TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Semester Thesis in Informatics

Efficient Path Planning for Modular Robots

Dian Yuan

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Semester Thesis in Informatics

Efficient Path Planning for Modular Robots

Effiziente Pfadplannung für Modulare Roboter

Author:	Dian Yuan
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Matthias Mayer, M.Sc.
Submission Date:	April 27, 2023

I confirm that this Semester's thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese Semesterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, April 27, 2023 Dian Yuan

Acknowledgments

I am delighted to extend my heartfelt appreciation to my advisor, Matthias Mayer, for his unwavering support and guidance throughout my semester thesis. Without his invaluable insights and expertise, this work would not have been possible.

I would also like to express my sincere gratitude to Prof. Matthias Althoff for his diligent supervision. Moreover, I would like to extend my thanks to the CPS team for their support and collaboration in developing the mcs and timor toolbox, which significantly aided my research.

Last but not least, I am grateful for the support of my family and friends who provided encouragement and motivation throughout the research process.

Thank you all for your contributions to this work.

Abstract

When deploying modular reconfigurable robots, there can be millions of possible assemblies that meet specific task requirements. For tasks involving a given target pose for the end-effector, traditional path planning algorithms plan independently for each combination of modules. This approach fails to exploit the structural similarities of these assembled robots, resulting in inefficient path planning.

This thesis proposes two extensions to sampling-based path planning algorithms that leverage previous results. The first algorithm only tries to connect the previous path to the desired goal with the changed kinematics, while the second algorithm identifies candidates for the inverse kinematic solution of the new robot from the previously calculated solution path.

The effectiveness of the two algorithms, both independently and in combination, was evaluated through testing. The results demonstrate that by using these algorithms, the required path planning time can be reduced by 89% while maintaining a good quality of the path planning results. Additionally, this thesis explores the impact of varying robot similarity on the performance of the algorithms and identifies the optimal similarity range of application for each algorithm.

Contents

1 Introduction

While robots have revolutionized mass-manufacturing and are making inroads into the service sector, many industries have yet to benefit from their capabilities. One significant challenge to their widespread adoption is the complexity of their deployment, which often requires costly specialists to be involved in every new task that is automated [1]. This limitation has prevented many companies from fully embracing automation, as the costs and time required to develop and deploy a robotic system can be prohibitively high, especially for smaller enterprises. Thus, there is a need for more accessible and user-friendly robotic systems that can be easily deployed and adapted to a variety of tasks, without the need for specialized expertise [2].

Modular reconfigurable robots (MRR) are a type of robotic system composed of multiple individual modules that can be rearranged to form different robot shapes and configurations. This modularity offers several advantages, including adaptability, fault tolerance, and scalability, making modular robots suitable for a wide range of applications, such as exploration [3], surveillance [4], and assembly [5]. Compared with traditional industrial robots, modular robots make it possible to generate robot systems that adapt to many different tasks, without the need for re-evaluation or re-calibration[1].

1.1 Motivation

With such great flexibility, there are potentially billions of combinations of robot modules that can be assembled to meet the requirements for a given task [6]. Finding an optimal configuration among them is currently mostly done by human expertise or trial and error [2]. We would like to have software that automatically selects the optimal combination of robot modules for the task.

In this thesis, we'll focus on the task of planning the path of an end-effector of a robot to reach the specific goal pose. In order to find the optimal combination for a certain task, it requires automated path planning to evaluate whether the task is feasible for any particular robot module combinations. Traditional sampling-based path planner like Rapidly-exploring Random Trees (RRT) [7] always considers each attempt independently, while given the properties of modular robots, if only one module of the robot is replaced, the new robot should still have a high degree of structural similarity to the old one. Here comes an idea that making results from planning for one robot configuration be reused when planning with another (similar) robot, and Figure 1.1 demonstrates an example of how to reuse the previous roadmap. Based on this idea,

[1]`https://www.ce.cit.tum.de/air/research/modular-robots/`

we would like to further explore which and how the results of previous path-planning attempts can be reused. We evaluate the efficiency of the new algorithms with suitable criteria.

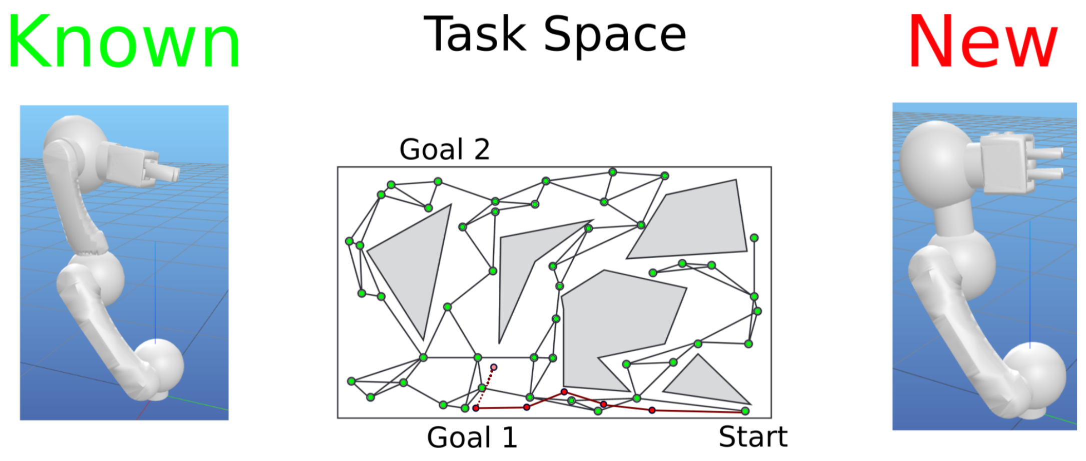


Figure 1.1: The previous roadmap planned with the left robot (green states and black connections) is repurposed for a new robot on the right with shorter arm. The new robot can get path to goal 1 without exploring the whole configuration space, as shown in red. To reach goal 2, the robot tries to follow the path previously explored by the other robot and extend new node as shown in lighter red with dotted connection. Image created by Matthias Mayer[2].

1.2 Task Description

The goal of the semester thesis is to improve the efficiency of the software solution for modular robots path planning problems. Specifically, we focus on planning problems that want the robot's end-effector to reach a set of goal poses (position and orientation in 3D space). In order to achieve this goal, the structure and main work of this thesis can be subdivided into the following parts:

- Chapter 2
  - Literature review on modular robots and state-of-the-art algorithms for robot path planning.
- Chapter 3
  - Use ModRob Configuration Synthesis (mcs) toolbox[3], Pinocchio [8], Toolbox for Industrial Modular Robotics (Timor) [9] and Open Motion Planning

[2]`https://www.ce.cit.tum.de/fileadmin/w00cgn/air/_my_direct_uploads/PathPlanning.pdf`
[3]`https://gitlab.lrz.de/cps-robotics/mcs`, insider access only

  Library (OMPL) [10] to create modular robots and implement basic path planning algorithms for them.
  - Store helpful data from path planning results and devise strategies to adapt previous planning results to similar modular robot assemblies.

- Chapter 4
  - Evaluate the effectiveness of algorithms reusing in terms of execution time and path quality.
  - Analyze the effect of the degree of similarity of different robots on the efficiency of the algorithms and summarize the scope of applicability for the different algorithms.
- Chapter 5
  - Discuss potential future directions for this work.
- Chapter 6
  - Provide concluding remarks.

2 Related Work

MRR, which consist of multiple interchangeable modules that can be assembled and disassembled to create different robot configurations, have gained increasing attention in recent years due to their flexibility, adaptability, and versatility [1, 11]. However, the use of MRR also presents unique challenges, including the area of path planning. Path planning for modular robots involves not only finding an optimal path that allows a single robot to accomplish a given task while avoiding obstacles but also how to solve the task efficiently when the robot is reconfigured.

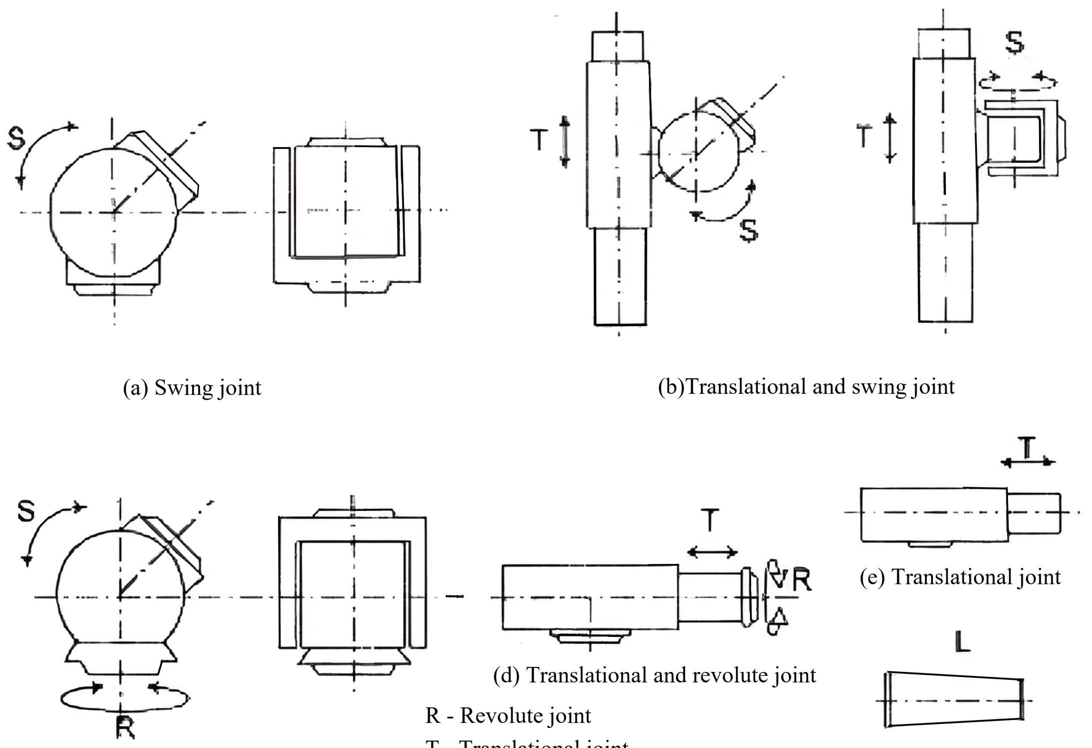
In this chapter, an overview of the state-of-the-art in modular robot design and development, robot path planning algorithms, and related libraries and platforms will be presented.

2.1 Modular Robots Development

Modularity has become a prominent concept in flexible machining systems, with increasing attention from research institutions in Europe and the United States. Modular robots have been studied since the late 1980s, with early research focused on module development, but more recent efforts targeting applications for these robots [12].

