conference on robotics and automation (ICRA)*, 5113–5120.

Franka Emika GmbH. (2024). *Franka control interface documentation* [https://frankaemika.github.io/docs/control_parameters.html]. Retrieved August 7, 2024, from https://frankaemika.github.io/docs/control_parameters.html

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2015). Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *2015 IEEE international conference on robotics and automation (ICRA)*, 3067–3074.

Ha, H., Xu, J., & Song, S. (2020). Learning a decentralized multi-arm motion planner [Last accessed 15 December 2023]. https://multiarm.cs.columbia.edu/

He, Y., & Liu, S. (2021). Analytical inverse kinematics for franka emika panda–a geometrical solver for 7-dof manipulators with unconventional design. *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, 194–199.

He, Y., Li, X., Xu, Z., Zhou, X., & Li, S. (2021). Collaboration of multiple scara robots with guaranteed safety using recurrent neural networks. *Neurocomputing*, *456*, 1–10.

Hsu, D., Jiang, T., Reif, J., & Sun, Z. (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. *2003 IEEE international conference on robotics and automation (cat. no. 03CH37422)*, *3*, 4420–4426.

Hsu, D., Latombe, J.-C., & Motwani, R. (1997). Path planning in expansive configuration spaces. *Proceedings of international conference on robotics and automation*, *3*, 2719–2726.

Ichter, B., Harrison, J., & Pavone, M. (2018). Learning sampling distributions for robot motion planning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7087–7094.

Janson, L., Ichter, B., & Pavone, M. (2018). Deterministic sampling-based motion planning: Optimality, complexity, and performance. *The International Journal of Robotics Research*, *37*(1), 46–61.

Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, *34*(7), 883–921.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, *30*(7), 846–894.

Kavraki, L., Svestka, P., Latombe, J.-C., & Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580. https://doi.org/10.1109/70.508439

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, *12*(4), 566–580.

Khamis, A., Hussein, A., & Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art. In A. Koubâa & J. Martínez-de Dios (Eds.), *Cooperative robots and sensor networks 2015* (pp. 31–51). Springer International Publishing. https://doi.org/10.1007/978-3-319-18299-5_2

Klosowski, J. T., Held, M., Mitchell, J. S., Sowizral, H., & Zikan, K. (1998). Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE transactions on Visualization and Computer Graphics*, *4*(1), 21–36.

Kuffner, J. J., & LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, *2*, 995–1001.

LaValle, S., & Kuffner, J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, *20*(5), 378–400. https://doi.org/10.1177/02783640122067453

LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*.

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

Liu, G., De Winter, J., Steckelmacher, D., Hota, R. K., Nowe, A., & Vanderborght, B. (2023). Synergistic task and motion planning with reinforcement learning-based non-prehensile actions. *IEEE Robotics and Automation Letters*, *8*(5), 2764–2771.

Marcucci, T., Petersen, M., von Wrangel, D., & Tedrake, R. (2023). Motion planning around obstacles with convex optimization. *Science robotics*, *8*(84), eadf7843.

Mazer, E., Ahuactzin, J. M., & Bessiere, P. (1998). The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, *9*, 295–316.

Orthey, A., Chamzas, C., & Kavraki, L. E. (2023). Sampling-based motion planning: A comparative review. *Annual Review of Control, Robotics, and Autonomous Systems*, *7*.

Prianto, E., Kim, M., Park, J.-H., Bae, J.-H., & Kim, J.-S. (2020). Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay. *Sensors*, *20*(20), 5911.

Qureshi, A. H., Miao, Y., Simeonov, A., & Yip, M. C. (2020). Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, *37*(1), 48–66.

Ratliff, N., Zucker, M., Bagnell, J. A., & Srinivasa, S. (2009). Chomp: Gradient optimization techniques for efficient motion planning. *2009 IEEE International Conference on Robotics and Automation*, 489–494. https://doi.org/10.1109/ROBOT.2009.5152817

Reif, J. H. (1979). Complexity of the mover's problem and generalizations. *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, 421–427. https://doi.org/10.1109/SFCS.1979.10

Reig, S., Carter, E. J., Fong, T., Forlizzi, J., & Steinfeld, A. (2021). Flailing, hailing, prevailing: Perceptions of multi-robot failure recovery strategies. *2021 16th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 158–167.

