TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik

Task-driven Composition Synthesis of Modular and Reconfigurable Robot Manipulators

Esra İçer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades einer

Doktorin der Ingenieurwissenschaften (Dr. -Ing.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr.-Ing. Pramod Bhatotia
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Matthias Althoff
	2.	Prof. Dr.-Ing. Martin Buss

Die Dissertation wurde am 30.08.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.04.2022 angenommen.

Abstract

Modular robots have been proposed to overcome the adaptation limitations inherent to standard industrial robots. To complete a given task in an arbitrary environment, modular robots can be reconfigured in a number of different compositions from a set of modules. However, the kinematics and the dynamics of each individual composition (called a module composition throughout this thesis) must be obtained to determine how they will be used in a given task. This time consuming process has restrained the widespread use of modular robots in industry-especially in those with limited experience with robotics. With the development of automated generation of kinematics and dynamics of modular robots, the have become more practical and the potential of modular robots is better understood.

However, the different assemblies (called as *module compositions* throughout this thesis) have their own unique kinematics and dynamics. Hence, the kinematics and the dynamics must be obtained for each module composition. This makes the usage of modular robots troublesome- especially for the people with limited knowledge on robotics. With the help of the automated generation of kinematics and dynamics of modular robots, they have become more practical and their potential is well-understood.

This thesis seeks to generate automated solutions to the cost-optimal module composition problem for a given task. This body of work tackles the two main pieces of this problem: *i*) establishing the automated cost-optimal composition synthesis algorithms, and *ii*) determining the manner of performing a task in a cost optimal way. To this end, the following three main contributions of this thesis are the development of: *i*) an exhaustive search-based automated module composition generation algorithm that systematically finds the cost-optimal solution, *ii*) an evolutionary algorithm-based optimal module composition algorithm that considers the task-related objectives while finding the cost-optimal solution, and *iii*) computationally fast hierarchical path planning methods.

Throughout this thesis, the theoretical considerations behind both automated composition synthesis algorithms and path planning algorithms are theoretically demonstrated in different chapters. Detailed simulations and experiments demonstrate the efficiencies of the proposed algorithms. The results show that the proposed methods give accurate solutions in a reasonable time that enable users to easily reconfigure modules to fulfill various tasks.

Zusammenfassung

Modulare Roboter stellen einen vielversprechenden Ansatz dar, das Anpassungsproblem von Standard-Industrierobotern an beliebige Aufgaben und/oder Umgebungen zu lösen. Modulare Roboter können auf verschiedenste Weise rekonfiguriert werden um so neue Baugruppen aus einem Satz von Modulen zu erzeugen. Jede Anordnung hat jedoch ihre eigene, einzigartige Kinematik und Dynamik. Daher müssen diese für jede Modulzusammensetzung neu ermittelt werden. Dies macht die Verwendung von modularen Robotern problematisch-insbesondere für Personen mit begrenzten Kenntnissen in der Robotik. Durch die automatisierte Generierung von Kinematik und Dynamik modularer Robotern lässt sich deren Potential der praktischen Anwendung näher bringen.

Diese Dissertation befasst sich mit der automatisierten Generierung der kostenoptimalen Modulkomposition für eine gegebene Aufgabe. Im Hauptteil dieser Arbeit werden hierzu die beiden wichtigsten Teilprobleme betrachtet: *i*) der Aufbau des automatisierten kostenoptimalen Kompositionssynthesealgorithmus und *ii*) die Bestimmung von Methoden zur kostenoptimalen Aufgabenerfüllung. In diesem Zusammenhang sind die drei wichtigsten Ergebnisse dieser Arbeit: *i*) ein auf erscöhpfender Suche basierender automatischer Algorithmus zur Erzeugung von Modulkompositionen, welcher die kostenoptimale Lösung auf systematische Weise findet, *ii*) ein auf evolutionären Algorithmen basierender optimaler Modulkompositionsalgorithmus, der auf der Suche nach der kostenoptimale Lösung findet die aufgabenbezogenen Ziele berücksichtigt, und *iii*) hierarchische Pfadplanungsmethoden, die es ermöglichen, einen Pfad in kurzer Rechenzeit zu finden.

In den verschiedenen Kapiteln dieser Dissertation werden sowohl automatische Kompositionssynthesealgorithmen, als auch Bahnplanungsalgorithmen an theoretischen Beispielen demonstriert. Um die Effizienz der vorgeschlagenen Methoden zu bewerten, werden detaillierte Simulationen und Experimente durchgeführt. Die Ergebnisse zeigen, dass die vorgeschlagenen Methoden in brauchbarer Zeit genaue Ergebnisse liefern und so Benutzer ermöglichen, auch ohne Kenntnisse der Robotik, einen gegebenen Modulsatz neu zu konfigurieren, um verschiedene Aufgaben zu erfüllen.

To my family.....

Acknowledgements

First of all, I would like to express my utmost gratitude and give sincere thanks to my Ph.D. advisor Prof. Dr.-Ing. Matthias Althoff for his continuous support in guiding me through my research and his extreme dedication for the whole period I spent under his supervision. This thesis would not have been possible without his patience and attention to my studies. Furthermore, without him, I would not have had the chance to meet exceptional people that I have met through my graduate years. I also would like to thank him for the great working environment he created within the Cyber-Physical Systems (CPS) research group, which I feel extremely lucky to have been a part of.

I would also like to thank Prof. Dr. Ata Muğan, Assoc. Prof. Dr. Pınar Boyraz, and Dr. Cihat Bora Yiğit for their support for the time spent at Istanbul Technical University during my research secondments and for the related research collaborations.

Additionally, I would like to thank to all members of the CPS research group especially, Ahmed, Albert, Bastian, Carmela, Christian, Dongkun, Hendrik, Markus, Sebastian, Silvia, Stefan and Stefanie, for the inspiring and friendly working environment that we created at the Chair of Robotics and Embedded Systems of the Technische Universität München. I also would like to thank Amy Bücherl and Ute Lomp for their friendliness and helpfulness from my first day in the chair. I also need to thank to Marie-Luise Neitz, Alexander Lenz and the Schleissheimer Strasse team for their couragement during my time in Schleissheimer Strasse. I am also grateful for many invaluable support of Burcu Karadeniz, Emeç Erçelik, Sina Shafaei, Ali Nasseri and Alper Tunga Çelebi.

I am also grateful for the invaluable help and support of all early-stage researchers of the EC FP7 SMART-E project: Aaron, Alex, Andrea, Constantin, Martijn, Mateo, Stefan, Stefania, Stefano, Yasmin, Roy, Taimoor, and Saber. In particular, I would like to thank my officemate Aaron for his support during these three years and Andrea and Martijn for the fruitful collaborations we made during our Ph.D. times.

A special thanks to my students, George Mesesan (whose work appears in Section 4.3), Gerardo Lopez Alonso (whose work appears in Section B.2), Heba Ali Hassan (whose work appears in Chapter 4) and Ali Parvez Anjum (whose work appears in Appendix B) for their great efforts that contributed to this thesis.

Finally, my family deserves the most gratitude: my mother Şadimen İçer, my sister Yasemin İçer Toker, my brother Yücel Toker and my nephew Mustafa Yiğit Toker. Without their love, patience and support, this dissertation would not exist. Lastly, I would like to thank my father, A. Nuri İçer who always encouraged me doing a Ph.D.. I finally kept my promise and I wish you could have seen it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This introductory chapter provides a general overview of modular and reconfigurable robots, particularly those employed in industrial settings. This chapter also addresses the main challenges to this area, the motivation behind this thesis, as well as its scope and addressed goals. Then, the thesis is outlined and the definitions used throughout are explained.

1.1 Scope and Goals

Industrial robots are an inseparable part of today's manufacturing processes because they are efficient, robust, and accurate in their execution of tasks. These fixed-assembly robots consist of structures that are designed to perform specific sets of tasks in predefined environments. Although their performance may be sufficient for a particular workspace or task, their adaptation to flexible tasks or changing environments limits their use in many industrial environments. Products in manufacturing are becoming more and more complex as the variety of products increases daily. Therefore, the continual purchase of new robots for different tasks makes the manufacturing process expensive. As an affordable solution to this problem, modular and reconfigurable robots have been proposed, which consist of a set of predesigned modules that can be reassembled into different robot structures. These module sets are grouped into the following categories: base modules, joint modules, link modules, and end effector modules. While base modules fix the robot onto a mobile platform or a desired position, joint modules provide the degrees of freedom (DOF) to a robot and contain mechanisms to move it. Link modules provide different lengths and connections to each robot and end effector modules provide the tools for various tasks. Numerous robot compositions can be generated by using different modules or by changing the order of modules in the assembly (see Figure 1.1).

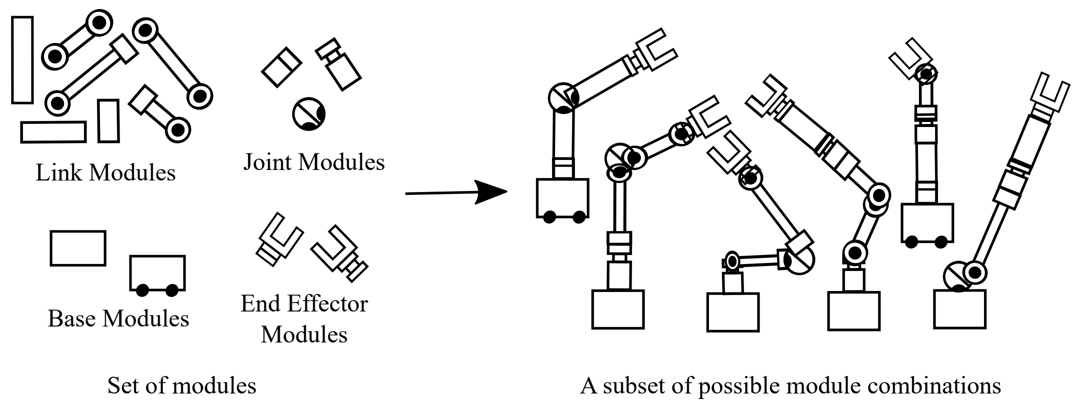


Figure 1.1: The illustration of a modular robot setup.