The subject of computer-aided design of robots has been of interest for a long time. In 1986, B.O. Nnaji published a monograph on *Computer-aided Design, Selection and Evaluation of Robots* [13]. He coded the range of motion and speed of the four possible joints of the robot and specified a total of 89 parameters (16 divisions) for actuators, joint drive units, joint control units, and design parameters, either qualitatively or quantitatively. K.-H. Wurst also gave general principles for module selection while developing modular robots [14]. His research focused on how to determine the topological relationships and structural parameters of the robot that meet the design requirements.

As the name implies, a modular robot consists of modules - i.e., modular joints and modular linkages. Modules should generally have standardized mechanical and electrical interfaces for inter-module connections. Module joints are driven by direct current (DC) or alternating current (AC) motors with integrated reduction mechanisms and controllers, while module links without degrees of freedom are used only for connections between joint modules. Usually, the modular links come in varying lengths and the standard interfaces have different orientations. These features enable the modular joints to connect in ways that satisfy the diverse kinematic and dynamic needs of the robot [12]. [14] argues that there are certain limitations to the composition of modular robots, i.e., finite joint modules and infinite linkage options. Figure 2.1 gives an illustration of the standard module developed by [14]. One-degree-of-freedom joints can be swing or translational, while two degrees of freedom (DoF) joints can be revolute with swing, translation with revolute, or translation with swing.



(a) Swing joint

(b)Translational and swing joint

(c) Swing and revolute joint

(d) Translational and revolute joint

(e) Translational joint

R - Revolute joint
T - Translational joint
S - Swing jount
L - Linkage

Figure 2.1: Standard modular robot modules suggested by [14]. Image based on [12, 14].

Theoretically, an infinite number of robots with different topological relationships can be constructed using the same type of standard module. However, from a practical point of view, a serial robot meeting the requirements of 6-DoF of space motion (the standard module in Figure 2.1 is limited to a serial robot) consists of no more than four multi-degree-of-freedom joint modules and three linkage modules [12]. If the end-effector itself has 3-DoF, the requirements for the manipulator's degrees of freedom are reduced. Typically, 6-DoF robots can meet the requirements of the most industrial applications,but some robots have higher DoF to improve obstacle avoidance performance or for additional dexerity, e.g. *Franka Emika Panda*[1], which has 7-DoF.

In recent years, researchers have gradually shifted their focus to how modular robots can be reconfigured for different tasks. The term reconfigurability pertains to a system's ability to be configured in various ways, regardless of the reconfiguration methods employed. Modular robots are typically categorized by their configurable capabilities

[1]`https://www.franka.de`

into two groups: manually configurable and self-reconfigurable (also known as auto-configurable) systems [1, 15]. Self-reconfigurable modular robots (SRMR) are capable of autonomously altering their configuration, whereas external reconfiguration is required for manually configurable modular robots. Each of these two design approaches has its own merits, and in recent years both options have been chosen almost identically [1]. In this thesis, we focus on manually configurable modular robots.

Moreover, people are not satisfied with just reconfiguration, they want the robot to be able to program itself after the reconfiguration, i.e., self-programming. Althoff et al. presented the concept of interconnectable modules for self-programming and self-verification (IMPROV) which contains three main highlights: self-programming, self-verification and optimal module composition. After conducting rigorous experiments, it was revealed that the IMPROV robots, despite their impressive reconfigurability, exhibited the same exceptional level of control performance as their non-modular counterparts [2]. Other studies have focused on finding the optimal configuration of modular robots for specific tasks, such as *A Composable Benchmark for Robotics Applications (CoBRA)* [16], a benchmark for evaluating the automatic selection and adaptation of robots to specific tasks. It is foreseeable that in the future, as artificial intelligence technologies such as reinforcement learning are further applied to modular robots [17], they could transform current automation practices.

2.2 Robot Path Planning

2.2.1 Inverse Kinematics Solution

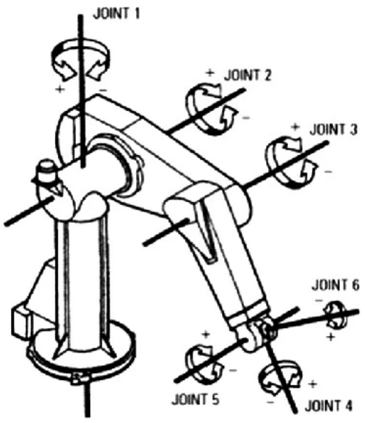
Before introducing the path planning algorithms, it is necessary to understand that most of the current path planning algorithms are based on the robot's configuration space. In practical applications, however, the goal is always to move the end-effector to a target position and orientation, and inverse kinematics (IK) is the problem of finding the robot joint configuration that achieves this goal. Unlike forward kinematics, robots with multiple revolute joints more than the number of DoF needed to fulfill specific task poses typically have multiple solutions for IK, and several methods have been proposed to address this issue. These methods can generally be classified into two categories: those that obtain an *analytical* solution and those that use *numerical* optimization.

In some, but not all cases, analytical solutions to inverse kinematic problems exist if the robot satisfies the *Pieper criterion* [18]. This criterion shows that a closed-form solution exists if a 6-DoF serial robot satisfies one of the following two sufficient conditions:

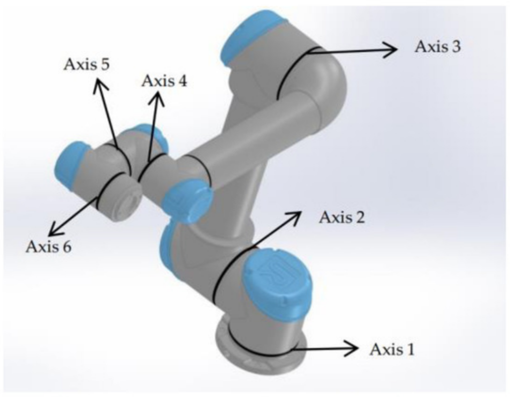
- *The three adjacent joint axes of the robot intersect at one point,* or
- *the three adjacent joint axes of the robot are parallel.*

Most commercial industrial robots today are designed to meet the *Pieper criterion* as much as possible when designing configurations because the analytical solution can be solved quickly with less arithmetic power and with a lower cost controller. Figure 2.2

shows two typical examples that satisfy two conditions respectively. However, for modular robots, it is difficult to satisfy any combination of modules to fulfill one of the criterion conditions, and sometimes there are redundant robots, in which case we need a numerical solution.



(a) PUMA560 6-axis robot arm. Image from [19].



(b) UR5 6-axis robot arm. Image from [20].

Figure 2.2: Examples of robots that satisfy the *Pieper criterion*: (a) its last 3 joint axes intersect at one point, which meets the first condition; (b) the second, third, and fourth joint axes are parallel, meets the second condition.

Due to the complexity of inverting the forward kinematics equation and the possibility of an empty solution space, most numerical methods for solving IK problems rely on iterative optimization to approximate a solution. Several of these methods are based on the core idea of using a Taylor series expansion to model the forward kinematics equation, as it can be easier to invert and iteratively solve than the original system.

Assume there is a mapping from the $n \in N^+$ dimensional joint space to Cartesian pose $\mathbf{F}(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^3 \times SO(3)$ and given a desired goal position $\mathbf{g} \in \mathbb{R}^3 \times SO(3)$, the optimization problem is to find the joint configuration $\boldsymbol{\theta}^*$ that minimizes the distance between the goal position and the end-effector position. The distance can be defined as the Euclidean distance between the two points. The optimization problem can be formulated as [21]

$$\theta_i^* = \arg \min_{l_i \le \theta_i \le u_i} \frac{1}{2} \|\mathbf{g}^{-1}\mathbf{F}(\boldsymbol{\theta}) - \mathbf{I}\|^2 \ (i = 1, 2, ..., n) \tag{2.1}$$

where $l_i < u_i$ are lower and upper bounds on the angel of joint i, and $\mathbf{I}$ is the identity matrix. The residual vector can be defined as $\mathbf{r}(\boldsymbol{\theta}) = \mathbf{g}^{-1}\mathbf{F}(\boldsymbol{\theta}) - \mathbf{I}$. Let $\mathbf{f}(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{r}(\boldsymbol{\theta})|^2$, solving the residual equation using an iterative Newton method implies solving the equation

$$\mathbf{J}\Delta\boldsymbol{\theta} = \mathbf{r}(\boldsymbol{\theta}) \tag{2.2}$$

where $\mathrm{J} = \frac{\partial \mathbf{F}(\theta)}{\partial \theta}$ is the Jacobian matrix. The gradient of the IK minimization formulation can be written as

$$\nabla \mathbf{f}(\boldsymbol{\theta}) = -\frac{\partial \mathbf{F}^T}{\partial \boldsymbol{\theta}} \mathbf{r}(\boldsymbol{\theta}) = -\mathbf{J}^T \mathbf{r}. \tag{2.3}$$

From (2.2) and (2.3) it shows an efficient method for computing Jacobian matrix and its inverse is desired. Common methods depending on the order of **J** include Jacobian transpose method [22], pseudoinverse method, damped least squares [23] and singular value decomposition [24].

Based on Newton's method it is also possible to use second derivatives information in Hessian matrix **H** by solving the equation

$$\mathbf{H}\mathbf{p} = -\nabla \mathbf{f}(\boldsymbol{\theta}) \tag{2.4}$$

where the descent direction **p** is used to update the joint configuration $\boldsymbol{\theta}$ with the step size α as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \mathbf{p}$. The (i, j) entry of the Hessian matrix is defined as

$$\mathbf{H}_{ij} = \frac{\partial^2 \mathbf{f}(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j} = \mathbf{J}_j^T \mathbf{J}_i - \mathbf{K}_{i,j}^T \mathbf{r} \tag{2.5}$$

where $\mathbf{K}_{i,j} = \frac{\partial \mathbf{J}_i}{\partial \theta_j}$ is a third order tensor, which usually can be ignored. Finally the problem leads to using iterative techniques to approximate Hessian matrix **H**, e.g. Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) [25] and its extension like L-BFGS [26], etc. In *Timor* an optimization algorithm provided by *sciPy* based on L-BFGS-B[2] is used to solve the IK problems for robots with less than 6-DoF, since Jacobian-based methods are not stable for small robots [27].

Heuristic methods like *cyclic coordinate descent (CCD)* [28] and *forward and backward reaching inverse kinematics (FABRIK)* [29] can also be employed to approximate the IK problem. Such methods use simple, iterative operations to progressively approach the solution. These algorithms have a low computational cost, enabling them to rapidly provide the final pose, while also supporting joint constraints.

2.2.2 Sampling-based Path Planning

Sampling-based algorithms are the most common solution to the robot path planning problem. Depending on the number of paths planned each time, they can be divided into multiple-query planners and single-query planners [30]. One of the most popular multiple-query planners is called Probabilistic Roadmaps (PRM) [31]. The idea is to take random samples from the configuration space, check that they are collision-free, and then attempt to connect these configurations to their nearby ones. It has been proven that PRM is probabilistically complete, which means that as the number of sampled points grows infinitely, the algorithm should always be able to find a solution as long as one exists. However, the rate of convergence depends on certain visibility properties of the free space and in some research scholars have modified the sampling and connection strategies of traditional PRM to improve the performance [32].