Rickert, M., Sieverling, A., & Brock, O. (2014). Balancing exploration and exploitation in sampling-based motion planning. *IEEE Transactions on Robotics*, *30*, 1305–1317. https://doi.org/10.1109/TRO.2014.2340191

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., & Abbeel, P. (2014a). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, *33*(9), 1251–1270.

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., & Abbeel, P. (2014b). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, *33*, 1251–1270. https://doi.org/10.1177/0278364914528132

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, *219*, 40–66.

Sharon, G., Stern, R., Goldenberg, M., & Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence*, *195*, 470–495.

Solis, I., Motes, J., Sandström, R., & Amato, N. M. (2021). Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics and Automation Letters*, *6*(3), 4608–4615.

Standley, T. (2010). Finding optimal solutions to cooperative pathfinding problems. *Proceedings of the AAAI conference on artificial intelligence*, *24*(1), 173–178.

Standley, T., & Korf, R. (2011). Complete algorithms for cooperative pathfinding problems. *IJCAI*, 668–673.

Stern, R. (2019). Multi-agent path finding–an overview. *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*, 96–115.

Strub, M. P., & Gammell, J. D. (2020). Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 3191–3198.

Şucan, I. A., Moll, M., & Kavraki, L. E. (2012). The Open Motion Planning Library [https://ompl.kavrakilab.org]. *IEEE Robotics & Automation Magazine*, *19*(4), 72–82. https://doi.org/10.1109/MRA.2012.2205651

Tamizi, M. G., Yaghoubi, M., & Najjaran, H. (2023). A review of recent trend in motion planning of industrial robots. *International Journal of Intelligent Robotics and Applications*, 1–22.

Van Den Berg, J. P., & Overmars, M. H. (2005). Prioritized motion planning for multiple robots. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 430–435.

Verginis, C. K., Dimarogonas, D. V., & Kavraki, L. E. (2022). Kdf: Kinodynamic motion planning via geometric sampling-based algorithms and funnel control. *IEEE Transactions on robotics*, *39*(2), 978–997.

Wagner, G., & Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. *2011 IEEE/RSJ international conference on intelligent robots and systems*, 3260–3267.

Wagner, G., & Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial intelligence*, *219*, 1–24.

Wang, J., Chi, W., Li, C., Wang, C., & Meng, M. Q.-H. (2020). Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, *17*(4), 1748–1758.

Warren, C. W. (1989). Global path planning using artificial potential fields. *1989 IEEE International Conference on Robotics and Automation*, 316–317.

Wilmarth, S. A., Amato, N. M., & Stiller, P. F. (1999). Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, *2*, 1024–1031.

Ying, K.-C., Pourhejazy, P., Cheng, C.-Y., & Cai, Z.-Y. (2021). Deep learning-based optimization for motion planning of dual-arm assembly robots. *Computers & Industrial Engineering*, *160*, 107603.

Zhang, S., Li, J., Huang, T., Koenig, S., & Dilkina, B. (2022). Learning a priority ordering for prioritized planning in multi-agent path finding. *Proceedings of the International Symposium on Combinatorial Search*, *15*(1), 208–216.

Appendix A

Search algorithms

Algorithm 3 Dijkstra's Algorithm

Require: A graph $G = (N, E)$, initial node n_{init}
Ensure: Shortest path from initial node n_{init} to all other nodes reachable from n_{init}

```
1:  initialize dist[n] = ∞ for all n ∈ N
2:  initialize prev[n] = NONE for all n ∈ N
3:  dist[n_init] = 0
4:  initialize set of unvisited nodes U = N
5:  while U is not empty do
6:      u = n ∈ U with min dist[u]
7:      remove u from U
8:      break if u = n_goal or if dist[u] = ∞
9:      for all neighboring n of u still in U do
10:         alt = dist[u] + cost(u, n)
11:         if alt < dist[n] then
12:             dist[n] = alt
13:             prev[n] = u
14:         end if
15:     end for
16: end while
17: return dist, prev
```