The idea of modular and reconfigurable robots was proposed in the 1960s [10] and became extremely popular in the late 1980s. Since then, they have attracted many researchers all over the world and a large number of studies have been published in the literature [10]. Modular robots have been divided in two groups based on their reconfigurability type: self-reconfigurable modular robots and manually-reconfigurable modular robots [11].

Self-reconfigurable robots have the ability to configure themselves with or without human assistance and generally consist of small cubic- or prismatic-shaped modules [11]. More details about self-reconfigurable and modular robots can be found in [10, 12–20]. However, self-reconfiguration in large-sized robots, like industrial manipulators, can be difficult or infeasible because the self-movement of a large module is restricted by weight and different features. All large-sized modular robots and a subset of small-sized modular robots that need a human operator for assembly are categorized as manually-reconfigurable modular robots.

Modular and reconfigurable robotic systems can be reconfigured in three different ways: *i*) on the module-level, *ii*) on the assembly-level, or *iii*) on the configuration-level [21]. In each category, the system's flexibility is obtained differently. For example, in module-level reconfigurable systems, flexibility is obtained by changing the module's parameters such as the size or the feature of the modules. Changing the module order or the selection of the assembly port enables the user to obtain different solutions in assembly-level reconfigurable systems. Finally, in configuration-level reconfigurable modular robotic systems, flexibility is obtained by changing the configurations of the modules [21].

Another classification of modular and reconfigurable robots considers their structure. In terms of the structure type, modular robots can be divided into three categories: *i*) serial robots, *ii*) parallel robots, or *iii*) hybrid robots [22]. Serial robots have an open-loop kinematic chain. For instance, starting from a base module and ending with one or more end effector modules (for tree-like robot structures, they may have more than one end effector module). Parallel robots have two or more closed-loop kinematic chains and there is more than one independent kinematic chain between the base and the end effector modules. Hybrid robots are the combinations of serial and parallel robots and they have both open-loop and closed-loop kinematic chains.

Throughout this thesis, we focus on manually-reconfigurable, serial, and assembly-level modular and reconfigurable industrial manipulators. The main advantages of these robots can be summarized as follows:

- *High versatility and task interchangeability:* Modular and reconfigurable robots are more adaptive when compared to standard fixed-assembly robots because they can be easily and quickly reconfigured for different tasks. Thus, their adaptability in case of a component or product change is high.
- *Easy maintenance and upgrade:* In case of malfunction, the broken module can be easily replaced by another module without a need for specific expertise. Moreover, the modules can be attached or detached quickly and thus reduces the lead time.
- *Efficiency:* There is no guarantee that a fixed-assembly robot can fulfil a changing task. In such a situation, a new robot is required or the current robot must be reprogrammed by an expert, which causes time loss during production. Because modular robots can be reassembled in various ways, they offer more efficient solutions for a variety of tasks when compared to standard fixed-assembly robots.
- *Low cost:* A standard industrial robot might be useless when tasks change, thus costing a lot of money for enterprises. Moreover, a malfunction in a robot causes a break in a production line that also leads to money loss. Modular and reconfigurable robots address all of these problems and only an initial investment cost is required. Besides these reasons, the production of repeated modules also decreases the per-module cost.

Considering the above-mentioned advantages, modular and reconfigurable industrial robots are expected to be popular in the manufacturing environment in the near future. Because this technology provides a solution for future flexible manufacturing scenarios, their potential impact is high. The main customers for modular and reconfigurable industrial robots are small and medium-sized enterprises (SMEs)[1] because of the decrease in risk of investment and increase in the level of automation. As a result, modular and reconfigurable industrial robots are a perfect solution for companies, especially those

[1]SME: Based on the definition provided by the European Commission in [23], SMEs are defined as:

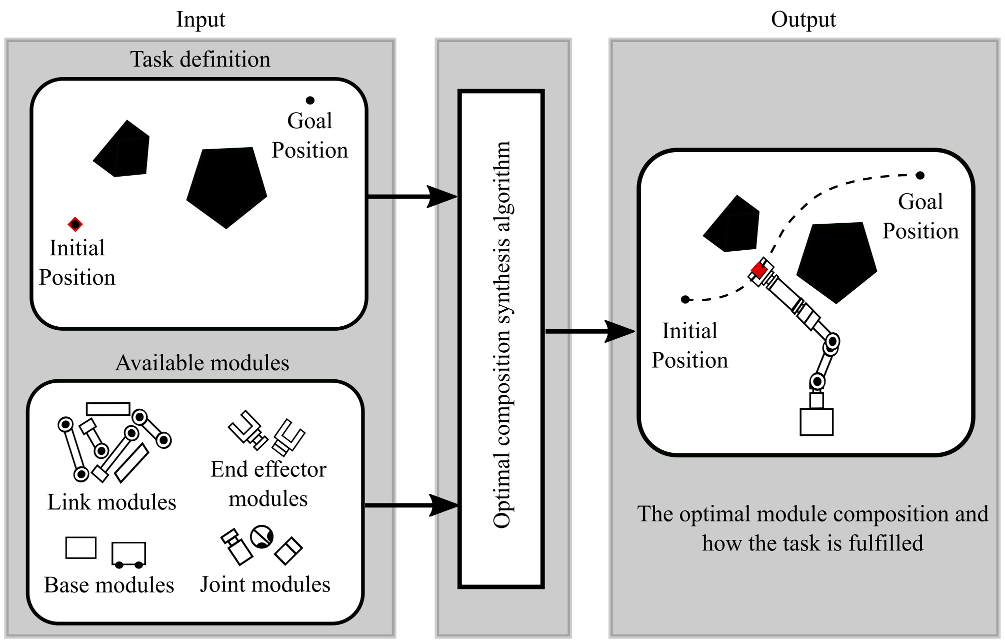


Figure 1.2: An illustration of the problem of finding the optimal module composition for modular robots.

with limited budgets. Although modular and reconfigurable industrial robots have many advantages, their concept also has some drawbacks such as *i*) an enormous number of different module compositions can be generated even from a small number of different modules and the user may not know which module composition provides the best solution for a given task, *ii*) mechanical components are complicated compared to standard industrial robots because reconfiguration requires a complex connection mechanism, *iii*) apart from standard robots, each module is an individual component in modular and reconfigurable robots, thus, the payload capacity is reduced, and *iv*) control and communication systems are also more complicated when compared to standard robots [24, 25].

One of the main challenges to modular robotics is the problem of determining the cost-optimal composition for given modules for a given task as illustrated in Figure 1.2. A plethora of compositions can be generated by varying modules or changing their order. This large search space makes the modular robot synthesis problem complex and time consuming. Designing a robot under consideration of a given task is called a task-based design (TBD) problem and it includes the kinematic and the dynamic models of the robot, its trajectory definition, and its control [25]. In addition to that, the optimal solution is desired as an output for most TBD problems, which is then called a task-based optimal design (TBOD)

- Staff headcount is less than 250, and
- annual turnover is less than or equal to €50m or annual balance sheet total is less than or equal to €43m.

problem. The TBOD problem is a highly nonlinear, complex, and computationally expensive problem because the design space rapidly increases with number of modules or change to their order. In addition, the objective function and its constraints are coupled and nonlinear [26]. Since storing models of all the different robot assemblies requires a huge amount of disk storage space, an automated model generation method is needed. After generating the kinematic and the dynamic model of a module composition, the module composition is checked by an algorithm to see if it fulfils the given task in a cost-optimal way. The cost is defined by the user, which can be defined in many forms such as minimum execution time, minimum power consumption, minimum jerk, or minimum displacement of the joints of the robot, etc. Obtaining the cost-optimal composition among all possible compositions is a highly nonlinear optimization problem in a discrete search space and the lack of solutions to this problem limit the usability of modular and reconfigurable robots.

1.2 Modular and Reconfigurable Industrial Manipulators

The first-known concept for modular and reconfigurable robotic systems was proposed in the 1960s by Neumann [27] in which a framework for modular and reconfigurable robots was presented [10]. Starting from the late 1980s, several modular and reconfigurable industrial robots have been designed and produced. One of the first examples of modular and reconfigurable robots was named the Reconfigurable Modular Manipulator System (RMMS), which was developed at Carnegie Mellon University [28]. The RMMS consists of joint and link modules that have discrete mechanical and electrical units (see Figure 1.3(a)). Each produced module is independent and has a standardized connection mechanism that allows the user to easily attach or detach modules. The connector design enables the user to connect the modules in eight different orientations. All joint modules have an actuator, a transmission mechanism, a position sensor, and a power amplifier to control the motor. Their design has two different types of 2-DOF joint modules to vary the direction of movement, namely, rotational and pivot joints. Besides hardware, the proposed software to operate this robot is also modular [29]. The modules are manually assembled considering task requirements and a centralized controller is automatically generated with the help of the developed modular software.