[2]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

RRT [33] is a basic single-query planner to efficiently search high-dimensional spaces by randomly building a space-filling tree. It works by incrementally building a tree of possible paths from the start position and randomly sampling new nodes to add to the tree until a node is added near the goal position or a predefined number of nodes have been added, the algorithm in pseudo code is shown in Algorithm 1.

Algorithm 1 Build RRT [33]

```
Require: Initial configuration $q_{init}$, number of vertices in RRT $N \in N^+$, incremental
    distance $\Delta q \in R^+$
Ensure: RRT graph $G$
1: $G.init(q_{init})$
2: for $i = 1$ to $N$ do
3:     $q_{rand} \leftarrow$ FreeSample()
4:     $q_{near} \leftarrow$ NearestVertex($q_{rand}, G$)
5:     $q_{new} \leftarrow$ NewConfiguration($q_{near}, q_{rand}, \Delta q$)
6:     $G.addVertex(q_{new})$
7:     $G.addEdge(q_{near}, q_{new})$
8: end for
9: return $G$
```

Based on this concept, scholars devised a strategy to incrementally construct two rapidly-exploring random trees (RRTs) with roots at the start and goal configurations. The path planning is considered successful when the two trees are connected. This led to the development of the RRT-Connect (RRTC) algorithm [7]. A sample point q_{rand} is obtained by sampling once, and then both search trees are expanded in the direction of sample point q_{rand} at the same time to speed up the establishment of connection between the two trees. Compared with the single-tree expansion RRT algorithm, RRTC adds a heuristic step to accelerate the search speed and has better results for narrow passages. However, like RRT, the RRTC algorithm is a single-query algorithm that only finds feasible paths as quickly as possible, and is not guaranteed to find the optimal solution.

To obtain the locally optimal solution, Karaman et al. proposed the RRT* algorithm [34], see Algorithm 2, which adds to RRT the selection strategy of searching the parent node, i.e., adding a rewiring process when adding q_{new} to the parent node, that is, finding the node with the smallest path cost (path length of moving from the starting point to q_{new}) after connecting with q_{new} in the neighborhood with q_{new} as the center and radius r, and rewiring the node with q_{min} as the parent node of q_{new} instead of q_{near}. The schematic diagram of the rewiring process is shown in Figure 2.3.

2.2.3 Path Planning for Reconfigurable Robots

The path planning algorithm mentioned in the previous section is basically applicable to a single robot performing a specific task. These approaches separately solve the same

Algorithm 2 Build RRT* [34]

```
Require: Initial configuration $q_{init}$, number of vertices in RRT $N \in N^+$, incremental distance $\Delta q \in R^+$, exploration constant $\gamma \in R^+$
Ensure: asymptotically optimal RRT* graph $G$
 1: $G.init(q_{init})$
 2: for $i = 1$ to $N$ do                                          ▷ Standard RRT
 3:     $q_{rand} \leftarrow$ FreeSample()
 4:     $q_{near} \leftarrow$ NearestVertex($q_{rand}, G$)
 5:     $q_{new} \leftarrow$ NewConfiguration($q_{near}, q_{rand}, \Delta q$)
 6:     if ObstacleFree($q_{near}, q_{new}$) then
 7:         $Q_{near} \leftarrow$ NearbyVertices($G, q_{new}, \gamma$)
 8:         $q_{min} \leftarrow q_{near}$
 9:         $c_{min} \leftarrow$ Cost($q_{near}$) + Distance($q_{near}, q_{new}$)
10:         for $q_{nearby} \in Q_{near}$ do                            ▷ Optimize $q_{min}$
11:             if ObstacleFree($q_{nearby}, q_{new}$) then
12:                 $c_{nearby} \leftarrow$ Cost($q_{nearby}$) + Distance($q_{nearby}, q_{new}$)
13:                 if $c_{nearby} < c_{min}$ then
14:                     $q_{min} \leftarrow q_{nearby}$
15:                     $c_{min} \leftarrow c_{nearby}$
16:                 end if
17:             end if
18:         end for
19:         $G.addVertex(q_{new})$
20:         $G.addEdge(q_{min}, q_{new})$
21:         for $q_{nearby} \in Q_{near}$ do          ▷ Rewire to shortest distance from start
22:             if ObstacleFree($q_{new}, q_{nearby}$) and Cost($q_{new}$) + Distance($q_{new}, q_{nearby}$) < Cost($q_{nearby}$) then
23:                 $G.removeEdge(q_{nearby},$ Parent($q_{nearby}$))
24:                 $G.addEdge(q_{new}, q_{nearby})$
25:             end if
26:         end for
27:     end if
28: end for
29: return $G$
```

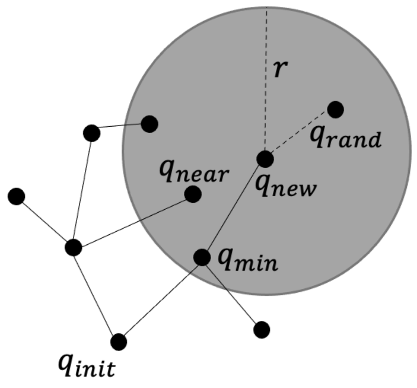


Figure 2.3: Schematic diagram of the rewiring process in RRT* [34]: 1. choose q_{new} to extend; 2. sample q_{rand} in sphere around q_{new}; 3. check if q_{rand} can be attached to the tree by comparing with q_{near}.

task using no prior knowledge of the task and environment (with slightly varying robots). These planning-from-scratch (PFS) approaches are not suitable for MRR because the robot's configuration is changing, which will take a lot of time to do similar calculations on complex problems [35]. Even if the robot configuration or the environment has only a little change, PFS affords no way to take advantage of the previous work.

To address this issue, [35] proposed a framework called *Lighting* that suggested an idea of making retrieve-and-repair path parallel to PFS. The idea of retrieve-and-repair can be explained as Figure 2.4. It retrieves the closest path (black) from the database containing the previous path planning results based on a heuristic algorithm, and repairs the infeasible path between violation with retrieved previous path (the sketched RRT connect trees in orange / blue). This framework has been tested on both PR2 and minimally-invasive surgery robot to significantly reduce the path planning time.

In a different direction, with the development of artificial intelligence in recent years, reinforcement learning techniques such as Q-learning [36] have also been applied to solve the similar problem.

2.2.4 Path Evaluation

In order to evaluate a path planner, commonly used evaluation methods include computation time, success rate, etc. Paulin et al. provide a comparison of sampling-based algorithms for a grape vine pruning robot arm, based on execution time, success rate, and number of samples [37]. On the other hand, other evaluation criterion focuses more on the generated trajectory itself, trying to quantify optimality.

Cartesian distance of tool center point (TCP) between each waypoint of the path is a trivial criterion often used. The metric, as defined in (2.6), where i is waypoint

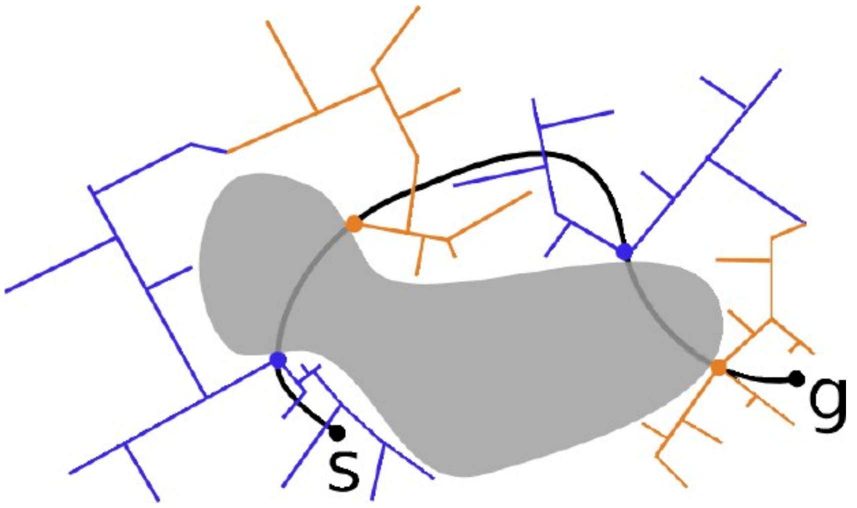


Figure 2.4: Principle of retrieve and repair path. Black: Retrieved path from the database. Gray: New obstacle in configuration space because of environment/robot changes. Blue: Forward-searching trees. Orange: Backward-searching trees. Figure from [35].

ID of n waypoint and p_i refers to the TCP position, involves minimizing the sum of the Euclidean distance between each consecutive TCP position in $\mathbb{R}^3$ [38]. Research has demonstrated the effectiveness of this criterion in optimizing robot programs for minimizing production cycle times in applications such as spot welding [39].

$$p_{dist} = \sum_{i=2}^{n} \|p_i - p_{i-1}\| \tag{2.6}$$

Similarly, in the joint space, we could define the joint distance criterion as an accumulated summary of the differences for each joint value as (2.7) [40]. The valuable $q_{i,j}$ refers to the j^{th} joint value of the i^{th} waypoint, and n and m are the numbers of waypoints and joints, respectively.

$$q_{dist} = \sum_{i=2}^{n} \sum_{j=1}^{m} \|q_{i,j} - q_{i-1,j}\| \tag{2.7}$$

In order to achieve optimization of selected joints, LaValle [30] proposed a modification on joint distance criterion (2.8) by multiplying each joint value a weight ω_j in the range between 0 and 1. In some articles, the criterion is also referred to as control pseudo-cost [38]. In certain applications, using this criterion may simplify energy consumption [41] even when the actual energy consumption is not known until after the trajectory has been executed [42].

$$q_{dist} = \sum_{i=2}^{n} \sum_{j=1}^{m} w_j \left(\|q_{i,j} - q_{i-1,j}\|\right) \tag{2.8}$$

Furthermore, taking robot dynamics into account, the criterion can be modified by introducing jerk, which is defined as the third time-derivation of the position. The joint jerk can be defined as a summary of jerks on each joint (2.9), alternatively, to find the maximal jerk between waypoints (2.10), in which Q represents all joint states of a path. The latter criterion is thought as a safety threshold [38].

$$q_{jerk} = \sum_{i=2}^{n} \sum_{j=1}^{m} \left\| \dddot{q}_{i,j} - \dddot{q}_{i-1,j} \right\| \tag{2.9}$$

$$q_{jerk} = \max_{q_i \in Q} \left\{ \sum_{j=1}^{m} \left\| \dddot{q}_{i,j} - \dddot{q}_{i-1,j} \right\| \right\} \tag{2.10}$$

2.3 Related Software Development

Over the last decade, several libraries and platforms have emerged in the field of robotics motion planning, all aimed at assisting in the resolution of the robot path planning predicament. This section endeavors to present a few of the prominent software in this regard.