Understanding these algorithms is crucial for discussions about the efficiency and performance of graph-based motion planners. Dijkstra's algorithm starts by assigning a distance of "infinity" (INF) to all nodes, except for the start node, which gets a distance of 0. This represents the shortest known distance from the start node to each other node. Then it initializes a previous node tracking system, which helps in reconstructing the path once the shortest distances are found. Initially, all previous nodes are set to None. Additionally, a set called unvisited nodes, keeps track of all nodes that haven't been visited yet. While there are still nodes to visit, the node from the unvisited set with the shortest path length is selected. This node is removed from the set of unvisited nodes. If the smallest distance is infinity, the remaining nodes are not reachable from the start node, so the loop is broken. If the selected node is the goal node, the destination has been reached, and the algorithm can stop early. For each neighbor of the selected

Algorithm 4 A* Algorithm

Require: A graph $G = (N, E)$, a heuristic function $h(n)$, initial node n_{init}, goal node n_{goal}
Ensure: Shortest path from n_{init} to n_{goal} if reachable

initialize dist$[n] = \infty$ **for all** $n \in N$
initialize est_total_cost$[n] = \infty$ **for all** $n \in N$
initialize prev$[n]$ = NONE **for all** $n \in N$
dist$[n_{\text{init}}] = 0$
est_total_cost$[n_{\text{init}}] = h(n_{\text{init}})$
initialize set of unvisited nodes $U = N$
while U is not empty **do**
    $u = n \in U$ with min est_total_cost$[u]$
    remove u from U
    break if $u = n_{\text{goal}}$ or if dist$[u] = \infty$
    for all neighboring n of u still in U **do**
        alt = dist$[u]$ + cost(u, n)
        if alt $<$ dist$[n]$ **then**
            dist$[n]$ = alt
            est_total_cost$[n]$ = dist$[n]$ + $h(n)$
            prev$[n] = u$
        end if
    end for
end while
return dist, prev

node, the distance to the neighbor is calculated using a defined distance function. If moving to this neighbor is shorter than any previously known path to this neighbor, update the shortest distance to this neighbor and record the selected node as the previous node on the path to this neighbor. The algorithm thus completes when the goal node has been found, no other nodes are reachable or all the nodes have been searched. Once this happens, you can reconstruct the shortest path from the start node to the goal node by backtracking from the goal node using the previous node references. The A* algorithm, on the other hand, introduces a heuristic to guide the search, which can make it more efficient in many cases. It starts similarly to Dijkstra's algorithm, by assigning a distance of 0 to the start node and infinity to all other nodes. It also maintains a set of unvisited nodes and a system to track the previous node on the shortest path to each node. However, the key difference lies in how the A* algorithm selects the next node to visit. Instead of choosing the node with the shortest known distance from the start, as Dijkstra's algorithm does, the A* algorithm selects the node that is estimated to have the lowest total path cost to the goal.

Appendix B

Probabilistic Roadmaps

The PRM planner works in two phases. The first phase is called the learning phase. During this phase a data structure called the *roadmap* is constructed in a probabilistic way for a given scene. The roadmap $R = (N, E)$ is an undirected graph consisting of a set of nodes and edges connecting them. The nodes in N are a set of configurations of the robot over $\mathcal{Q}_F$. The edges (q_1, q_2) in E are valid paths of movement connecting the configurations q_1 and q_2. These local paths are computed by the fast and computational less expensive local planner. Recomputing the local paths is inexpensive, therefore local paths are not explicitly stored in the roadmap to save a considerable amount of space. The learning phase is entirely performed before any path planning query. In the query phase, the roadmap is searched to solve path planning problems for the given input scene. Given a start/initial configuration q_{init} and a goal configuration q_{goal}, the method first tries to connect q_{init} and q_{goal} to two nodes n_{init} and n_{goal} in N. If successful, the actual query takes place and R is searched for a sequence of edges in E connecting n_{init} to n_{goal}. Finally, this sequence of edges is transformed into a feasible path for the robot by recomputing the local path for each edge.

The learning phase

In the learning phase, the planner tries to learn the connectivity of the free configuration space. The logic behind constructing the roadmap is detailed in Algorithm 5.