Modular Reconfigurable Robots (MRRs) developed at Nanyang Technological University is another example of early modular robot design, which consist of link modules of different lengths and revolute, prismatic, and pivotal joint modules [30]. As opposed to the RMMS, these designed modules have cubic or prismatic shapes that can be configured to form serial, parallel, or hybrid manipulators (see Figure 1.3(b)). All modules have connectors on all of their faces, which increase the total number of

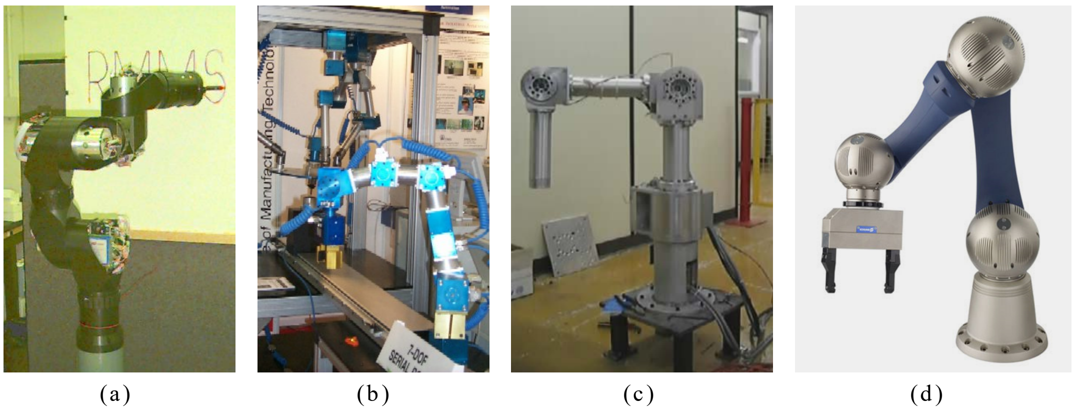


Figure 1.3: Different modular robot designs (a) the reconfigurable modular manipulator system (RMMS) developed by Carnegie Mellon University (CMU) [1], (b) modular reconfigurable robots (MRRs) developed by Nanyang Technological University [2], (c) the Waterloo modular and reconfigurable robot (WMRR) developed by the University of Waterloo [3] and (d) the SCHUNK LWA 4P robot developed by SCHUNK GmbH & Co. KG [4].

possible assemblies. A host controller-based control system is used and the communication between modules is achieved by a controller area network (CAN) bus. Although this idea is applicable to industrial use, the designed modules are not robust enough for industrial applications due to their low payload capacity [31].

The Toshiba Modular Manipulator System (TOMMS) was designed by Toshiba Corporation in the first half of the 1990s. Their design includes a 1-DOF revolute module with three input and two output connectors per module and a link module consisting of concentric two cylinders that enable the user to adjust the link length [32]. TOMMS uses a single control software with no specification and the robot is controlled with a joystick operated by a human. However, the designed modules can generate robots with a maximum of 3-DOF. Control inputs are sent from the joystick and the inverse Jacobian matrix-dependent control algorithm calculates the desired joint values.

The Waterloo Modular and Reconfigurable Robot (WMRR) in Figure 1.3(c) was developed by the University of Waterloo [3]. The WMRR has three different joint modules with varying power capabilities and link modules of multiple lengths. The designed joint modules are categorized as rotational, pivotal, or perpendicular-rotational joints. Each joint module has an embedded micro-controller unit (MCU) board that allows the user to control the joint actuator, communication between modules, and synchronize the operation [3].

Another modular robot manipulator was developed by the Technical University of Munich to be used for agricultural purposes [33, 34]. Their design consists of one prismatic joint module, eight revolute joint modules with three different sizes, and link modules with different lengths. They use a bus system to transfer data while fulfilling a task and each joint is controlled by a decentralized control system on the motor drivers on a position or velocity level. Although the idea is modular, they focus on three different structures, namely, 6-DOF, 7-DOF and 9-DOF, and only use these pre-determined structures in their tasks. They place large and relatively heavy modules close to the base module and only use the above-mentioned three assemblies when considering the requirements for tasks.

Besides these mentioned research-oriented modular robot designs, several companies have also designed industrial modular robots for commercial use. The company SCHUNK GmbH & Co. KG designs and produces commercially available modular robots [35]. They have several modular industrial manipulators such as the SCHUNK LWA 4D and the SCHUNK LWA 4P (see Figure 1.3(d)). One of the latest generations of their lightweight arm SCHUNK LWA 4P consists of three spherical 2-DOF joint modules called *powerballs*, two link modules with different designs, and various types of end effector modules whose details can be found in [35]. Although this robot is modular, its reconfigurability is limited because the input and output ports of the link modules are different and each joint module is not compatible with all other modules.

The company igus® designs Robolink®, which has pre-defined and user-defined structure options [36]. The design has only one type of rotational joint module and various link modules with different lengths. The current modules allow the user to configure 2-DOF to 5-DOF robots with a maximum 3-kg payload capacity [36] and its payload capacity decreases with the increase in the robot's DOF. One well-known robot producer, COMAU, also produces a modular robot called *e.Do* that is used for personal or educational purposes. It is designed as an industrial robot but the commercially available version does not fit the industrial scenarios due to limited capabilities such as payload capacity, which is only 1 kg [37].

The previously mentioned modular and reconfigurable industrial robot manipulators developed by research institutes or universities have not been adapted to a large-scale production and are not yet commercially available [3, 28, 30, 32–34]. As a result, it is not possible to buy or rent them for use in experiments. The commercially available ones produced by the companies listed above [35–37] have several drawbacks. The payload capacity of Robolink® is limited and its capacity decreases with an increase in the number of DOFs, which limits its industrial usage. Similar to Robolink®, the load capacity of *e.Do* is also too low even with the structure predetermined by the company.

1.2.1 Modules Designed for Numerical and Real Experiments

Throughout this thesis, we use two different module sets to implement our algorithms. In view of the robots detailed above, we design new virtual modules to perform our algorithms in a simulated environment. The first module set is designed considering the most widely used modules in industrial robots. Although the modules are not physically produced, they are utilized in the simulation environment (more details can be found in Appendix A). Users can employ the same module as many times as they prefer, which increases the number of possible assemblies.

The second module set is the extended version of the SCHUNK LWA 4P robot and this set allows us to demonstrate our algorithms in a real environment. When compared to other commercially available robots, the 7-kg load capacity of the SCHUNK LWA 4P robot makes it preferable even though it has a reconfigurability problem. Because designing and producing a new robot from scratch is time consuming and not the target of this thesis, the Chair of Robotics, Artificial Intelligence and Real-time Systems purchased the SCHUNK LWA 4P robot. New modules were designed to solve its reconfigurability restrictions and its was reconfigurability augmented as explained in Appendix B.

1.3 Formulation of the Optimal Task-based Modular Robot Problem

Serially connected, modular, and reconfigurable robot manipulators are considered in this problem with n-DOFs and kinematics defined uniquely by the vector $\mathbf{q} \in \mathbb{R}^n$ of the joint positions, where angles stand for rotational joints and translations stand for prismatic joints. Throughout the thesis, we assume *i*) four different types of modules, which are bases, joints, links, and end-effectors, *ii*) each link module and each joint module has only one input connection port and one output connection port. Each base module has only one output connection port and each end effector module has only one input connection port, and *iii*) all manipulators begin with a base module, end with an end effector module, and there are only joint and link modules between the base and end effector modules. These are realistic assumptions for industrial robots.

The variable $k \in \{1, \dots, N\}$ uniquely refers to a possible module composition and the variable N refers to the maximum number of possible compositions without considering task-defined constraints. The task requirements are constrained by the kinematic and dynamic models of the module composition and static obstacles in the environment. The environmental space containing the robot and obstacles is denoted by $\mathcal{W} \subset \mathbb{R}^3$. The subset of space occupied by the robot is $\mathcal{A} \subset \mathcal{W}$ and the robot's occupancy for a joint position vector $\mathbf{q}$ is shown by $\mathcal{A}(\mathbf{q}) \subset \mathcal{W}$. The variable $\mathcal{A}(\mathbf{q}_k) \subset \mathcal{W}$ indicates the space occupied by the k^{th} composition. Obstacles can be in any geometric shape in $\mathbb{R}^3$ and the union of all obstacles is shown by $\mathcal{O} = \bigcup_j \mathcal{O}_j$. An obstacle-free space in the environment is defined as $\mathcal{F} = \mathcal{W} \backslash \mathcal{O}$.

We indicate the time variable with $t \in [0, t_f]$ and t_f is the total time to reach the goal with an initial time of zero. The vector $\mathbf{q}(t)$ maps the time t to the joint position vector. The forward kinematic function from the joint position vector $\mathbf{q}(t)$ to the end effector position is denoted by $f(\mathbf{q}(t))$. We define all path planning problems starting from a given initial position $\mathbf{p}_s$ as $\mathbf{p}_s = f(\mathbf{q}(0))$ and ending at a given position $\mathbf{p}_g$ as $\mathbf{p}_g = f(\mathbf{q}(t_f))$.

The dynamic model of the modular robot is obtained from the Newton-Euler formula in (1.1), where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the definite symmetric mass matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n\times n}$ is the Coriolis and centrifugal matrix, $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ is the gravity vector term, $\boldsymbol{\tau} \in \mathbb{R}^n$ is the actuation forces/torques vector, $\dot{\mathbf{q}}(t)$ is the joint velocity vector, and $\ddot{\mathbf{q}}(t)$ is the joint acceleration vector (see [38]).

$$\boldsymbol{\tau}(t) = \mathbf{M}(\mathbf{q}(t))\ddot{\mathbf{q}}(t) + \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t))\, \dot{\mathbf{q}}(t) + \mathbf{g}(\mathbf{q}(t)) \tag{1.1}$$

The set of module combinations, $\mathcal{K}$, that fulfils a given task is described as:

$$\begin{aligned} \mathcal{K} = \{k \quad | \ 1 \le k \le N \wedge \forall t \in [0, t_{f,k}] \ \exists \ \mathbf{q}_k(\,\cdot\,) : \\ &\wedge \ \mathcal{A}(\mathbf{q}_k(t)) \in \mathcal{F} \\ &\wedge \ \mathbf{q}_k(t) \in \ [\mathbf{q}_{k,min}, \ \mathbf{q}_{k,max}] \\ &\wedge \ \dot{\mathbf{q}}_k(t) \in \ [\dot{\mathbf{q}}_{k,min}, \ \dot{\mathbf{q}}_{k,max}] \\ &\wedge \ |\boldsymbol{\tau}_k(\mathbf{q}_k(t))| \le \boldsymbol{\tau}_{k,max} \\ &\wedge \ f_k(\mathbf{q}_k(0)) = \mathbf{p}_s \ \wedge \ f_k(\mathbf{q}_k(t_{f,k})) = \mathbf{p}_g\}. \end{aligned} \tag{1.2}$$

The composition that gives the cost-optimal solution is denoted by κ and the optimal cost value for the k^{th} composition is denoted by c_k:

$$\kappa = \underset{k\in\mathcal{K}}{argmin} \ c_k. \tag{1.3}$$

The motivation behind this thesis is to develop algorithms that find the optimum composition κ.