1. **OMPL** is an open-source software library that has gained widespread use in the field of motion planning. The library is implemented in the C++ programming language and features an extensible and customizable design. It offers users a flexible interface that enables them to define their own problem domains and constraints, and incorporates various optimization techniques to enhance the efficiency of the planning process. Additionally, OMPL boasts support for the automatic generation of Python bindings, further extending its usability to developers utilizing Python-based applications.

   OMPL provides a set of tools and algorithms for motion planning, including sampling-based planning methods such as PRM, RRT and their variations. These algorithms can be used for a variety of applications, including robotics, autonomous vehicles, and animation [10].

2. **Flexible Collision Library (FCL)** [43] is an open-source software library designed for collision detection and proximity computation in robotics, virtual reality, and computer graphics applications. The library provides a suite of efficient algorithms for performing collision checks between geometric shapes, such as meshes, point clouds, and octrees. It is designed to be flexible, extensible, and easy to use, allowing developers to quickly integrate collision detection functionality into their applications.

3. **Pinocchio** [8] is a state-of-the-art Rigid-Body Algorithm for poly-articulated systems, with analytical derivatives of the main Rigid-Body Algorithms. It is designed primarily for robotics applications but can also be utilized in other contexts such as

biomechanics, computer graphics, and vision. Pinocchio utilizes *Eigen*[3] for linear algebra and *FCL* for collision detection, and comes with a Python interface for fast code prototyping[4].

4. **Timor** is an open-source Python library that simplifies the creation and simulation of modular robots, bridging the gap between conventional and modular robots. Its various features, including fitting robots into *Pinocchio*, configuration optimization, and model identification, enable researchers to perform complex simulations and generate precise models of modular robots. By contributing to the development of robotics as a field, Timor facilitates the study and advancement of modular robot technology [9].

5. **mcs toolbox** is currently an internal repository to collect and develop algorithms to synthesize modular robot configurations and possibly robot morphologies, maintained by Cyber-Physical Systems group at Technische Universität München (TUM). Different from Timor, mcs focuses more on the optimization of modular robot morphologies and implements high-level path planners.

6. **CoBRA** represents the pioneering benchmark suite for discovering optimal solutions for conventional and modular robots. Specifically tailored to industrial settings featuring familiar environments, the benchmark suite greatly simplifies task description and eliminates the need for perception. With its emphasis on the motion planning problem and its inherent degrees of freedom, CoBRA extends the scope of many tasks, including rotational symmetries of tools, execution tolerances, and base position flexibility. To ensure easy access and encourage collaboration, all benchmarks are available on the CoBRA website[5] for convenient sharing, referencing, and comparison of solutions [16].

[3]`https://eigen.tuxfamily.org`
[4]`https://github.com/stack-of-tasks/pinocchio`
[5]`https://cobra.cps.cit.tum.de`

3 Solution Approach

This chapter describes the approaches to achieving efficient path planning. First, the tools and methods used for development will be presented. Second, the overall framework and principles of the program will be introduced by planning the path for a single robot. Finally, two ideas and three algorithms are explained that reuse the previous path planning results.

3.1 Used Tools and Methods

The developed software is written in Python and is designed to work with *Python 3.9.7*. Development and testing were done using the Linux distribution *Ubuntu 20.04 LTS*. The path planners are inherited from the planners provided in the *mcs* toolbox, with *pinocchio 2.6.4*, *OMPL 1.5.2* and *Timor 0.0.5* used as dependencies. They are all described in Section 2.3.

Within OMPL the RRTC algorithm is used as the default algorithm to perform path planning for this thesis. The Jacobi-based iterative optimization algorithm will be used to solve the IK problem. We record the time taken by the planner to find the path to measure the efficiency and evaluate the quality of the generated paths with the metric q_{dist} defined as (2.7).

3.2 Single-Task Approach

For the single-task problem, we define it as planning a path for a particular robot such that its end-effector arrives at a specified pose in $SE(3)$ from the initial joint configuration. We propose a framework to solve this problem as presented in Figure 3.1, and each stage of the process is described below.

3.2.1 Set Up

In the stage of setup, we need to construct the robot, design the environment, add obstacles, set up tasks, select a path planner, and configure parameters. Many thanks to the developers of *Timor* and *mcs* for making all this easy by providing a rich interface and resources.

In order to represent the pose of the target in space, a 4×4 homogeneous transformation matrix-shaped as in (3.1) is used, where $R_{i,j}$ are the elements of the 3×3 rotation matrix that represents the orientation of the end-effector, and T_x, T_y, and T_z are the

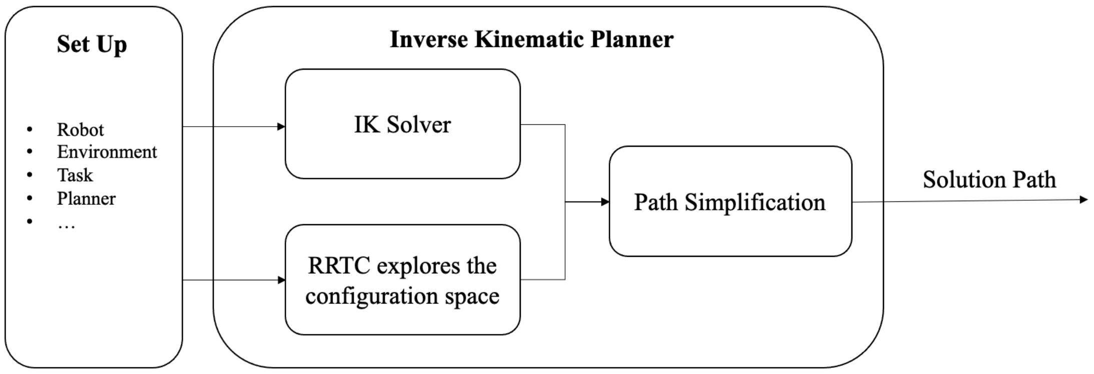


Figure 3.1: A framework for single-task solving.

translation parameters that represent the position of the end-effector in the space. The last row is always $[0, 0, 0, 1]$ to maintain homogeneity.

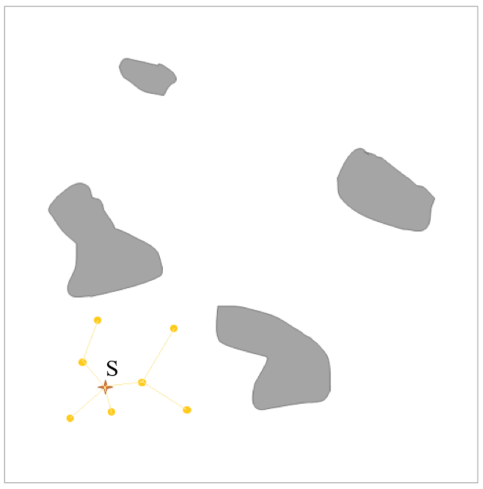
$$\mathbf{T} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

3.2.2 Path Planning

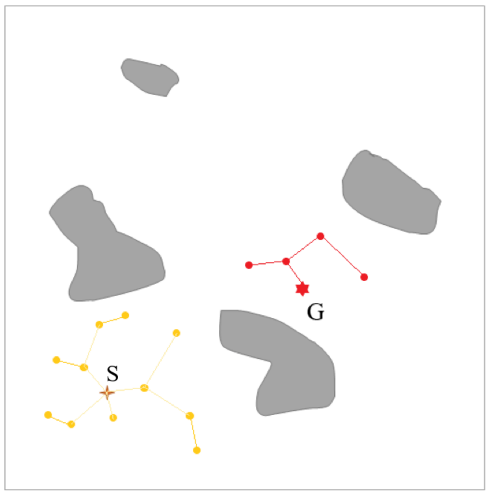
Finding a connected path is the main task of the planner. There are two main jobs, the first is to find a solution in the joint space from solving the IK problem, and the second is to explore the environment, and find a path that connects the initial configuration to the target configuration solution using the RRTC algorithm, the two tasks are designed to be performed in parallel.

To solve the IK problem, Timor provides a Jacobian-based iterative approach as in (2.3). It used the damped least squares method [24] to calculate the inverse of Jacobian and iterate from the current configuration as the initial value as default. For robots with DoF greater than 6, there may be an infinite number of IK solutions, and the IK solver will keep searching for the solution that satisfies the error tolerance as the starting point of the RRTC backward search tree until the path is found.

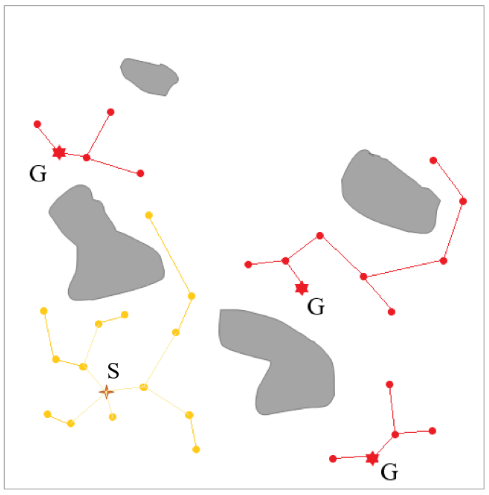
The RRTC as described in Section 2.2.2 is implemented by OMPL and we'll use the Python bindings it provides. When the planner starts running, it immediately builds a tree to explore the (collision-free) configuration space starting from the initial configuration, and when the IK solver gets a solution, it also builds a tree starting from each solution until it is connected to the tree starting from the initial point. Figure 3.2 illustrates the generation of trees in the joint configuration space of a simple 2D robot at different stages of path planning.



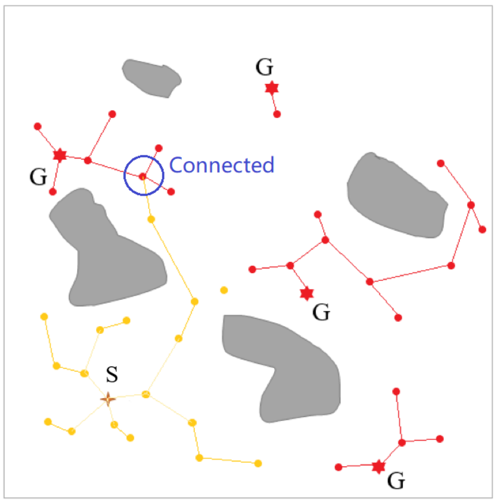
(a) Once the planner starts, RRTC randomly expands the map starting from the initial configuration.



(b) After the IK solver finds a solution, the RRTC similarly expands the tree backward from the goal configuration.



(c) As more IK solutions are found, an increasing number of backward search trees are generated.



(d) When any of the forward and backward search trees are concatenated, the path is found, and planning ends.