Algorithm 5 PRM

```
1: G = ∅
2: while Termination Condition is false do
3:     q_new ← valid sample from X_F
4:     Add q_new to nodes N of G
5:     N_c(q_new) ← K closest candidate neighbors of q_new
6:     for each q_near ∈ N_c(q_new) do
7:         e ← local path q_new to q_near
8:         if e ∈ X_F and e ∉ edges E of G then
9:             Add e to edges E of G
10:        end if
11:    end for
12: end while
13: return G
```

Some parts of this pseudocode need to be further detailed. Specifically, we need an answer to the following questions: When to terminate the learning phase? How do we sample configurations in X_{F}? How do we determine the nearest neighbors already in G of the newly sampled node? This also requires a certain distance function to define. Finally, how do we connect the newly sampled node to the neighbors?

Termination condition The termination condition of the learning phase is typically dictated by a predetermined number of iterations or a specified timeout. Both were implemented in Python. The value of these parameters, i.e. the number of points or the time limit, have to be chosen based on the complexity of the problem and the computational resources available. It is important to note that the quality of the roadmap, which directly impacts the effectiveness of the PRM planner, is heavily influenced by the density of the sampled configurations. A higher density typically leads to a more accurate representation of the free space, which can result in more efficient and accurate path planning. However, a higher density also means more computational resources are required, both in terms of memory to store the roadmap and processing power to construct it and search it. Therefore, choosing the termination condition for the learning phase is a balancing act. It should be set such that it allows the roadmap to be dense enough for accurate path planning, but not so dense that it becomes computationally prohibitive. Alternatively, the learning phase and the query phase can also be interleaved. This means that at a certain point in the learning phase, the constructed roadmap is queried for a solution. If no solution is found, the algorithm continues learning the connectivity of X_F. Otherwise the algorithm stops its operation and the solution is returned. In such cases, the termination condition is the identification of a valid solution to the motion planning query.

Sampling Initially, the graph $G = (N, E)$ is empty. To approximate the connectivity of X_F, the nodes of G are sampled uniformly over X or equivalently sampling is performed uniformly over the joint interval for each joint. This is known as unbiased sampling. Biased sampling on the other hand, alters the probability distribution to favor sampling of interesting regions of the composite configuration space (Orthey et al., 2023). Various biases can be employed including obstacle-based (Amato et al., 1998), clearance-based (Wilmarth et al., 1999) and deterministic sampling (Janson et al., 2018). Obstacle-based sampling biases the sampling towards obstacles, improving the planning in narrow passages. Examples are the Gaussian sampling method (Boor et al., 1999) and the bridge-based sampling method (Hsu et al., 2003). Clearance-based sampling aims to increase the distance between robots and obstacles. For instance, first a feasible configuration can be sampled and next random steps can be taken to increase the distance to obstacles (Verginis et al., 2022). Sampling in a deterministic way generates each time the same samples. A sampling distribution can be learned from demonstrations, and then used to bias sampling. (Ichter et al., 2018) use a conditional variational autoencoder to compute the sampling distribution. Also Halton sequences and Sukharev grids can minimize the largest unsampled area in the configuration space (S. M. LaValle, 2006). Next, the sampled configuration is checked for collision. Collision checking of a configuration entails both checking for collisions between different parts of the robot itself and collisions of the robot with obstacles or other robots. This can be done using various techniques such as the Gilbert-Johnson-Keerthi algorithm (Coulombe & Lin, 2020) and Bounding Volume Hierarchies such as Bounding Spheres or Axis-Aligned Bounding Boxes (Klosowski et al., 1998). For the planar manipulators regarded in this work the collision checking can be done analytically using very fast but specific collision checking methods (see **??**). If the configuration is collision-free, it is added to N and discarded otherwise.

Node neighbors For a newly found collision-free node, the roadmap is searched for a number of candidate neighboring nodes from the current N to which the new node will try to be connected using the local planner. The selection of these potential neighbors can happen in a number of ways. In the original paper introducing the PRM method (L. E. Kavraki et al., 1996), the set of potential neighbors N_p of a newly sampled node n_{new} is defined as

$$N_p = \{n_{\text{near}} \in N \mid D(n_{\text{new}}, n_{\text{near}}) \leq maxdist\} \tag{B.1}$$