1.4 Author's Contribution

New methods to determine the optimal composition of modular and reconfigurable robots for a desired task have been proposed by the author. Finding the cost-optimal module composition from an enormous number of different module combinations can be divided into two subsections: *i*) the determination of the composition synthesis algorithm, and *ii*) the determination of the method to perform the task in a cost-optimal way. In the first step, research is focused on classification of the tasks that must be fulfilled and the production of exhaustive search-based generation methods to obtain all possible compositions. Because checking all possible compositions is inefficient and time consuming, a method based on the systematic elimination of unfeasible compositions is proposed. This approach was first published in [24]. This algorithm is also applied in [5] to a real-world problem given by industry and implemented using the extended SCHUNK LWA-4P robot (as detailed in Appendix B). This method enables the user to generate all feasible compositions that can fulfil the desired task and shows them how each module composition can achieve a given task.

Secondly, optimality is taken into consideration and the research is concentrated on finding the task-optimal solution that can be generated from the module set. The generation and computation of all possible compositions to achieve a given task is time-consuming and not essential. Both deterministic and evolutionary approaches to solve the task-based cost-optimal composition synthesis problem are proposed throughout this thesis. The implementation of deterministic optimization methods is the first approach used to find the task-optimal solution. The composition synthesis method is based on the elimination of the less-likely compositions from being the optimal one in each iteration and the proposed exhausted search-based was published in [39]. In the second proposed method, evolutionary algorithms are used to find the task-optimal solution, which was published in [40]. Task-related objectives are used in the evaluation of generated compositions and thus reduces the search space of the potential compositions. A set of the best compositions considering the evaluation function was generated. Then, the best composition is selected from this subset of compositions.

Because a large number of compositions exist, an algorithm that not only efficiently synthesizes compositions but also performs the task in a short computational time is required. Due to the fact that a task can be achieved in numerous ways (especially for redundant robots), fast path planning algorithms are needed to find how a robot fulfils a given task in an optimal way in the shortest computational time. A hierarchical path planning method was introduced in [6] which generates multiple collision-free paths and then finds the shortest collision-free path. A hierarchical path planning method consisting of *i*) multiple local task-space RRT planners running in parallel, and *ii*) a global planner that uses the free-space connectivity information for coordinating the local planners was introduced in [41].

To make each chapter clear, we explain terms which are widely used throughout the thesis. *Module composition* is the assembly of the modules which has the information of which module is used in the robot structure and the order of the used modules in the structure. *Configuration* is the joint position vector of a robot structure. *Posture* is the position of the manipulator in work space in a specific set of joint angles. Posture is composed of a the specified position and the specified orientation of the end effector. *Robot kinematics* is the motion of the a robot mechanism without considering the forces and torques that cause it. In the kinematic model, forces, torques, masses and moments of inertia are disregarded and the position and the orientation of the end-effector is calculated with the help of the robot kinematics. *Robot dynamics* is the representation of the relationship among the driving force or torque on the joint actuator, the external work load, the mass and inertia. *Path* is a curve in pose space specifying position and attitude. A path specification contains no time specification. *Trajectory* is a time-parametrized curve in pose space specifying position and attitude and time. Implicitly, a trajectory specifies a velocity profile of motion along a path in pose space.

1.5 Thesis Outline

The problem of building task-based optimal compositions of modular and reconfigurable robots is structured in this thesis with two main subsections: task-based composition synthesis algorithms and task execution algorithms. It should be noted that the chapters are independent with each chapter considered as a stand-alone contribution. In each chapter, a detailed literature review is given at the beginning, then problems are formulated, and then the proposed solutions are detailed afterwards. At the end of each chapter, the applicability of the proposed methods are demonstrated with various examples.

Chapter 2 describes the proposed optimal composition generation method. The main contributions of this chapter are *i*) an exhaustive search-based possible compositions synthesis method, and *ii*) an exhaustive search-based the optimal composition generation method. The proposed approach for *(i)* is based on discarding unfeasible compositions in a systematic way. The contribution for *(ii)* is based upon elimination of the least-likely compositions during the optimization process. Deterministic optimization methods are also considered in this chapter.

Chapter 3 details a new evolutionary algorithm-based optimal composition generation method. An approach that considers task requirements in the objective function is explained in detail. The proposed method only considers a set of best module compositions while applying time-consuming tasks and it gives faster solutions when compared to finding optimal assemblies by individually optimizing trajectories for each assembly.

Unlike the other chapters, Chapter 4 focuses on the execution of given tasks and novel path planning algorithms. A path planning method that aims to reach the goal positions following the largest gaps in the environment was proposed. In these regions, the probability of the existence of obstacles is lower, therefore it allows the user to find a path in a shorter computational time. Additionally, a two-level path planning algorithm is introduced. Due to the large number of different module compositions, task-space path planners are considered in the first level and refined planning is applied in the second level of the algorithm. Both methods highly decrease the computational time and comparisons are given with several examples. The conclusion of the thesis is provided in Chapter 5, which also includes possible future research directions.

To quickly demonstrate the results of the simulations used throughout this thesis, the reader is invited to explore the virtual module set, the extended SCHUNK LWA 4P robot, and the generated visualization toolbox found in Appendix A, Appendix B, and Appendix C, respectively.

Chapter 2

Exhaustive Search-Based, Cost-Optimal Composition Synthesis of Modular and Reconfigurable Robots for a Given Task

Standard industrial robots are robust, efficient, and accurate while demonstrating remarkable performance of a single pre-defined task. On the other hand, it is difficult to adapt standard industrial robots to new tasks or environments. Modular robots are a solution to this problem because they can be reassembled in a variety of ways. However, one drawback associated with modular robots is difficulty with determining the optimal module composition from a set of modules for a given task. In this chapter, we propose a search-based composition synthesis algorithm for modular robots that eliminates the unfeasible or least-likely compositions in a hierarchical order. It is well known that search-based approaches are accompanied by high computational times. However, the algorithm proposed in this thesis overcomes this drawback by systematically eliminating unfeasible compositions with tests that ascend in difficulty and computational time. Thus, to dramatically decrease the simulation time, unfeasible compositions are first rejected through simple tests, which leaves the more complex and time-consuming tests for the few remaining compositions.

First, a literature review on presently available modular robot synthesis algorithms is conducted in Section 2.1 and the task-based modular robot synthesis problem is formulated in Section 1.3. Then, the proposed algorithm is detailed in Section 2.2 and the assumptions made in the algorithm are detailed in Section 2.3. The task requirements are explained in Section 2.4 and its implementation is demonstrated for two module sets in Section 2.6.

2.1 Literature Review

The task-based robot design problem, which determines the optimal lengths of link modules for a pre-defined structure, has been investigated for decades [42–45]. In our task-based design problem, only modular robots are considered with the aim of finding the optimal assembly of modules from a set of pre-defined modules so-called the *optimal module composition* throughout this thesis. As understood from the definition, the task-based robot design problem not only considers executing the given task in the optimal way, but also targets the module composition that can perform said task in an optimal way among many potential module compositions. Thus, the search space increases tremendously. In this section, existing algorithms are reviewed for a task-based, optimal module composition of modular robots. The existing works consider this problem as three sub-problems: assembly representation, model generation, and the task-optimal module composition selection. Literature reviews of each sub-problems are detailed in the following sub-sections.

2.1.1 Representation of the module composition

To generate different module compositions, several assembly representation methods have been proposed in [31,46,47]. These methods are classified as graph theory-based methods, task-based configuration methods, and axiomatic design theory-based methods (ADT) (more details can be seen in [47,48]).

Authors in [46] introduce a graph-based technique called the assembly incident matrix (AIM) to represent module compositions. They use cubic-shaped or rectangular prism-shaped link modules that have ports on each side (one port for cubes or square faces of rectangular prism-shaped modules and two ports on rectangular faces of rectangular prism-shaped modules). As a result, there are many assemblies generated from different modules that produce the same module composition due to symmetry in the modules and the locations of their connection ports. Thus, AIM enables the user to discard identical assemblies from a set of potential module compositions. Then, serial or tree-structured module compositions can be generated with the help of AIMs. For each module composition, a string is generated using AIMs and this string represents a unique module composition.

Another research group in [47] combines graph-based and axiomatic design theory-based techniques to represent different module compositions. First, they group the module compositions considering their functional requirements (such as topology, kinematic chain structure, etc.) and then they narrow the search space with the features of the system like module and connection types. They also consider similar modules as in [46] and use AIM for their assembly representation. The authors insert additional data to AIM considering the modules' features and this new representation is called the object incidence matrix (OIM). All matrices are stored in a database and they use this database for their task-based optimal design (TBOD) algorithms.

Besides, the configuration coupling matrix (CCM) is introduced in [49]. The authors consider the direction of the output ports of joint modules, length of link modules, and the direction of the relative connection between two sequential joints to represent the assembly of their module compositions. Different from previous groups, link modules with one input connector and one output connector are designed in [31] and the authors propose a novel modular robot representation called the kinematic matrix representation (KMR). Similar to CCM, KMR has information on the orientation of each component, the type of joint module, and the type of link module. Then, the models of possible module compositions are generated using these matrices.

Another representation of module composition is proposed in [50] and the authors parameterize their modules considering their properties like geometric dimensions, control parameters, etc., in the representation of their robot. Their modules are represented as nodes and their connections are represented by edges. Their assembly representation is done with configuration graphs and their method is called parameterized module configuration graphs (PMCG). This parameterized representation enables the user to represent various fixed modules in one variable and decrease the computational time spent for module composition generation.