Figure 3.2: Situation in joint configuration space at different stages during the planning period. S: start configuration. G: IK solution. Yellow: forward search tree. Red: backward search tree.

3.2.3 Path Simplification

The RRTC algorithm generates trees of potential paths by randomly sampling the configuration space and connecting the sampled points to the nearest points in the tree. While this process generates a feasible path, the resulting path may have many unnecessary, redundant, or convoluted segments that could be removed to improve the path's quality. Path simplification involves iteratively removing unnecessary segments of the path while maintaining the path's validity. In our planner, the default path simplification method *shortcut*[1] provided by OMPL is chosen, which involves iteratively removing intermediate points from the path while checking for collisions with obstacles. The process continues until no further simplification is possible or reaches the set time limit. Figure 3.3 is a simplified schematic representation of the solution path Figure 3.2d.

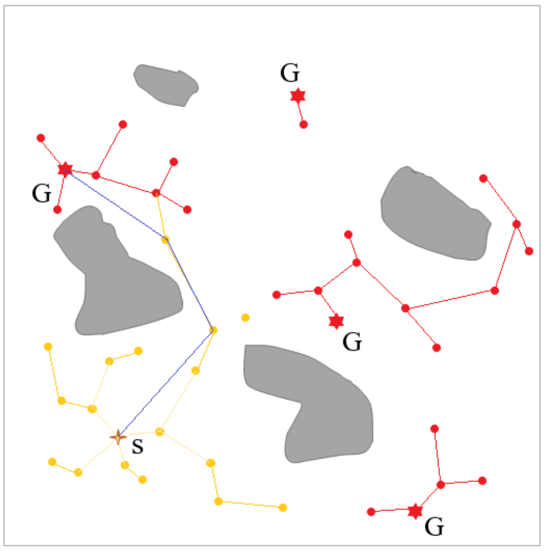


Figure 3.3: Path simplification schematic: the blue line is a simplified path based on the original red and yellow paths, with unnecessary intermediate nodes removed.

3.3 Reuse Path Approach

From now on, we assume that the modular robot has undergone minimal structural alterations, such as the replacement of a specific linkage with another model. Nonetheless, the environmental conditions, initial configuration, and target pose remain unchanged. At this juncture, the objective closely aligns with the initial robotic task, thus warranting a justifiable assumption that the outcomes of both tasks exhibit a high degree of resemblance. We note that indeed the collision map in joint space will have changed

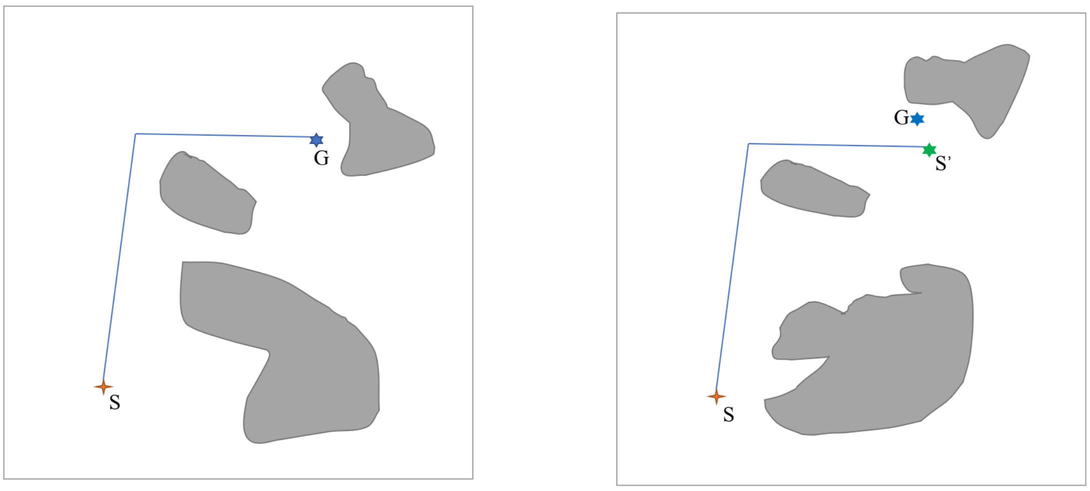
[1] `https://ompl.kavrakilab.org/classompl_1_1geometric_1_1PathSimplifier.html`

with the different kinematics but these changes are proportional to the alterations of the robot [44]. Consequently, our aim is to leverage the valuable insights gained from the preceding path results to expedite planning for the next robot. We shall approach this matter from two perspectives, namely, from the configuration space and the IK solution.

3.3.1 Reuse Path

Drawing inspiration from [35], we sought to repair the previous path. Given the close resemblance between the two robots, we opted to utilize the complete path obtained from the prior result directly. This strategy enabled the end-effector to remain close to the target pose, which can be readily verified using forward kinematics.

Assuming that the robot does not encounter self-collisions or collisions with its surroundings subsequent to utilizing the prior path directly[2], the planning problem transforms into a problem of devising a path from the vicinity of the target pose to the exact target pose, as shown in figure 3.4. At this point, the distance[3] between the start and goal points is very close, and therefore, a connected tree can be built more quickly.



(a) Original robot path planning results in configuration space.

(b) Use the previous path for current configuration.

Figure 3.4: Schematic of reuse path: (b) is the case after using the path of (a), the end-effector can only reach the point S' (same point as G in (a)) due to the change in configuration space, and the task of path planning is reduced to connecting S' to G.

Note that at this point we can first join the two paths together and then simplify the

[2]The algorithm does conduct collision detection and refrains from employing this technique if collisions exist. In Section 4.2 we will demonstrate experimentally that assuming the absence of collisions is a feasible assumption.

[3]More precisely, it should be the difference in position and orientation.

merged path as described in Section 3.2.3.

3.3.2 Feed IK Solution Candidates

The majority of Jacobi-based IK solutions highlight the significance of iterative initial value selection [24]. Typically, the Jacobi method can converge swiftly and precisely if the initial values are in proximity to the actual solution. However, if the initial values are considerably distant from the solution, the Jacobi method may either converge slowly or fail to converge entirely. Given that employing the previous path approach permits the robot's end-effector to approximate the vicinity of the target pose, the prior solution of the IK serves as a suitable initial value for the optimization problem. Consequently, we devised a strategy that involves feeding the waypoints from the preceding result paths to the IK solver in reverse order as prospective solutions (initial values) for the new IK problem. We expect this technique facilitates expedited identification of solutions.

The algorithms introduced in Section 3.3.1 and Section 3.3.2 can be implemented either independently or in conjunction with each other. For the convenience of later expression, we have named these algorithms with the abbreviations:

- **Original method (OR)**
- **Reuse-Path method (RP)**
- **feed Inverse Kinematics Candidates method (IKC)**
- **Reuse Path and feed Inverse Kinematics Candidates method (RP-IKC)**

In the forthcoming chapter, we will assess the effectiveness of these algorithms utilizing these performance metrics via targeted experiments.

4 Performance Evaluation

This chapter presents a comprehensive evaluation of the three algorithms proposed in the previous sections. To assess their effectiveness, we designed practical cases that test the algorithms' performance in terms of running time and path quality. Specifically, we considered robots with varying degrees of similarity to determine how the algorithms perform under different conditions. By evaluating the algorithms in a realistic task, we can gain insight into their practical applicability and identify any limitations that may arise.

4.1 Test Task Description

4.1.1 Modular Robot Selection

We selected *ModRob Gen2*[1] as our test robot. It is a modular robot prototype designed jointly by Chair of Cyber-Physical System at TUM and RobCo GmbH[2], which consists of various modules that can be flexibly combined to form a diverse range of robot configurations.

In the current study, several modules were selected to create a 6-DoF robot, as shown in Figure 4.1. It is important to note that the selection of modules was not based on any specific criteria and was arbitrary to simulate a range of possible configurations encountered when testing modular robots combination. Thus, the structure of the robot used in this study may differ from that of a typical industrial robotic arm.

Starting from the base to the end-effector, the modules with IDs *105*, *2*, *2*, *24*, *2*, *25*, *1*, *1*, *6*, *1* and *GEP2010IL* were selected in order to form the robot. *I*-shape linkages with IDs *5*, *23*, *7*, *9*, *11* and *L*-shape linkage with ID *13* were chosen to serve as substitutions for the last linkage *6* in the original robot to construct robots with different degrees of similarity. Table 4.1 and Table 4.2 present the properties of the modules used by the original robot and the linkages for substituting, respectively.

By categorizing these replacement linkages into two distinct groups according to their degree of similarity, we can identify the initial three linkages as being more akin to the original linkage, owing to their same shape and comparable length (no more than 50% difference in length from the original linkage). Conversely, the latter three linkages are classified as less similar, as they exhibit either substantial dissimilarities in length (greater than 50% difference in length from the original linkage) or differences in shape from the original linkage.

[1]`https://gitlab.lrz.de/cps-robotics/robot-data/modrob-gen2`, insider access only
[2]`https://www.robco.de`

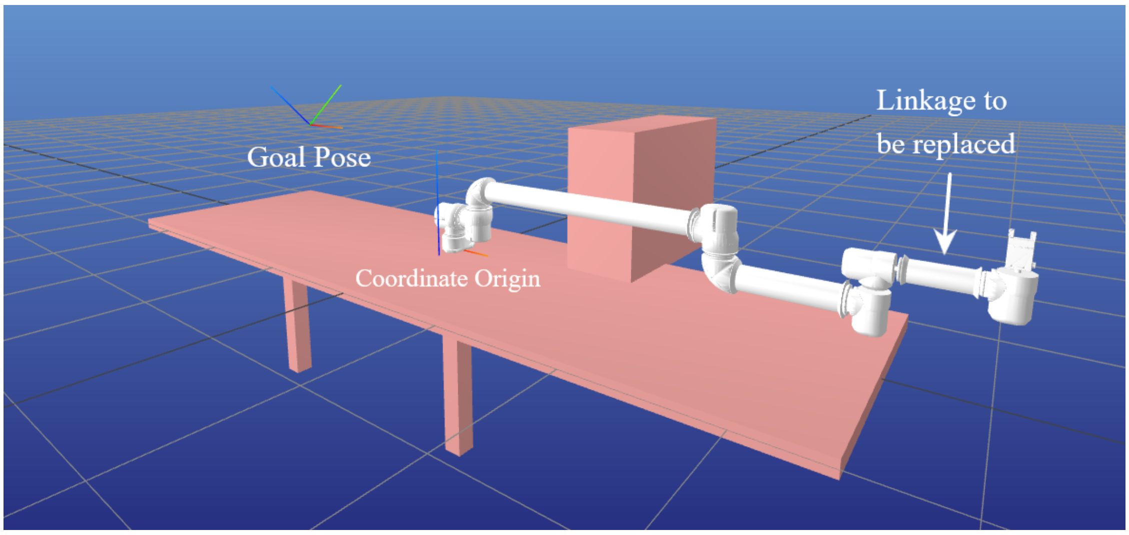


Figure 4.1: Structure of the original robot including the specified modules and the task goal pose.