Ideally, n_{new} is connected to nodes in N_p in order of increasing distance and nodes connected to the same connected component are ignored. Connecting the same node to two different nodes of the same connected component does not lead to any additional solutions but is not wrong per se. It only affects the efficiency of the search in the query phase. For this work this check was not implemented possibly resulting in shorter learning times and slightly longer query times. In a way, this can however be compensated by changing predefined parameters discussed later on. It is important to keep in mind that the amount of calls to the local planner dominates the learning time. The idea behind evaluating the potential neighbors in this way is that the larger the relative distance between n_{new} and n_{near}, the less likely the local planner will find a collision-free path. Thus defining N_p based on $maxdist$ and evaluating with increasing distance will ensure that the most likely successful connections are prioritized for evaluation. For further decreasing the number of calls to the local planner and thus the learning time, both the amount of potential neighbors to evaluate, so the size of N_p, and the amount of actual neighbors can be upper bounded. Let's denote these upper bounds by k_1 and k_2 respectively. By bounding the size of N_p by k_1, is ensured that the worst case runtime of each iteration in the learning phase is independent of the current size of the constructed graph G. Now the number of calls to the local planner scales linearly with the amount of nodes in graph G. The version of PRM without k_2 specified is called *distance PRM* in this work. The method which uses an upper bound k_2 is called *degree PRM* as it limits the degree or branching factor for each node in the graph. Both versions were implemented for this work and are visualized in Figure B.1.

Distance function For constructing and sorting the set N_p of potential neighbors for each new node n_{new}, a distance function D is used. The further away n_{near} from n_{new}, the lower the chance that the local planner will find a valid edge connecting both nodes. The distance $D(n_{\text{new}}, n_{\text{near}})$ thus has to reflect the chance that the local planner will *fail* to compute a feasible path between two configurations. The most straightforward measure of distance one could use is the euclidean distance between configurations in the composite configuration space

$$D(n_{\text{new}}, n_{\text{near}}) = \|q_{\text{new}} - q_{\text{near}}\| \tag{B.2}$$

where q_{new} is the configuration associated to the node n_{new} and q_{near} is the configuration associated to the node n_{near}. This is typically the distance function used when no geometric information about the robot is available a priori. However, for the use of a motion planner on a specific robot, it is interesting to design the distance function with more care as the distance function has to be a measure of the distance between two configurations in the workspace of the robot, not the configuration space. As they will be used later on, let's take the example of a planar articulated RRR robot for instance. The configuration space of the RRR robot is a 3-dimensional torus since the joint angles are periodic. If the joint limits are specified, the configuration space reduces to a domain on this 3-dimensional torus. For this domain there exists a bijective mapping between this domain and a 3-dimensional block whose sides correspond to the joint intervals. Starting from a certain configuration, the configurations obtained by changing the joint angle with 90° of the joint at the base of the robot or changing the joint angle with 90° of the joint steering the

last link of the robot, will result in two different configurations that are equally distant from the configuration started from. It is clear however that in the workspace, the first change of joint angle leads to a larger sweep of the robot. Thus the definition of D has to be a measure of the area or volume of the workspace region swept by the robot when it moves along the local path between n_{new} and n_{near} in the absence of obstacles. However, computing of the swept areas or volumes is more time-consuming then using a rough estimate of the swept-region. This estimate can for instance be

$$D(n_{\text{new}}, n_{\text{near}}) = \max_{p \in \text{robot}} \|p_{\text{new}} - p_{\text{near}}\| \tag{B.3}$$

where p denotes a point on the robot, p_{new} is the position of p in the workspace when the robot is at configuration q_{new}.

Local planner The purpose of the local planner is to evaluate whether or not the new valid sample n_{new} can be connected with its potential neighbors n_{near} in N_p. The planner is called local because the samples are connected over short distances using simple path segments which are usually straight lines in the configuration space. Next to evaluating the validity of a potential edge correctly, the local planner also has to be fast. The amount of calculations for collision checking dominates the learning phase and the learning time mainly depends on the local planner and the amount of calls to it. On the other hand, the whole idea behind the learning phase is to reduce the path planning during the querying phase significantly. At the start of a query, the start and goal configurations have to be connected to the graph constructed in the learning phase. Thus a very fast local planner is necessary to ensure that the query happens quasi-instantaneously. Depending on the formulation of the motion planning problem, the local planner might have to take other constraints than collisions into account. For instance differential constraints in the case of kinodynamic motion planning. Also efficient and optimal local planners are an area of research (Choudhury et al., 2016; Faust et al., 2018).