Throughout this thesis, we consider simple modules that are detailed in Appendix A and Appendix B. Both module sets consist of modules that have only one input and output connection ports and they can be assembled in a fixed orientation. Considering their simplicity, the complex matrix representations as in [31, 46, 47, 49] are not essential. In the module representation method that we follow, all modules are classified based on their types and all their properties are embedded within, which enables the user to easily select the modules and configure them. More information about the module composition representation that is used throughout this thesis is given in Section 2.3.

2.1.2 Model generation

After assembling the modules, the kinematic and dynamic models of the robot compositions need to be generated. For modular robots, it is important to automatically obtain the kinematic model of the module composition because each module composition has unique kinematic and dynamic models and manual attainment of these models for each composition is troublesome. The previous studies on automated kinematic model generation are mainly focused on the product-of-exponentials (POE) formulation and Denavit-Hartenberg (D-H)-based methods [22].

The POE formulation consists of 3×3 rotation matrices and 3×1 transformation matrices, both of which transmit information among components (for more detail see [51]). In [52], the authors introduce a POE formula-based, automated kinematic model generation method for tree-structured modular robots so-called *dyad kinematics*. The proposed method is independent of the module composition and only considers cubic-shaped modules with connection ports on each side. Using the module composition representation method in [46], the proposed algorithm automatically derives forward kinematics using dyad kinematics.

The same authors improve their automated model generation algorithm in [53] to obtain not only the kinematic model of the module composition but also the dynamic model using the Newton-Euler algorithm. However, kinematic models obtained from these methods are independent of joint type since POE-based methods represent different types of joint motions as the same. Another dyad kinematics-based, automated forward kinematic algorithm for modular robots is proposed in [54]. AIM is used to represent the assembly angles in their method and they introduce a new term called the path matrix to define a multi-chain assembly. In their proposed method, sequential link and joint modules are defined as a dyad and the forward kinematics solution is obtained by multiplication of the dyads considering the gathered path matrix. It should be noted that POE-based methods are only applicable to serial or tree-like structured robots [22].

D-H-based methods generally use 4×4 homogeneous matrices to define relations among the components [55]. In [56], the authors propose a new D-H-based kinematic model generation method for all types of modular robots in which they generalize the existing D-H method. The proposed automated kinematic and dynamic modelling method maps the modular design variables on the module assembly and generates D-H parameters. The same authors also develop a specific finite element method (FEM)-based automated kinematic and dynamic model generation method for modular robotic systems in [22], which is applicable to serial, parallel, or hybrid robots. Different from the previously mentioned approaches, the D-H-based method presented in [57] considers two sets of coordinate frames for robot definition: a modular frame and a D-H frame. Kinematic parameters of the modules are given in the modular frame and the D-H notation is represented in the D-H frame. The kinematic model of the whole robot is automatically obtained by transforming the modular frame to the D-H frame. In [8], the authors extend the standard D-H convention adding new parameters to make the automated generation procedure easier and all kinematic and dynamic parameters are stored in each module.

In addition to kinematic and dynamic model generation, the calculation of inverse kinematics (IK) also plays an important role in the TBOD problem. To solve the IK problem, analytic, numerical, and meta-heuristic methods have been used [58]. Analytic methods are not common in modular robotics because they depend on the structure of the robot. Moreover, they are not efficient in case there are multiple or infinite (i.e., redundant robots) IK solutions, which is one of the main difficulties of the IK problem. To address this difficulty, numerical methods and meta-heuristic methods have been widely used for IK calculations [58]. The authors introduce the Newton-Raphson iteration method as an IK solver for modular robots [59] and they use an extra DOF to avoid the singularities. Similar to this work, another Newton-Raphson iteration method-based IK solver is introduced in [60]. The main difference between those works is the authors in [60] use a local POE formula for their model generation and also consider non-redundant and tree-like robot structures in their calculations. Another task-based inverse kinematics solver for serial manipulators is proposed in [61] and uses a mixture of the Newton-Raphson algorithm and the genetic algorithm (GA). This combination gives better results in terms of computational time and accuracy when compared to only the Newton-Raphson algorithm. In [31], the authors consider all possible IK solutions for each composition and select the optimal IK solution among them. They implement the Niching genetic algorithm-based IK solver proposed in [62].

Considering its easy applicability and flexibility, we implement the method proposed in [8] while generating the kinematic model and the dynamic model of potential module compositions. More details on the kinematic model generation are given in Section 2.4.2.

2.1.3 Task-optimal module composition selection

The task-optimal module composition selection is the last step of the algorithm. In most of the works done so far, evolutionary algorithm-based task optimal module composition methods have been used [63–67]. All those works aim to optimize a given objective function by applying a predetermined evolutionary algorithms. A more detailed literature review of these kind of algorithms is presented in Section 3.1.

A framework for the task-based design of serial manipulators with revolute joints, called the progressive design, is introduced in [65]. The method consists of four sequential steps: design, prototype, planning, and control. The multi-population genetic algorithm (MPGA), which basically uses the standard GA in parallel, is used to solve the highly nonlinear optimization problem by pursuing the following procedure: first, it selects the number and the orientation of the joint modules, then it defines the lengths of link modules and joint angles for predefined sequential positions and then it generates a planner to reach intermediate task points in the given order.

The same research group presents a numerical approach in [66] to find the optimal module assembly for fault-tolerant systems to perform the given task considering reachability, joint limits, obstacle avoidance, and a measure of isotropy. They use task specifications as an input to their algorithm and the algorithm determines the kinematic configuration for the desired manipulator. Then, it selects the modules that can generate the desired kinematic configuration. After that, the proposed algorithm synthesizes the feasible module compositions, which is called form synthesis, and finally it optimizes the dimensions of the components to create the desired form called dimension synthesis. A simulated annealing algorithm is used to optimize the objective function and the algorithm works by penalizing module compositions that do not satisfy all task requirements. An analytical approach for a 2-DOF planar manipulator and a numerical approach for a 6-DOF manipulator is proposed in [68] to solve the IK problem of modular robots. They use D-H notation to calculate the forward kinematics and consider Pieper's inverse kinematic solution for their analytic approach and simulated annealing for their numeric approach. However, these proposed methods are limited as they do not consider singularities. The same authors also present another approach in [69] to determine the configuration of fault tolerant modular robot manipulators for a given task, which has the same objective function as in [66]. They aim to find the best inverse kinematics solution using their simulated annealing-based inverse kinematics algorithm and define the module composition that gives the best inverse kinematics solutions for a given task. Another task-based serial manipulator synthesis algorithm is introduced by the same authors in [70]. They propose a concept of a rapidly deployable manipulator system to solve the drawbacks of the fixed manipulators with their developed modular and reconfigurable control software. The same authors introduce an agent-based approach in [71] for modular robots that simultaneously considers kinematics, dynamics, trajectory planning, and control in the synthesis algorithm.

Different from the aforementioned work, a concurrent design procedure is applied in [21] in which the authors reduce the number of independent variables. They consider the types of modules and their quantities as design variables and task specifications while using joint limits as design constraints. The authors implement GA as an optimization algorithm and consider the manipulability, error measures, torque requirements, and the difference in joint angles between sequential points in the objective function.

A hierarchical selection method that classifies the modules according to their physical capabilities is presented in [72]. While generating assemblies, the authors categorize each module based on their features and cancel the sets of modules and assemblies that have incompatible groups of modules. However, this elimination process only considers simple physical design rules as it decreases the search space. The same authors extend their method in [73] and categorize the design problem into three levels: *i*) module level, *ii*) sub-assembly level, and *iii*) assembly level. In the module level, the proposed method eliminates the modules that are not suitable for the given task such as those with a specific type of power supply or maximum payload capacity. Then, it cancels unfeasible module combinations in sub-assemblies such as module compositions in which the end effector is an intermediate component or the joint's capability is not sufficient enough to fulfil the task.

A method to obtain the optimal design of 6-DOF modular parallel manipulators based on choosing the composition from a look-up table is proposed in [63]. The table is generated based on the experiences and knowledge within the database. Geometric and actuator parameters are considered as design parameters and the task is defined as reaching a predefined set of task points. The performance constraints are set as reachability, joint and torque limits, and accuracy. The look-up table is generated by the authors, which consider the type of task, and users are able to decide which composition is appropriate for the task using the look-up table. Thus, they can obtain an optimal robot composition from the selected compositions using the simplex method. The same authors also introduce a two-stage design methodology to obtain the task-specific optimal configurations for reconfigurable parallel manipulators in [74]. In the first stage of the proposed algorithm, the robot structure is determined using an enumeration of the modules. In the second stage, design parameters are determined considering the given objective function.

A stochastic programming-based approach for modular and reconfigurable robots is proposed in [75]. The approach is based on a two-stage decision-making process called a two-stage mixed integer linear stochastic model. In the first stage, a decision is made with uncertain future possibilities about the task and a set of possible module compositions is defined afterwards. The set of compositions are checked as to whether they are feasible for the given task in the second stage.

All the works mentioned above consider evolutionary-based synthesis algorithms and these algorithms do not guarantee reaching the global optimal solution. The reason behind this is that evolutionary algorithms only consider a subset of their search space and are therefore unable to check all possible solutions. Approaches that consider and investigate all potential solutions are needed. However, checking all potential solutions is not feasible with existing methods. Therefore, we aim to find a solution to this research gap and propose a method that considers all possible solutions. The main contribution of this chapter is a search-based hierarchical composition synthesis algorithm that finds the cost-optimal solution for any user-defined task. The proposed method applies the brute-force algorithm to generate all possible compositions and eliminates the unfeasible compositions by tests that are sorted from the simplest one to the most complicated one. With these tests, one can obtain all potential module compositions that can fulfil the given task. The proposed cost-optimal composition generation algorithm checks the obtained solution at a specified number of iterations and eliminates the least-likely module compositions. With the proposed algorithm, the problem of finding the global optimum solution using heuristic algorithms and the problem of high computational time with search-based algorithms can both be solved.