Table 4.1: Used *ModRob Gen2* Modules Description.

ID	Type	Size	Length	Shape
105	base	110mm	-	-
2	joint	110mm	-	L
24	linkage	110mm	1100mm	L
25	linkage	110mm - 80mm	400mm	L
1	joint	80mm	-	L
6*	linkage	80mm	200mm	I
GEP2010IL	end-effector	80mm	-	-

* *Module to be substituted.*

4.1.2 Path Planning Task

We devised a straightforward experimental setup to simulate a robotic task requiring a precise pose in space for grasping an artifact. To mimic obstacles present in a typical environment, we incorporated a table with a box on it into the experimental environment. The starting configuration of the robot is a state where the displacement of each joint is 0. The base pose and the goal pose, described in the form of (3.1), are set as (4.1) and

Table 4.2: Used Linkage Substitutions Description.

ID	Type	Size	Length	Relative length*	Shape	Similarity
5	linkage	80mm	150mm	0.75	I	high
23	linkage	80mm	250mm	1.25	I	high
7	linkage	80mm	300mm	1.5	I	high
9	linkage	80mm	500mm	2.5	I	low
11	linkage	80mm	700mm	3.5	I	low
13	linkage	80mm	200mm	1.0	**L**	low

* *current linkage length/original linkage length*

(4.2), respectively.

$$T_{base} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.1 \\ 0.0 & 0.0 & 1.0 & 0.035 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{4.1}$$

$$T_{goal} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & -1.4 \\ 0.0 & 1/\sqrt{2} & -1/\sqrt{2} & 0.3 \\ 0.0 & 1/\sqrt{2} & 1/\sqrt{2} & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{4.2}$$

For position, the error tolerance is set to 0.001 in each dimension, and for orientation the tolerance is set to a maximum of 0.5° deviation for arbitrary axis[3] in space. To evaluate the distance between a pose $\mathbf{T}$ and a desired pose $\mathbf{T_d}$, we adopt a notation suggested by [16]. In this notation, we use a mapping $S : SE(3) \mapsto \mathbb{R}^N, N \in \mathbb{N}^+$ to constrain and assess the distances. The difference between the two poses after the mapping can be expressed as $\Delta_S(\mathbf{T}, \mathbf{T}_d) = S\left(\mathbf{T}_d^{-1}\mathbf{T}\right)$. The default mapping function $S(\mathbf{T}) = [x(\mathbf{T}), y(\mathbf{T}), z(\mathbf{T}), \theta_R(\mathbf{T})]^T$ includes the three Cartesian coordinates and the rotation angle about an axis for any given pose $\mathbf{T}$. The environment and goal pose are shown in Figure 4.1.

4.1.3 Test Environment

The software version at the time of testing is the same as the development environment described in Section 3.1. All performance tests were conducted in a controlled environment to ensure the accuracy and validity of the results. The program was executed on *Ubuntu 20.04.5 LTS* operating system installed in *Windows Subsystem for Linux (WSL)*

[3]Not limited to coordinate axes.

1.1.3.0, with a *Linux version 5.15.90.1*. The testing machine was equipped with 16GB dual-channel DDR4 3200MHZ memory and a *Ryzen R5 5600U* processor that has 6 cores and 12 threads, with a dynamic acceleration frequency of up to 4.2GHz[4]. To eliminate any external variables that may affect the test results, no other software was running in the background except for the programs being tested and windows internals were out of our control.

4.1.4 Test Process

The complete test process is as follows:

1. Construction of a robot for each substitution linkage in Table 4.1.
2. Perform the OR for the original robot to obtain the path planning result for the task described in Section 4.1.2.
3. Planning for the same task with the altered robot using the OR, IKC, RP, and RP-IKC.
4. Record the time taken for each planning task and the q_{dist} of the trajectory using (2.7).
5. For each new robot, perform step 2 to step 4 200 times.
6. Following the completion of each series of tests, make a 20-minute interval for the machine to cool down to its standard operating temperature prior to initiating the subsequent group of tests.

The total time for each planning task, including path finding and path simplification, is limited to 30 seconds. To further explore the probability that RP does not meet collisions, we recorded separate statistics on the number of collisions that would have occurred if the previous path had been used directly.

4.2 Results Evaluation

Prior to examining the specific outcomes, it is pertinent to address a key concern regarding the likelihood of collisions when employing the previous path directly. Through conducting the aforementioned evaluation, a total of 112 collisions were observed out of total 1200 tasks. Consequently, it can be inferred that in the present testing environment, the probability of encountering a collision when utilizing the previous path directly is less than 10%. As such, it is justifiable to initially assume the absence of collisions and subsequently abstain from using this algorithm if a collision is encountered, given that

[4]Technically the virtualization and core boost can have a significant effect on the test results especially, for long calculations. In the future, we expect to perform tests on native Ubuntu at a fixed clock frequency.

efficient collision checking done by OMPL might be hard to apply to a splicing path and we can always go back to plan from scratch if a collision exists.

Table 4.3 presents the overall results of the test, which will be analyzed from varying perspectives in the following subsections. Within the table, the metric "avg. planning time" refers to the average time (unit: second, same below) taken for the path planning task, whereas "avg. whole time" refers to the average time taken for the entire process, including path finding and path simplification. The percentages in parentheses indicate the degree of change relative to the corresponding metric of the OR algorithm. The best-performing data for each metric is shown in bold.

Table 4.3: Overview of Test Results.

ID	Metrics	OR	IKC	RP	RP-IKC
5	Avg. planning time (s)	0.52	0.31 (-40%)	0.38 (-27%)	**0.09 (-83%)**
5	Avg. whole time (s)	1.17	0.91 (-22%)	0.83 (-29%)	**0.33 (-72%)**
5	Avg. q_{dist} (rad)	13.12	**12.27 (-6%)**	12.89 (-2%)	12.64 (-4%)
23	Avg. planning time (s)	0.33	0.28 (-15%)	0.23 (-30%)	**0.07 (-79%)**
23	Avg. whole time (s)	0.85	0.88 (+4%)	0.68 (-20%)	**0.29 (-66%)**
23	Avg. q_{dist} (rad)	13.00	**12.42** (-4%)	12.86 (-1%)	12.81 (-1%)
7	Avg. planning time (s)	0.76	0.31 (-59%)	0.46 (-39%)	**0.08 (-89%)**
7	Avg. whole time (s)	1.38	0.92 (-33%)	0.87 (-37%)	**0.31 (-78%)**
7	Avg. q_{dist} (rad)	13.36	**12.39 (-7%)**	12.73 (-5%)	12.76 (-4%)
9	Avg. planning time (s)	0.68	0.31 (-54%)	0.52 (-24%)	**0.11 (-84%)**
9	Avg. whole time (s)	1.33	0.94 (-29%)	1.09 (-18%)	**0.35 (-74%)**
9	Avg. q_{dist} (rad)	**12.2**	13.1 (+7%)	12.44 (+2%)	12.54 (+3%)
11	Avg. planning time (s)	0.69	0.35 (-49%)	0.39 (-43%)	**0.12 (-83%)**
11	Avg. whole time time (s)	1.42	1.01 (-29%)	0.87 (-39%)	**0.40 (-72%)**
11	Avg. q_{dist} (rad)	16.14	13.77 (-15%)	13.11 (-19%)	**12.94 (-20%)**
13	Avg. planning time (s)	2.07	0.36 (-83%)	1.88 (-9%)	**0.22 (-89%)**
13	Avg. whole time time (s)	2.95	1.17 (-60%)	2.47 (-16%)	**1.02 (-65%)**
13	Avg. q_{dist} (rad)	16.65	15.01 (-10%)	12.98 (-22%)	**12.89 (-23%)**

4.2.1 Running Time

Since the focus of our algorithm is to speed up finding connected paths, we are more concerned with the planning time than the total time. From Table 4.3 we see that almost all algorithms achieve a speedup in average time for path planning, with RP-IKC always having the best performance. The speedup of it is at least 79% compared to the original algorithm. Table 4.4 counts the number and probability of each algorithm's planning time outperforming OR in each group of 200 tests. In general, when RP and IKC are used individually, they have their own strengths for different robot configurations, but when they are used at the same time, it yields exceptional and consistent performance.

Table 4.4: Count of Planning Time Better than OR for Each Algorithm.

ID	**IKC**	**RP**	**RP-IKC**
5	152 (76%)	132 (66%)	**187 (94%)**
23	111 (56%)	116 (58%)	**188 (94%)**
7	174 (87%)	152 (76%)	**197 (99%)**
9	120 (60%)	103 (52%)	**187 (94%)**
11	151 (76%)	156 (78%)	**193 (97%)**
13	**200 (100%)**	175 (88%)	**200 (100%)**

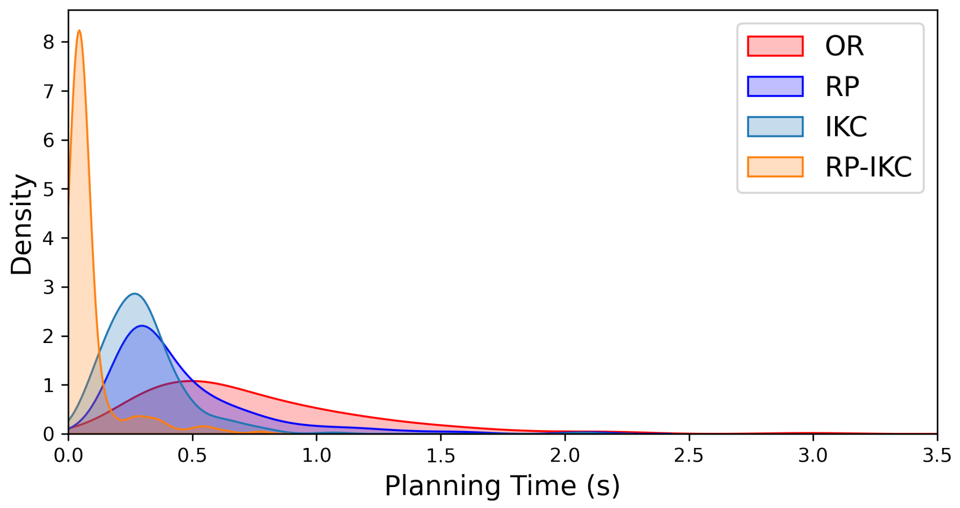
In order to conduct a more detailed analysis of algorithmic performance, we opted to examine two distinct data sets, specifically those identified as possessing the greatest efficacy (ID = 7, 13). In order to generate a visual representation of planning time distributions for the various algorithms employed within each loop, we employed kernel density estimation techniques[5], shown in Figure 4.2.