The query phase

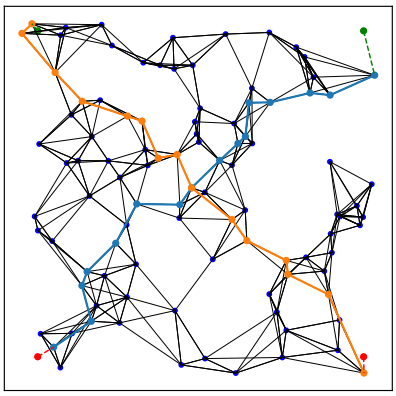
The second phase is the query phase in which the start and goal locations for the agents are specified. The purpose of PRM is that it can be used for many queries using the same graph and so explicitly adding the start and goal nodes to the graph and rebuilding the graph again would not make sense. Instead the closest nodes to the start and goal configurations are taken as the start and goal nodes of the query. If the local planner does not find feasible paths to the closest node, we go on to the second closest node and so on. Nodes further away than *maxdist* can be chosen to be ignored since the chance of failure of the planner increases with distance. Now a graph search algorithm is used to find a path from start to goal node. Popular choices for graph search problems are Dijkstra and A*. Both search methods are optimal and return the shortest path to the goal in the graph. A* uses the euclidean distance from a certain node to the goal as a heuristic in the search. Without leveraging this heuristic, A* search reduces to the Dijkstra algorithm.

In Figure B.1, two agents are introduced with each a red start node and a green goal node. The blue path is returned as the shortest path for agent 1 and the orange path is returned as the shortest path for agent 2. Both Dijkstra and A* were used to compute these shortest paths and a difference in performance was noted, which is shown in Table B.1.

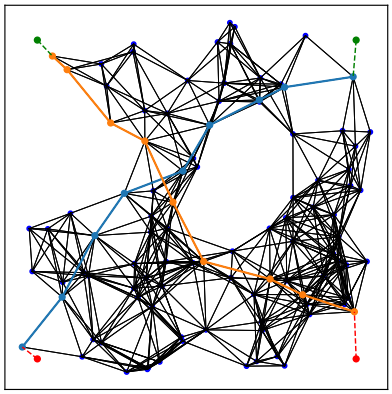
Table B.1: Performance comparison between Dijkstra search and A* search on both k-nearest neighbor and distance-based PRM.

PRM	Method	Agent	Time (seconds)	Distance (units)
k-nearest	Dijkstra	Agent 1	0.005556	26.81
k-nearest	A*	Agent 1	0.003087	26.81
k-nearest	Dijkstra	Agent 2	0.006415	29.30
k-nearest	A*	Agent 2	0.004585	29.30
distance	Dijkstra	Agent 1	0.011657	25.64
distance	A*	Agent 1	0.004791	25.64
distance	Dijkstra	Agent 2	0.014606	23.77
distance	A*	Agent 2	0.006310	23.77

This experiment shows that in practice A* requires less time for finding the optimal path than Dijkstra thanks to the well-defined heuristic it uses in the search. Dijkstra visits nodes in order of increasing distance, updating the distances to neighboring nodes as it goes. The time complexity of Dijkstra's algorithm is typically $\mathcal{O}(N^2)$ when using a simple array implementation or $\mathcal{O}(E\ log(N))$ using a priority queue, where N is the number of nodes and E is the number of edges in the graph. Instead of visiting nodes in order of their distance from the start, A* visits nodes in order of an estimate of the total path length through that node to the goal. This estimate is the sum of the current shortest known distance from the start to the node (as in Dijkstra's algorithm) and a heuristic estimate of the distance from the node to the goal. The time complexity of A* is also $\mathcal{O}(E\ log(N))$ in the worst case, but in practice it can be much faster if the heuristic is well-chosen. Therefore, A* will be chosen as low-level search method for PRM variants.



(a) Roadmap connecting the k-nearest neighbors.



(b) Roadmap with predefined connection distance.

Figure B.1: Both roadmaps were sampled in a box (with width and height both 20 units) using 100 samples. For connecting the roadmap in Figure B.1a, the value k_2 was taken to be 5. For connecting the roadmap in Figure B.1b, the connection distance $maxdist$ equals 4.