2.2 The Cost-Optimal Composition Synthesis Algorithm for Modular Robots for the Execution of a Given Task

We propose a solution to the cost-optimal module composition for the given task problem: a hierarchical composition elimination method. The main idea behind this proposed method is to group task requirements into subtasks (called as *tests*) and to implement these tests starting from the simplest and computationally fastest test to the more complicated and time-consuming ones. After the implementation of each test, the unfeasible compositions that cannot fulfil this test are eliminated and more complex and time-consuming tests are only applied to the remaining compositions. The required tests can vary among different tasks and some tests are not needed for some tasks (see Table 2.1). For example, welding does not require an additional path planning because all points that a robot must follow have already been determined, therefore, the algorithm skips the path planning test. Details of these tests are explained in Section 2.4. After the implementation of all required tests for all generated module compositions, those compositions that can fulfil the given task (called as *feasible compositions*) are obtained.

To find the cost-optimal module compositions among all feasible compositions, trajectory planning is implemented afterwards. When compared to tests, the computational time to find the cost-optimal trajectory is extremely high, therefore implementation of this to all feasible compositions is not efficient. To handle this disadvantage, we propose a method that eliminates the cost-inefficient compositions after every user-defined number of iterations. We roughly optimize all remaining compositions in parallel and record the obtained cost values for each user-defined number of iterations. The obtained cost values for all feasible compositions are compared to the best obtained cost value among them in every user-defined iteration is obtained as in (2.1),

$$c_{k,p} = \frac{c_{k,l} - c_{best,l}}{c_{k,l}} \tag{2.1}$$

where $c_{k,l}$ is the cost value of the k^{th} composition in the l^{th} iteration and $c_{best,l}$ is the minimum cost value among all remained compositions in the l^{th} iteration. The value of $c_{k,p}$ is compared with a specific threshold determined by the user. This composition is eliminated if it is greater than the threshold and it is no longer considered in further iterations. This procedure is applied to all module compositions remaining in each iteration and it is repeated until the optimal cost values of all remaining module compositions are obtained. When the optimization procedure ends, there may be several remaining module compositions. To find the optimal composition, the obtained cost values are compared and the module composition that gives the best solution is selected as (1.3).

2.3 Preliminaries

The first step of the proposed algorithm is the assembly representation and search space determination. Most industrial modular robots are similar to standard industrial robots in terms of single input and output connection ports. The module sets used throughout this thesis (in Appendix A and Appendix B) are similar to standard industrial robots in terms of single input and single output connection ports with a single orientation and so forth. These simple module features do not require a complex assembly representation as the matrices in [31, 46, 47]. Even with these module features, one can generate numerous different module compositions. For example, one can generate infinite different module compositions using the virtual module set as explained in Appendix A. However, some of these compositions are apparently not feasible to form a robot structure such as compositions consisting of only one module or compositions considering a base module or an end effector module as an intermediate module (see Figure 2.1). To restrict the search space, it is assumed that modules can only be assembled in the following order:

Base - Joint - Link - Joint - · · · - Link - End Effector.

The crossed compositions in Figure 2.1 are a set of discarded compositions based on this assumption. Starting from the base module, all possible robot compositions are generated from the required minimum number of DOF to the desired maximum number of DOF, which are shown as n_{min} and n_{max}, respectively. We consider that each joint module has one DOF and each link module and base module have zero DOF. The variable x shows the total number of DOF of the end effector. The total number of all possible module compositions can be obtained from the following formula:

$$N = \sum_{i=n_{min}}^{n_{max}-x} \tilde{b} \cdot \tilde{j}^{i} \cdot \tilde{l}^{i} \cdot \tilde{e} \tag{2.2}$$

where $\tilde{b}$, $\tilde{j}$, $\tilde{l}$, $\tilde{e}$ indicate the number of unique base modules, joint modules, link modules, and end effector modules, respectively. This assumption substantially decreases the search space, i.e., for the first set of modules, the search space decreased to 15552 different compositions to generate a 6-DOF robot. Nevertheless, the search space is still exceptionally large and should be pruned to speed up the

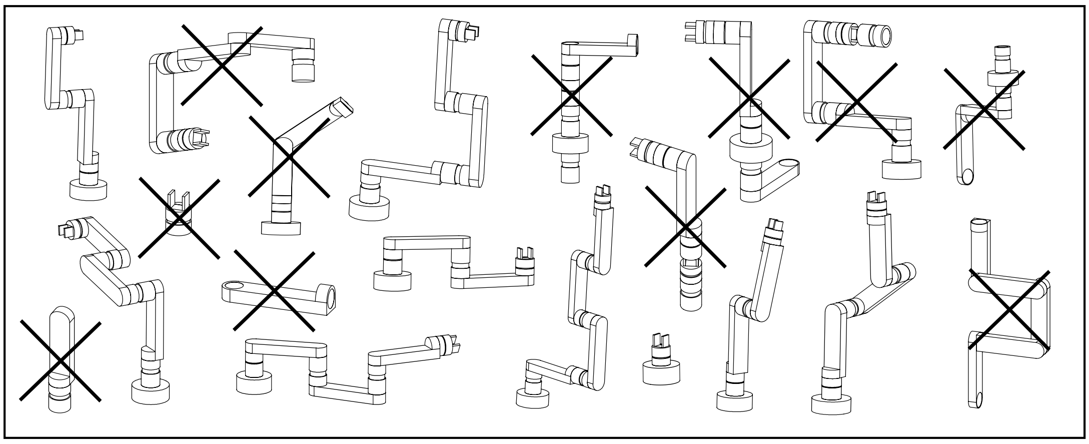


Figure 2.1: A set of possible module compositions generated from the first set of modules detailed in Appendix A and the eliminated composition using the above-mentioned assumption.

process. To do this, the task requirements (as detailed in Section 2.4) are determined so that the search space decreases after each test.

Each module is encoded with a unique name and the following data: module type, kinematic parameters, dynamic parameters, and module properties such as length, thickness, and limits for joint modules.

2.4 Task Requirements

As a second step, the task requirements should be considered in the design procedure. The most common tasks in manufacturing environments are assembling, pick and place, welding, spot welding, and gluing [76]. In this section, requirements (called *tests*) for the above-mentioned tasks in the manufacturing industry are explained in detail (see Table 2.1). These tests are *i*) reachability test, *ii*) kinematic test, *iii*) static force-torque test, *iv*) collision test, and *v*) path planning test. The main motivation behind classifying tasks into tests as mentioned above is that these tests are individually simple and they are all

Table 2.1: Possible tasks for industrial robots and required tests for them

	Pick and Place	Welding	Gluing	Spot Welding
Reachability Test	✓	✓	✓	✓
Kinematic Test	✓	✓	✓	✓
Static-torque Test	✓	✓	✓	✓
Collision Test	✓	✓	✓	✓
Path Planning Test	✓	-	-	✓

independent of one another. Every test determines a prerequisite ability that is a part of fulfilling the entire task.

We sort tests in the following order considering their simulation time: reachability test, kinematic test, static torque/force test, path planning in the obstacle-free environment, and path planning in the obstacle-laden environment. The total number of compositions tested decrease with each test and finally, optimal trajectory planning is implemented. For example, the order of the pick and place test, which is one of the tasks required by all tests, is

$$N_{\text{reachability}} \geq N_{\text{kinematics}} \geq N_{\text{static}} \geq N_{\text{path planning without obstacles}} \geq N_{\text{path planning with obstacles}}$$

where $N_A = |\kappa_A|$ and $|\kappa_A|$ is the cardinality of κ_A.

2.4.1 Reachability Test

The reachability test demonstrates whether the desired point is in the workspace of the robot, and it is obvious that a robot cannot fulfil the requested task if desired points are not in its workspace. To check this, the point is mapped on the robot's configuration space and the availability of a solution is checked. Although simple, checking this for all compositions takes time and it is a computationally expensive process. Therefore, to quickly discard the unfeasible compositions, we only consider the reachability in terms of a robot's total length and check whether the given point is in a sphere with a radius matching the maximum total length of the robot. The maximum length of the robot (l) is obtained from:

$$l = l_B + l_E + \sum_{i=1}^{n-x} (l_{J,i} + l_{L,i}) \tag{2.3}$$

where l_B, $l_{J,i}$, $l_{L,i}$, and l_E are the length of the base module, the length of the i^{th} joint module, the length of the i^{th} link module, and the length of the end effector module, respectively. For prismatic joints, the value of $l_{J,i}$ is the length of the joint with the maximum displacement. The variables n and x are the number of DOFs of the robot and the end effector module, respectively. The distance between the base location of the robot $\mathbf{p}_b$ and the desired position to be reached $\mathbf{p}_d$ is calculated by $d_{b2d} = \|\mathbf{p}_d - \mathbf{p}_b\|_2$, where $\|\mathbf{p}_d - \mathbf{p}_b\|_2$ is the Euclidean distance between $\mathbf{p}_d$ and $\mathbf{p}_b$. Then, whether the module composition can satisfy the condition $l \geq d_{b2d}$ is checked. If a module composition cannot satisfy the condition, it is not considered in further tests.

2.4.2 Kinematics Test

The kinematic test demonstrates whether a robot can reach the desired point. Kinematics includes the motion of structures without considering the forces or moments that results in the motion. Finding the end effector's position with joint angles is called forward kinematics and finding the angles of joints from the end effector's position is called inverse kinematics (IK). Because each module composition has different kinematic models, generating kinematic models for all possible compositions is not efficient. To solve this problem, we implement the automated kinematic modelling method proposed in [8]. The authors extend the D-H convention, where all kinematic and dynamic parameters are stored in each module, to make the automated generation procedure easier.

In standard D-H parameters, there are four variables, which are: a_i, d_i, α_i, and θ_i. The variable a_i denotes the distance along the x_i-axis between the origins of the frames o_i and $o_{i'}$. The variable d_i denotes the distance along z_{i-1} between the origins of the frames o_{i-1} and $o_{i'}$. The variable α_i denotes the angle between the axis z_{i-1} and z_i around the x_i axis and the variable θ_i denotes the angle between the axis x_{i-1} and x_i around the z_{i-1} axis. In the method proposed by [8], the authors extend the standard D-H convention with two new parameters and thus facilitate automated capture of the standard D-H parameters: a_i, α_i, p_i, n_i, and γ_i where p_i represents the distance along z-axis between the origin $o_{i'}$ and joint connection PJ_{i-1} (joint between $link_i$ and $link_{i-1}$), n_i represents the distance along z axis between o_i and joint connection PJ_i (joint between $link_i$ and $link_{i+1}$), and γ_i is the parameter that shows the angular offset between the sequential x-axes at the time that the joint is in its zero position ($q_i = 0$). These three variables are used to determine d_i and θ_i considering the joint type. More details about the method in [8] can be seen in Appendix D.