Through an examination of this figure, the benefits of utilizing RP-IKC become readily apparent, including its ability to perform quickly and consistently. To verify the stability from a statistical point of view, we chose the coefficient of variation (CV) (4.3) as an indicator, given that the mean values of each data set are different. A low CV usually indicates the data points are clustered closely around the mean value. In (4.3) σ denotes the standard deviation of the data and μ denotes the mean. The results of the CV of the planning for all groups of tests are shown in Table 4.5.

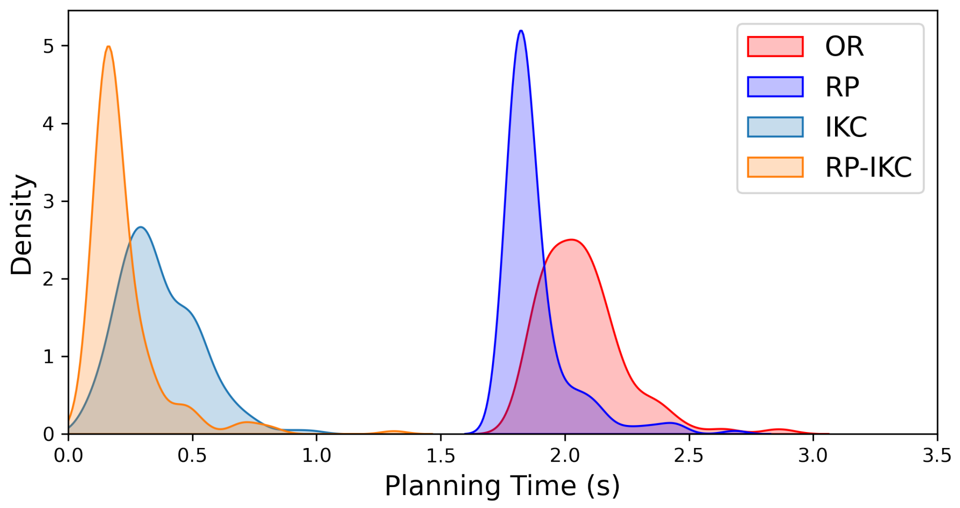
$$CV = \frac{\sigma}{\mu} \tag{4.3}$$

The data shows that the planning time of RP-IKC has the lowest CV, which verifies our observation that RP-IKC's planning time possesses less volatility. While RP and IKC do not show advantages in ths regrad.

[5]`https://seaborn.pydata.org/generated/seaborn.kdeplot.html`



(a)



(b)

Figure 4.2: Planning time density plot for linkage substitutions ID *7* (a) and ID *13* (b).

Table 4.5: CV of Planning Time for Each Algorithm.

ID	**OR**	**IKC**	**RP**	**RP-IKC**
5	1.61	1.52	1.86	**0.71**
23	1.44	1.80	2.04	**0.65**
7	1.60	1.55	1.46	**0.72**
9	0.79	0.77	2.03	**0.43**
11	1.20	1.02	1.16	**0.59**
13	12.00	13.45	2.29	**1.47**

Upon examining Figure 4.2b, a noteworthy observation is that the duration of solution times for OR and RP exhibit a remarkable similarity, while the duration for IKC and RP-IKC are almost same. Noting that linkage 13 has an L-shaped configuration, it can be posited that the utilization of RP allows the joints of the linkage to attain a position akin to that of the original robot. However, due to the alteration in the shape, it moves in a distinctly different direction, resulting in the end-effector failing to reach the vicinity of the intended goal area. Henceforth, it can be inferred that the RP algorithm encounters difficulty in leveraging previous results if there is a modification in the shape of the linkage.

4.2.2 Path Quality

The reuse of a prior path results in a reduction of the distance between the start and endpoint of the new plan. However, a critical concern arises regarding the optimality of the previous path for the new robot configuration and the potential impact of reusing it on the overall quality of the path, since using previous path may allow the robot to take some unnecessary path and thus have a large q_{dist} (2.7). We argue here that for the same robot, a solution path having a smaller q_{dist} implies a higher path quality.

From Table 4.3, we are somewhat surprised to find that in most cases, the paths obtained using the reuse algorithms have even smaller average q_{dist}. Given that we did not attempt to optimize the quality of the paths specifically and that the difference in the results is small, we need to explore whether this is due to the specific robot structure and goal.

We designed a set of experiments that reverse the previous test, in which we first constructed the original robot using the connecting modules with IDs *7*, *9*, and *11*, respectively, and then replaced with the linkage with ID *6* (in previous tests was part of the original robot, see Table 4.1) afterward to test the reuse algorithms. Similar procedures as described in Section 4.1.4 were conducted, at which time we focus on the q_{dist}, and the results are shown in Table 4.6. Compared with Table 4.3, it can be seen that the relationship between RP, RP-IKC and OR's q_{dist} is exactly the opposite of the previous results. This is because these two methods directly use the previous path, while for a given goal, the solution path quality(distance) is highly dependent on the structure of the robot. That's why q_{dist} improves or worsens with varying robot structures. Furthermore, if we calculate the standard deviation of these data, for example, for ID 7, the standard deviation of the four algorithms are $\sigma_{OR} = 2.6, \sigma_{RP} = 1.6, \sigma_{IKC} = 1.7, \sigma_{RP-IKC} = 1.7$, from which it is difficult to determine whether the differences in q_{dist} originate from noise. On the other hand, IKC is not affected too much since it doesn't use the previous path directly but probably allows more resources to be spent on optimizing the path.

Table 4.6: Average q_{dist} of the Path for the Reverse Test.

Original Linkage ID	**OR**	**IKC**	**RP**	**RP-IKC**
7	12.74	**11.96 (-6%)**	13.00 (+2%)	12.96 (+2%)
9	12.73	**10.61 (-16%)**	12.52 (-2%)	12.44 (-2%)
11	**12.79**	13.15 (+3%)	15.66 (+22%)	15.88 (+24%)

Nevertheless, the path quality gets better or worse within a certain range of randomness, and overall, we can say that the reuse algorithms do not significantly affect the path quality, which eliminates the concerns raised at the beginning of this subsection.

4.2.3 Summary of Results

Through the above evaluation, we found that the RP-IKC algorithm almost always has the best performance. As the RP and IKC optimization components of the algorithm run in parallel, it is gratifying to note that their optimization effects exhibit a surprising overlap. And we also confirm that using these reuse algorithms does not affect the quality of the obtained paths. The evaluation of these three algorithms is conducted based on the type of linkage substitution, and the results are tabulated in Table 4.7. A greater number of "+" symbols denote increased efficiency in comparison to the original algorithm.

Table 4.7: Overall Algorithms Comparison.

Linkage Change Type	**IKC**	**RP**	**RP-IKC**
Only length changes	++	+	++++
With shape changes	+++	-	++++

Regarding the modular robot path planning problem that involves substituting a single linkage for the same task, the RP-IKC algorithm has been shown to be the most efficient in test tasks similar to the one used in this chapter. We previously defined similarity based on length variation and shape change, but experimental results show that variation in length did not significantly affect the efficiency(ratio of reduced planning time) of the algorithm. Conversely, modification in linkage shape causes the RP algorithm to lose its effectiveness and approach the OR. Therefore, the efficiency of these algorithms needs to be re-evaluated for path planning problems with a greater degree of difference in the robot, such as replacing more than one module, or replacing modules closer to the base.

4.3 Profiling Analysis

To explore additional opportunities for optimizing the extant algorithm, we conducted performance analysis during the algorithm testing by employing a profiling tool. Specifically, we utilized *cProfile*[6] to record runtime data in a distinct series of 300 tests. It generated a text file that recorded the number of times each function was called and the time spent within that function, providing a comprehensive overview of the algorithm's performance at a granular level.

To have a more intuitive feel for this data, we utilized the visualization tool *SnakeViz*[7]. It facilitates the display of time spent on each function at runtime in a chart or table by level, as shown in Figure 4.3.

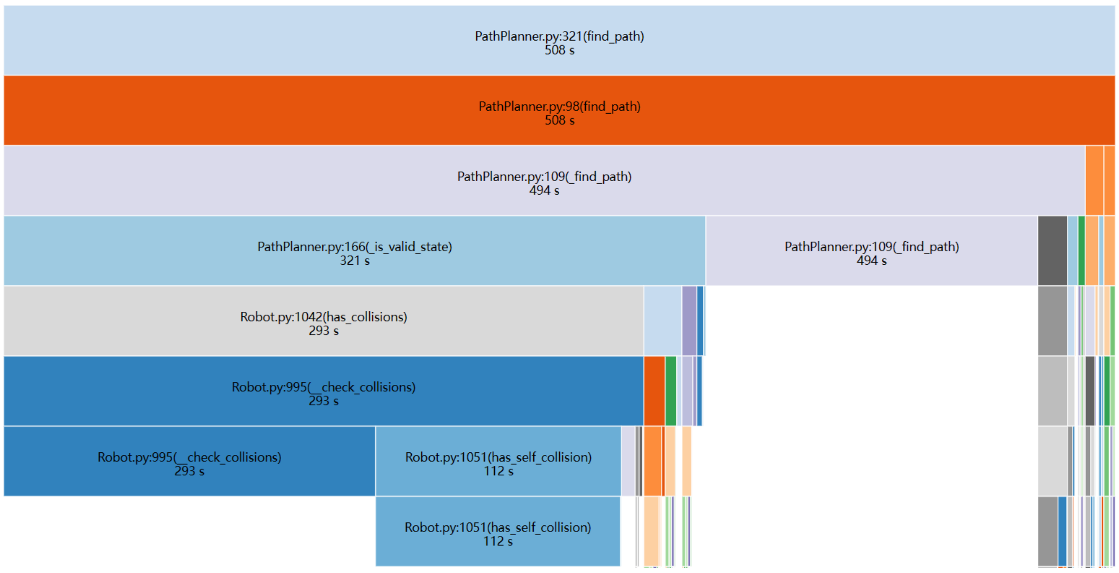


Figure 4.3: Run-time percentages for functions needed in path planning.

As we can see from the figure, more than 60% of the planning time is spent on the validity of the current configuration, the most important of which is the presence of

[6]https://docs.python.org/3/library/profile.html#module-cProfile
[7]https://jiffyclub.github.io/snakeviz/

collisions, including the detection of collisions with the environment and self-collisions. Further examination of the function calls shows that "`_is_valid_state`" was called 360,914 times in all 300 planning task tests, an average of over 1200 calls per planning task. This finding also provides a direction for further optimization of the algorithm, which could involve the implementation of a more efficient sampling algorithm to minimize the number of function calls, or optimizing the function itself to improve its computational efficiency.

5 Future Work

The experimental results presented in the preceding chapter validate the efficacy of the algorithms proposed in this thesis, in terms of reducing planning time without compromising the path quality. However, the current tests are limited to a single linkage replacement in the modular robot and only enable immediate utilization of the previously planned results.