Appendix C

Planar manipulators

The experiments carried out in the plane will involve planar manipulators. Both the number of links and the length of the links will vary depending on the experiment performed. Simulations in 3-dimensional space were done using Pybullet and involve both mobile robots and 7 DOFs Franka Emika Panda manipulators. For the planar manipulators, the forward kinematics and collision detection methods were implemented in Python and for simulations in 3D methods from the Pybullet library were used. The implementation on planar manipulators also has practical relevance as it can be used for planning the motion of SCARA robots. SCARA robots are one of the most widely utilized robots in industrial settings and they have two revolute joints whose axes of rotation are parallel and vertically oriented. The ability to coordinate multiple SCARA robots and to navigate around obstacles could enable collaborative operation on more sophisticated tasks and reduce operation time with respect to execution of the same task by single robot (He et al., 2021). The forward kinematics for planar RRR manipulators are detailed in appendix A and the collision detection methods in appendix B.

A Forward kinematics

Determining the forward kinematics is done using the DH convention. Following this convention, reference frames are assigned to each of the links of the robot. The four reference frames are defined as shown in Figure C.1.

The axis $\vec{z_i}$ is oriented along the axis of joint i+1 (it points upwards out of the plane) and the axis $\vec{x_i}$ is oriented along the common normal between joint axes i and i+1. The corresponding axis $\vec{y_i}$ is chosen accordingly to obtain a right-handed coordinate system.

Based on these frames, the link parameters α_i, a_i, d_i and θ_i are determined. α_i is the twist angle between z_{i-1} and z_i around x_i. a_i is the distance along the common normal between the joint axes i and i+1. d_i is the distance, along the axis of joint i, between the intersection of joint axis i with the common normal coming from joint axes i+1 and i-1. θ_i is the angle between x_{i-1} and x_i around z_{i-1}. Because all three joints are revolute, the forward kinematics of the robot are described by θ_1, θ_2 and θ_3 as the joint parameters. The DH parameters corresponding to Figure C.1 are given in Table C.1.

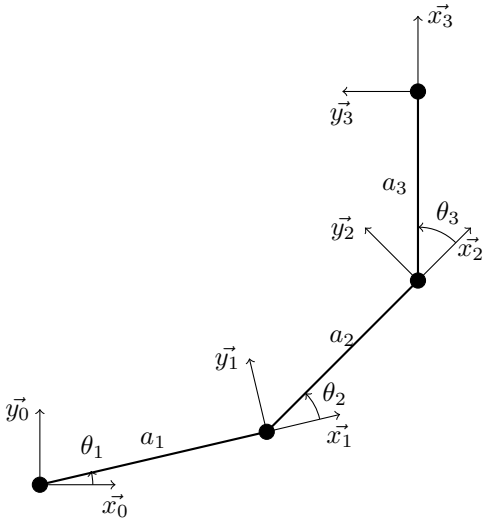


Figure C.1: Definition of the four reference frames for determining the DH parameters of the planar RRR robot.

Link	α_i	a_i	d_i	θ_i
1	0	a_1	0	θ_1
2	0	a_2	0	θ_2
3	0	a_3	0	θ_3

Table C.1: DH parameters of the planar RRR robot.

The homogeneous transformation matrix from frame i to frame i-1 is given by:

$$ {}^{i-1}T_i = \begin{bmatrix} cos(\theta_i) & -cos(\alpha_i)sin(\theta_i) & sin(\alpha_i)sin(\theta_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\alpha_i)cos(\theta_i) & -sin(\alpha_i)cos(\theta_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.1} $$

Using the parameters from Table C.1, the following homogeneous transformation matrices are found for i from 1 to 3.

$$ {}^{i-1}T_i(\alpha_i = 0, a_i, d_i = 0, \theta_i) = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i) & 0 & a_i sin(\theta_i) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.2} $$

The transformation from the end-effector frame to the base frame is given by Equation (C.3). This allows us to calculate the end effector's pose with respect to the base frame using the joint angles.