As an inverse kinematics algorithm, we use the unit quaternion-based, closed-loop inverse kinematics (CLIK) algorithm from [77] in the proposed method. Based on the CLIK algorithm, the relationship between $\dot{\mathbf{p}}$ and $\boldsymbol{\omega}$ is shown in (2.4), which represents the linear velocity and angular velocity of the end effector, respectively. The variables $\mathbf{J}(\mathbf{q})$ and $\dot{\mathbf{q}}$ denote the geometric Jacobian of the manipulator at the joint variables $\mathbf{q}$ and joint velocities, respectively.

$$\begin{bmatrix} \dot{\mathbf{p}} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{2.4}$$

The given rotation matrix in Euler angles is shown in (2.5). The unit quaternions are defined as $\mathbf{Q} = \{\eta, \boldsymbol{\epsilon}\}$, where the variables η and $\boldsymbol{\epsilon}$ represent the scalar part and vector part of the quaternion, respectively, and are shown in (2.6) and (2.7), respectively.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.5}$$

$$\eta = \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1} \tag{2.6}$$

$$\boldsymbol{\epsilon} = \begin{bmatrix} \frac{1}{2} sgn(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \frac{1}{2} sgn(r_{13} - r_{31})\sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \frac{1}{2} sgn(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix} \tag{2.7}$$

Based on the method in [77], the desired end effector position and orientation are represented as $\mathbf{p}_d$ and $\mathbf{Q}_d = \{\eta_d, \boldsymbol{\epsilon}_d\}$, respectively. The position and orientation errors are calculated as in (2.8) and (2.9), respectively, where **S** is the matrix operator that gives the cross product between the (3×1) vector.

$$\mathbf{e}_{pos} = \mathbf{p}_d - \mathbf{p}(\mathbf{q}) \tag{2.8}$$

$$\mathbf{e}_{ori,quat} = \eta(\mathbf{q})\, \boldsymbol{\epsilon}_d - \eta_d\, \boldsymbol{\epsilon}(\mathbf{q}) - \mathbf{S}(\boldsymbol{\epsilon}_d)\, \boldsymbol{\epsilon}(\mathbf{q}) \tag{2.9}$$

The velocities of the joints are calculated as in

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q}) \begin{bmatrix} \dot{\mathbf{p}}_d + \mathbf{K}_{pos}\, \mathbf{e}_{pos} \\ \dot{\boldsymbol{\omega}}_d + \mathbf{K}_{ori}\, \mathbf{e}_{ori,quat} \end{bmatrix} \tag{2.10}$$

where $\mathbf{K}_{pos}$ and $\mathbf{K}_{ori}$ are positive gain matrices. The CLIK algorithm calculates the position and orientation errors and runs until those values are less than the user-defined threshold. If the algorithm cannot find a solution, the composition is discarded for the following tests.

2.4.3 Static Force-Torque Test

The static force-torque test shows if a robot's joints can carry the given payload or not. Each joint in an industrial manipulator has a static force-torque capacity to carry a specific payload. Forces based on the carried payload are distributed across all joints and affect the robot's entire structure. While calculating the effect of the payload on each joint, the wrench vector $\mathbf{W} \in \mathbb{R}^6$ is used where $\mathbf{W} = [\mathbf{f}; \mathbf{n}]$. The two variables $\mathbf{f} \in \mathbb{R}^3$ and $\mathbf{n} \in \mathbb{R}^3$ indicate the force and moment vectors applied to the end effector, respectively. If the wrench vector of an end effector **W** is known, the static force-torque values applied to each joint can be calculated from:

$$\boldsymbol{\tau}(\mathbf{q}) = \mathbf{J}(\mathbf{q})^{\mathrm{T}}\, \mathbf{W} \tag{2.11}$$

where $\boldsymbol{\tau}(\mathbf{q})$ is the static force-torque vector and $\mathbf{J}(\mathbf{q})^{\mathrm{T}}$ is the transpose of the Jacobian matrix at the joint position $\mathbf{q}$. In this test, the static force-torque vector is checked for the satisfaction of the condition $\boldsymbol{\tau}(\mathbf{q}) \leq \boldsymbol{\tau}_{max}$ where $\boldsymbol{\tau}_{max}$ is the maximum force-torque vector of the manipulator. The robot composition is discarded from further tests if it does not satisfy the condition.

2.4.4 Collision Test

The collision test is implemented to check if there are any collision-free solutions for a robot's given task. Collision checking is a complex and computationally expensive work that can be done either in the configuration space (C-space) of the robot or in the workspace [78]. One way to check for collisions is to map all obstacles onto the robot's joint space. These mapped obstacles in C-space are called C-obstacles. However, the projection of obstacles onto C-space has exponential complexity in the number of DOFs and, because each module composition's C-space is different, the projection of obstacles for each module composition is required [79]. Considering these drawbacks, the implementation of C-space collision detection algorithms is difficult and computationally inefficient for our problem. Therefore, we use a second collision checking method that gives faster solutions when compared to the C-space collision detection algorithms. This is because workspace collision detection methods do not require mapping obstacles onto the robot's joint space.

To simplify the problem and reduce the computational resources, we consider the following assumptions: *i*) all modules are modelled as spheres or cylinders considering the ratio of a module length to a module radius (if the ratio $l/r >> 1$ modules are modelled as cylinders, otherwise, modules are modelled as spheres), and *ii*) obstacles are enclosed by a sphere or spheres about their shape. For example, in case the l/r ratio of an obstacle is around one (e.g. cubes), the obstacle is considered to be enclosed by one sphere. Otherwise, the obstacle is represented by more than one sphere as in [80]. For both module sets used in this thesis, the l/r ratio is much greater than 1 for link modules and around 1 for joints, end effectors, and base modules. For this reason, link modules are considered as cylinders and represented by $(x_s, y_s, z_s, x_e, y_e, z_e, r)$ where the subscripts $(\,\cdot\,)_s$ and $(\,\cdot\,)_e$ represent the center points of the input port and the output port of the cylinder, respectively. The notations x, y, and z represent the Cartesian coordinates of the center points and r is the radius of the component. Other components are represented by their center points and their radii (x_c, y_c, z_c, r). We perform collision checking in the following categories: *i*) collision check between spheres, *ii*) collision check between a cylinder and a sphere, and *iii*) collision check between cylinders.

2.4.4.1 Collision between spheres

The distance between the center points of two spheres, i.e., $\mathbf{p}_A$ and $\mathbf{p}_B$, are calculated from $d_{A2B} = \|\mathbf{p}_A - \mathbf{p}_B\|$. The variable d_{A2B} must satisfy the condition $d_{A2B} \geq r_A + r_B$ where r_A is the radius of sphere A and r_B is the radius of sphere B.

This procedure is applied to collision checks between the base module and obstacles, the base module and joint modules, the base module and end effector modules, end effector module and obstacles, joint module and obstacles, and between two joint modules. Using forward kinematics, the position of the center of the end effector is obtained as in Appendix D. The coordinates of the center of each joint are obtained by the subsequent multiplication of the homogeneous transformation matrices of D-H frames as in Appendix D. The length of the last joint is added to the transformation matrices to find the joint's center point.

2.4.4.2 Collision between a cylinder and a sphere

The distance between a cylinder and a sphere is considered as the distance between a point and a line. Collisions between link modules and obstacles or any components except other link modules are some examples of this type of collision. Each link module is modelled as a cylinder, which can be represented by a line and a radius. The start point of the line $\mathbf{p}_{L,s}$ and the endpoint of the line $\mathbf{p}_{L,e}$ are calculated using transformation matrices as explained in Appendix D. The start and the end point of the i^{th} link is calculated as in (2.12) and (2.13) where l_i is the total length of the i^{th} link and p^{pl} is the distance along the z-axis between the origin $o_{i'}$ and joint connection PJ_{i-1} as in Figure D.2 in Appendix D.

$$T^i_{L,s} = T^i_0 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & p^{pl} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$T^i_{L,e} = T^i_0 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

The first three rows of the last column of $T^i_{L,s}$ and $T^i_{L,e}$ give $\mathbf{p}^i_{L,s}$ and $\mathbf{p}^i_{L,e}$, respectively. The line made using those two points defines the center line of the cylinder. The distance between the line and a point is represented by the subscript $(\cdot)_P$ and is calculated as

$$d_{L2p} = \frac{|(\mathbf{p}_{L,e} - \mathbf{p}_{L,s}) \times (\mathbf{p}_{L,s} - \mathbf{p}_P)|}{|\mathbf{p}_{L,e} - \mathbf{p}_P|}. \tag{2.14}$$

Then, this distance is compared to the sum of the obstacle's radii and the cylinder $d_{L2p} \geq r_L + r_P$. If the distance is less than the sum, a collision between the link module and obstacle or end effector module and joint module will occur. This procedure is repeated for all link modules in the robot's composition and all obstacles in the environment along with all other components of the robot.

2.4.4.3 Collision between cylinders

The collision between two cylinders can be calculated as the distance between two lines. Therefore, the collision between two link modules is represented by a collision between cylinders and the distance between two cylinders is calculated as follows:

$$d_{L_1 2 L_2} = \frac{|(\mathbf{p}_{L_2,s} - \mathbf{p}_{L_1,s}) \cdot ((\mathbf{p}_{L_1,e} - \mathbf{p}_{L_1,s}) \times (\mathbf{p}_{L_2,e} - \mathbf{p}_{L_2,s}))|}{|(\mathbf{p}_{L_1,e} - \mathbf{p}_{L_1,s}) \times (\mathbf{p}_{L_2,e} - \mathbf{p}_{L_2,s})|} \tag{2.15}$$

where the subscript $(\cdot)_{L_i}$ represents line 1 and line 2. The minimum distance between two lines is compared against the sum of the radii of cylinders $d_{L_1 2 L_2} \geq r_{L_1} + r_{L_2}$ and the composition is discarded from the further tests if it does not satisfy the condition.