To accomplish the ultimate objective of automated selection of the most suitable previous path, and thereby facilitate efficient path planning while exploiting the modularity of the robot, future work can concentrate on the following aspects.

Create Database

To facilitate efficient path planning, it is necessary to store each planning outcome in a suitable data structure that includes relevant information such as the path, module combination, task, and cost. For this purpose, the creation of a database is desirable, wherein the most suitable previous paths can be automatically retrieved at the start of each planning task, and the resulting paths can be stored in the database upon completion of planning.

Guidelines for Path Retrieval

In order to facilitate the automatic selection of the most suitable historical data from the database, it is necessary to develop a criterion for evaluating the similarity. This criterion should not be limited to the structure of the robot itself, but should also consider other factors such as the task and environment. The evaluation of these similarities and the determination of their relative weights are important questions that require further investigation.

Repair Failed Paths

Although the experiments demonstrated a 90% probability of collision-freeness for the new robot when using the previous path directly, this result was based on the fact that only one of the robot's original linkages was replaced. To broaden the range of applications for the algorithm, it is necessary to consider the case where collisions may occur and explore potential solutions, such as the retrieve-repair framework proposed in [35], which involves recalculating the path from the collision point onwards to rectify the issue. Additionally, running traditional (OR) and reuse algorithms in parallel, as proposed in this paper, is a promising approach as reuse algorithms don't always yield improved results.

Implemented with Optimal Planner

The presented implementation of the proposed algorithms is based on the RRTC algorithm [7] which is not optimal, and thus the path optimality is not guaranteed. Therefore, it is desirable to implement the proposed algorithms using an optimal planner, such as the RRT* [34] algorithm, to further enhance the path optimality.

6 Conclusion

In this thesis, we have proposed two innovative concepts that leverage prior path planning results to enable efficient path planning for modular robots with similar structures, based on a comprehensive review and analysis of existing algorithms. The first algorithm RP uses the previous result as the first part of the path and then tries to reach the desired goal, whereas the second algorithm IKC searches for potential solutions for the inverse kinematics of the new robot based on the previously computed solution path. Unlike existing planners that plan independently of each other, the proposed algorithms avoid some repeated calculations by reusing previous results, thus reducing the time consumption. The testing results reveal that when these two approaches work together (RP-IKC), they exhibit exceptional performance across all evaluated aspects, delivering a path planning acceleration of up to 89%, while not visibly affecting the quality of the computed path, regardless of whether the replacement linkage changes length or shape.

However, we should note that the task investigated in this thesis constitutes merely a small subset of the modular robot path planning problem, thus imposing certain constraints on the generalizability of the results. There is still some future work required to make the process of using these algorithms more efficient and complete, as well as to optimize the obtained paths, as described in Chapter 5. Nevertheless, the findings highlight the promising potential of the proposed concepts, thereby underscoring the necessity to investigate a broader range of application scenarios to unleash the full potential of MRR.

Acronyms

AC alternating current.

BFGS Broyden–Fletcher–Goldfarb–Shanno algorithm.

CCD cyclic coordinate descent.

CoBRA A Composable Benchmark for Robotics Applications.

CV coefficient of variation.

DC direct current.

DoF degrees of freedom.

FABRIK forward and backward reaching inverse kinematics.

FCL Flexible Collision Library.

IK inverse kinematics.

IKC feed Inverse Kinematics Candidates method.

IMPROV interconnectable modules for self-programming and self-verification.

mcs ModRob Configuration Synthesis.

MRR Modular reconfigurable robots.

OMPL Open Motion Planning Library.

OR Original method.

PFS planning-from-scratch.

PRM Probabilistic Roadmaps.

RP Reuse-Path method.

RP-IKC Reuse Path and feed Inverse Kinematics Candidates method.

RRT Rapidly-exploring Random Trees.

RRTC RRT-Connect.

SRMR Self-reconfigurable modular robots.

TCP tool center point.

Timor Toolbox for Industrial Modular Robotics.

TUM Technische Universität München.

WSL Windows Subsystem for Linux.

List of Figures

List of Tables

Bibliography

[1] A. Brunete, A. Ranganath, S. Segovia, J. P. De Frutos, M. Hernando, and E. Gambao. "Current trends in reconfigurable modular robots design." In: *International Journal of Advanced Robotic Systems* 14.3 (2017), p. 1729881417710457.

[2] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira. "Effortless creation of safe robots from modules through self-programming and self-verification." In: *Science Robotics* 4.31 (2019), eaaw1924.

[3] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. "M-TRAN: self-reconfigurable modular robotic system." In: *IEEE/ASME Transactions on Mechatronics* 7.4 (2002), pp. 431–441.

[4] B. Salemi, M. Moll, and W.-M. Shen. "SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system." In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 3636–3641.

[5] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund. "Modular ATRON: Modules for a self-reconfigurable robot." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 2. Ieee. 2004, pp. 2068–2073.

[6] S. B. Liu and M. Althoff. "Optimizing performance in automation through modular robots." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4044–4050.

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

[8] J. Carpentier, F. Valenza, N. Mansard, et al. *Pinocchio: fast forward and inverse dynamics for poly-articulated systems*. https://stack-of-tasks.github.io/pinocchio. 2015–2021.

[9] J. Külz, M. Mayer, and M. Althoff. "Timor Python: A Toolbox for Industrial Modular Robotics." In: *arXiv preprint arXiv:2209.06758* (2022).

[10] I. A. Sucan, M. Moll, and L. E. Kavraki. "The open motion planning library." In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82.

[11] R. J. Alattas, S. Patel, and T. M. Sobh. "Evolutionary modular robotics: Survey and analysis." In: *Journal of Intelligent & Robotic Systems* 95 (2019), pp. 815–828.

[12] S. LIU, Y. CHEN, and W. ZHANG. “MODULAR ROBOTS AND COMPUTER AIDED DESIGN.” Chinese. In: *Robot* 21 (1999), pp. 16–22.

[13] B. Nnaji. *Computer-aided Design, Selection, and Evaluation of Robots*. Manufacturing research and technology. Elsevier, 1986.

[14] K.-H. Wurst. *Flexible Robotersysteme — Konzeption und Realisierung modularer Roboterkomponenten*. Vol. 85. Springer-Verlag, 1991.

[15] K. Stoy, D. Brandt, and D. Christensen. *Self-Reconfigurable Robots: An Introduction*. Mit Press Cambridge, 2010.

[16] M. Mayer, J. Külz, and M. Althoff. *CoBRA: A Composable Benchmark for Robotics Applications*. 2022. arXiv: `2203.09337 [cs.RO]`.

[17] J. Whitman, R. Bhirangi, M. Travers, and H. Choset. “Modular robot design synthesis with deep reinforcement learning.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 06. 2020, pp. 10418–10425.

[18] D. L. Pieper. *The kinematics of manipulators under computer control*. Stanford University, 1969.

[19] A. Medjebouri, F. Inel, L. Rybak, and G. Carbone. “Robust Control Strategies of Puma 560 Robot Manipulator.” In: *Advances in Italian Mechanism Science: Proceedings of the 3rd International Conference of IFToMM Italy 3*. Springer. 2021, pp. 218–227.

[20] J. Wang, M. Yang, F. Liang, K. Feng, K. Zhang, and Q. Wang. “An algorithm for painting large objects based on a nine-axis UR5 robotic manipulator.” In: *Applied Sciences* 12.14 (2022), p. 7219.

[21] K. Erleben and S. Andrews. “Inverse kinematics problems with exact hessian matrices.” In: *Proceedings of the 10th International Conference on Motion in Games*. 2017, pp. 1–6.

[22] W. A. Wolovich and H. Elliott. “A computational technique for inverse kinematics.” In: *The 23rd IEEE Conference on Decision and Control*. 1984, pp. 1359–1363.

[23] C. W. Wampler. “Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods.” In: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1 (1986), pp. 93–101.

[24] S. R. Buss. “Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods.” In: *IEEE Journal of Robotics and Automation* 17.1-19 (2004), p. 16.

[25] R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[26] D. C. Liu and J. Nocedal. “On the limited memory BFGS method for large scale optimization.” In: *Mathematical programming* 45.1-3 (1989), pp. 503–528.

[27] “Differential Kinematics and Statics.” In: *Robotics: Modelling, Planning and Control*. London: Springer London, 2009, pp. 105–160.

[28] D. G. Luenberger, Y. Ye, et al. *Linear and nonlinear programming*. Vol. 2. Springer, 1984.

[29] A. Aristidou and J. Lasenby. "FABRIK: A fast, iterative solver for the Inverse Kinematics problem." In: *Graphical Models* 73.5 (2011), pp. 243–260.

[30] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[31] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[32] R. Geraerts and M. H. Overmars. "A comparative study of probabilistic roadmap planners." In: *Algorithmic foundations of robotics V* (2004), pp. 43–57.

[33] S. M. LaValle. "Rapidly-exploring random trees: A new tool for path planning." In: *The annual research report* (1998).

[34] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[35] D. Berenson, P. Abbeel, and K. Goldberg. "A robot path planning framework that learns from experience." In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3671–3678.

[36] S. Haghzad Klidbary, S. Bagheri Shouraki, and S. Sheikhpour Kourabbaslou. "Path planning of modular robots on various terrains using Q-learning versus optimization algorithms." In: *Intelligent Service Robotics* 10 (2017), pp. 121–136.

[37] S. Paulin, T. Botterill, J. Lin, X Chen, and R Green. "A comparison of sampling-based path planners for a grape vine pruning robot arm." In: *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*. IEEE. 2015, pp. 98–103.

[38] M. Dobiš, M. Dekan, P. Beňo, F. Duchoň, and A. Babinec. "Evaluation Criteria for Trajectories of Robotic Arms." In: *Robotics* 11.1 (2022), p. 29.

[39] K. Trnka and P. Božek. "Optimal motion planning of spot welding robot applications." In: *Applied Mechanics and Materials*. Vol. 248. Trans Tech Publ. 2013, pp. 589–593.

[40] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane. "Computational design of robotic devices from high-level motion specifications." In: *IEEE Transactions on Robotics* 34.5 (2018), pp. 1240–1251.

[41] J. Gregory, A. Olivares, and E. Staffetti. "Energy-optimal trajectory planning for robot manipulators with holonomic constraints." In: *Systems & Control Letters* 61.2 (2012), pp. 279–291.

[42] A. Mohammed, B. Schmidt, L. Wang, and L. Gao. "Minimizing energy consumption for robot arm movement." In: *Procedia Cirp* 25 (2014), pp. 400–405.

[43] J. Pan, S. Chitta, and D. Manocha. "FCL: A general purpose library for collision and proximity queries." In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3859–3866.

[44] V. N. Hartmann, J. Ortiz-Haro, and M. Toussaint. *Efficient Path Planning In Manipulation Planning Problems by Actively Reusing Validation Effort*. 2023. arXiv: `2303.00637 [cs.RO]`.