$$
\begin{aligned}
{}^0T_3(a_1, \theta_1, a_2, \theta_2, a_3, \theta_3) &= {}^0T_1(a_1, \theta_1)\,{}^1T_2(a_2, \theta_2)\,{}^2T_3(a_3, \theta_3)^3 \\
&= \begin{bmatrix} {}^0R_3 & {}^0\vec{p}_3 \\ 0 & 1 \end{bmatrix}
\end{aligned} \tag{C.3}
$$

B Collision detection

Collision detection is done to ensure that a planar manipulator is not in collision with obstacles or with other manipulators. Additionally, no two different links of the same manipulator are allowed to collide. For each revolute joint i, joint limits are set so that θ_i takes on values in the interval $(-\pi, \pi)$. This ensures that no two adjacent links collide. Lastly, the planar manipulators are not allowed to move outside the environment which is the 20 by 20 square depicted in each of the scenes in Figure 6.1, and Figure 6.3. The *robot_collision* function is one comprehensive check for these four types of collisions.

Robot Collision Function

```
def robot_collision(self, robot):
    if not self.robot_in_env_bounds(robot):
        return True, "Robot out of environment bounds"
    if self.robot_self_collision(robot):
        return True, "Robot self collision"
    for obstacle in self.obstacles:
        if self.robot_obstacle_collision(robot, obstacle):
            return True, "Robot obstacle collision"
    for other_robot in self.robot_models:
        if other_robot != robot and self.robot_robot_collision(robot,
        ↪  other_robot):
            return True, "Robot robot collision"
```

The collision detection process involves several key steps to ensure the safe operation of manipulators within a defined 2D workspace. Initially, the algorithm verifies whether each manipulator remains within the environmental boundaries. This is accomplished by checking that the position of each joint, as determined by forward kinematics, falls within the predefined limits of the environment. Following boundary verification, the algorithm proceeds to assess potential self-collisions using the *link_link_collision* function, which will be discussed in further detail later. Since joint limits are designed to prevent collisions between adjacent links, this function specifically checks for collisions among non-adjacent links. The next step is to evaluate interactions with obstacles present in the workspace, which are either rectangular or circular in shape. The *link_link_collision* function is employed once again to handle collisions with rectangular obstacles by treating the sides of these rectangles as if they were additional links in the environment. For circular obstacles, a collision is identified if the projection of the circle's center onto the axis of a link falls between the link's endpoints, and the distance from this projection point to the circle's center is less than the radius of the circle. Finally, the detection process addresses potential collisions between different robots. This is managed through the *robot_robot_collision* function.

Robot-Robot Collision Function

```
def robot_robot_collision(self, robot1, robot2):
    for i in range(robot1.n_links):
        prev_pos = robot1.base_pos if i == 0 else robot1.pos[i-1]
        for j in range(robot2.n_links):
            prev_pos2 = robot2.base_pos if j == 0 else robot2.pos[j-1]
            if self.link_link_collision(robot1.pos[i], prev_pos,
            ↪  robot2.pos[j], prev_pos2):
                return True
    return False
```

This function is designed to check for collisions between all distinct pairs of links from two different planar manipulators. To accomplish this, it iterates through each link of both robots and applies the *link_link_collision* function to determine if any pair of links intersect. However, for links that are almost parallel, it was observed that collisions may occur between the simulation time steps, which the basic *link_link_collision* function might miss. Addressing this issue by reducing the time steps would lead to a significant increase in collision detection calculations, burdening the algorithms with excessive computational overhead. Instead, a more conservative approach has been chosen. This method involves detecting potential collisions by measuring the distance between a joint of one robot and a link of another. If this distance falls below a predefined threshold, a collision is presumed to occur.

Link-Link Collision Function

```
def link_link_collision(self, p1, q1, p2, q2):
    # Check if the line segments intersect
    def ccw(a, b, c):
        cross_product = (b[0] - a[0]) * (c[1] - a[1]) - (b[1] - a[1]) * (c[0]
        ↪  - a[0])
        return cross_product > 0
    return ccw(p1, q1, p2) != ccw(p1, q1, q2) and ccw(p2, q2, p1) != ccw(p2,
    ↪  q2, q1)
```

This function implements an algorithm to determine if two line segments (representing planar manipulator links) intersect. The helper function *ccw* computes a cross product to check the relative orientation of each triplet of points.