2.4.5 Path Planning Test

The path planning test is implemented to find feasible points in the environment that a robot can follow while moving from initial to final position. This test is important because the task cannot be executed if a path between these two positions does not exist. To perform a path planning test, two methods are used: *i*) configuration space (C-space) approaches and *ii*) Cartesian space approaches [81]. Because C-space approaches provide better efficiency and singularity avoidance, they are widely preferred for the generation of a feasible path between the initial and goal positions. To form the path, first the initial and goal positions are mapped onto joint space using the inverse kinematics, which maps the problem from workspace to C-space. Then, any path planning algorithm in the literature (see [78, 81]) can be used to generate a feasible path between the initial and goal positions. The rapidly-exploring random trees-connect (RRT-connect) algorithm is selected in the proposed method due to its simplicity and high success rate. Collision detection and obstacle avoidance is also needed to find a feasible path between two points. As explained in Section 2.4.4, we implement collision checks in the workspace to all knots obtained from the path planning algorithm, regardless of which algorithm is used. Considering only the kinematic requirements, the remaining compositions from all the tests completed so far are now considered capable of potentially fulfilling the given task.

Indeed, all tests done up until this point help the user find all feasible module compositions that can fulfil the given task and roughly show how each composition can achieve the task. All steps explained so far give a solution to the task-based design (TBD) problem. After finding all feasible module compositions, optimal trajectory planning is implemented for the remaining compositions to find the task-optimal solution. The following sections are for finding a solution to the task-based optimal design (TBOD) problem.

2.5 Optimal Trajectory Planning

The optimal robot composition is the composition that fulfils the given task considering the user-defined objective function with the constraint of minimum cost. For optimal trajectory planning, the task is to generate the optimal trajectory considering the given objective function. Finding the optimal trajectory for a robot is considered an optimal control problem. The general form of an optimal control problem is defined as

$$\underset{\mathbf{x}(\,\cdot\,),\mathbf{u}(\,\cdot\,),t_f}{\text{minimize}} \int_0^{t_f} L(\mathbf{x}(t), \mathbf{u}(t))\, dt \tag{2.16}$$

where $\mathbf{x}(\,\cdot\,)$ indicates the vector of state variables shown as $\mathbf{x} = (\mathbf{q}; \dot{\mathbf{q}})$ and $\mathbf{x} \in \mathbb{R}^{2n}$, $\mathbf{u}(\,\cdot\,)$ indicates the vector of control variables described as the torques/forces acting on the joints where $\mathbf{u} = \boldsymbol{\tau}$ and $\mathbf{u} \in \mathbb{R}^n$, and $L(\mathbf{x}(t), \mathbf{u}(t))$ defines the desired objective function as in [82]. The value obtained from (2.16) gives the optimal cost value, c, in (1.3). While calculating this value, robot dynamics are transformed from (2.16) to the state space form which is showns as $\dot{\mathbf{x}} = g(\mathbf{x}_k(t), \mathbf{u}_k(t))$. Point to point motion is constrained by $\mathbf{x}(0) = \mathbf{x}_0$ and $\mathbf{x}(t_f) = \mathbf{x}_f$ where $\mathbf{x}_0$ and $\mathbf{x}_f$ indicate the initial and final state variables, respectively. Joint limits and collision avoidance constraints are defined as inequality constraints in the function $h(\mathbf{x}(t), \mathbf{u}(t))$.

Direct methods can transform the infinite problem in (2.16) into a finite-dimensional nonlinear problem and provide better computational efficiency and accuracy in comparison to other methods in the literature [82]. We use the direct multiple shooting method in our problem because it is more robust and faster than other methods, it is easy to parallelize, and it is applicable to unstable systems [82]. The idea behind the direct multiple shooting method is to discretize the state variables and control variables into N_{int} pieces between the initial and goal positions. Each discretized control value is formulized as $\{\forall t : \mathbf{u}(t_{int}) = \mathbf{b}_m,\ t_{int} \in [t_m, t_{m+1}]\}$ to simplify the problem where $m \in \{0, \ldots, N_{int-1}\}$. An ordinary differential equation (ODE) solver is implemented individually to each interval (t_{int}). The points generated by the path planning method and artificial velocity values are considered as state variables where the starting points of each interval are represented as $\mathbf{x}_m(t_m) = \mathbf{s}_m$. The results obtained from the ODE solver give the trajectories and each trajectory must satisfy the following condition at each interval: $\mathbf{s}_{m+1} = \mathbf{x}_m(t_{m+1}; \mathbf{s}_m, \mathbf{b}_m)$.

We rewrite the optimal control problem given in (2.16) considering the direct multiple shooting method for modular robots as in (2.17). The cost-optimal trajectory for the k^{th} composition is formulated in (2.17) where the variable $l_{k,m}$ represents the discretized objective function for the k^{th} composition in the interval $[t_{k,m}, t_{k,m+1}]$. The variables $\mathbf{x}_{k,min}$ and $\mathbf{x}_{k,max}$ are the minimum and maximum state variables for the k^{th} composition, respectively, and the variables $\mathbf{u}_{k,min}$ and $\mathbf{u}_{k,max}$ are the minimum and maximum control variables for the k^{th} composition, respectively. The vector $\mathbf{h}$ indicates the inequality constraints. The occupancy set of the robot is projected onto a function $d_{k,j}(t)$ and thus represents the distance between the k^{th} robot's components and j^{th} obstacles at time t. The variable r_j is the radius of the j^{th} obstacle.

$$c_k = \underset{\mathbf{s}_k, \mathbf{b}_k}{\text{minimize}} \sum_{m=0}^{N_{int}-1} l_{k,m}(\mathbf{s}_{k,m}, \mathbf{b}_{k,m}) \tag{2.17}$$

subject to

$$\begin{aligned}
&\mathbf{s}_{k,0} - \mathbf{x}_{k,0} = 0, && \text{(initial constraints)}\\
&\mathbf{s}_{k,m+1} - \mathbf{x}_{k,m}(t_{k,m+1}; \mathbf{s}_{k,m}, \mathbf{b}_{k,m}) = 0, && \text{(ODE model)}\\
&\mathbf{h}(\mathbf{s}_{k,m}, \mathbf{b}_{k,m}) \geq 0, && \text{(path constraints)}\\
&\mathbf{s}_{k,N_{int}} - \mathbf{x}_{k,f} = 0 && \text{(terminal constraints)}
\end{aligned}$$

where

$$\mathbf{h}(\mathbf{s}_{k,m}, \mathbf{b}_{k,m}) = \begin{pmatrix} \mathbf{x}_{k,max} - \mathbf{s}_{k,m} \\ \mathbf{s}_{k,m} - \mathbf{x}_{k,min} \\ \mathbf{u}_{k,max} - \mathbf{b}_{k,m} \\ \mathbf{b}_{k,m} - \mathbf{u}_{k,min} \\ d_k(t) - \mathbf{r}_j \end{pmatrix}.$$

Initial constraints, final constraints, and path constraints represent a part of (1.2) and those parts can be shown as $f_k(\mathbf{q}_k(0)) = \mathbf{p}_s$, $f_k(\mathbf{q}_k(t_{f,k})) = \mathbf{p}_g$, and the remaining parts of (1.2), respectively. In the determination of the optimal trajectory, collision avoidance is considered as an inequality constraint during optimization and the collision test detailed in Section 2.4.4 is implemented. It should also be noted that the first two rows of $\mathbf{h}(\mathbf{s}_{k,m}, \mathbf{b}_{k,m})$, $\mathbf{x}_{k,max} - \mathbf{s}_{k,m}$ and $\mathbf{s}_{k,m} - \mathbf{x}_{k,min}$, correspond to $\mathbf{q}_k(t) \in [\mathbf{q}_{k,min},\ \mathbf{q}_{k,max}]$ and the third and fourth rows of $\mathbf{h}(\mathbf{s}_{k,m}, \mathbf{b}_{k,m})$, $\mathbf{u}_{k,max} - \mathbf{b}_{k,m}$ and $\mathbf{b}_{k,m} - \mathbf{u}_{k,min}$, correspond to $|\boldsymbol{\tau}_k(\mathbf{q}_k(t))| \leq \boldsymbol{\tau}_{k,max}$.

As explained in Section 2.2, the values for all feasible compositions obtained from (2.17) are recorded at every user-defined iteration and compositions that are the least-likely optimal compositions are discarded. At the end of the simulation, a set of c values in (1.3) is obtained from (2.17) and the best solution among the set (κ in (1.3)) is selected as the optimal one.

2.6 Numerical Experiments

This section demonstrates the application of the proposed optimal composition synthesis algorithm to several scenarios to illustrate the applicability of the method. All computations are performed on a standard computer with an Intel$^{®}$ Core$^{\text{TM}}$ i7 processor with 2.30 GHz and 16 GB of RAM. We implement our algorithm as explained in Chapter 2 in both module sets in Appendix A and Appendix B. For the first module set, all experiments are done in the simulation environment and it is assumed that there is an unlimited number of pieces from each module. The details of the simulations are explained in Section 2.6.1. In the second experiment, we use the extended SCHUNK LWA 4P robot. As opposed to the simulation, there is only one piece from each module and the experiment is demonstrated in the real world. The task is based on a real problem from industry, which is defined as inserting an insulation material into a car door in the minimum time, and it is explained in Section 2.6.2.

To compare the simulation time for each test, we generate random scenarios for the pick and place task that requires the implementation of all tests. As seen in Table 3.1, we sort tests in the following order considering their simulation time: reachability test, kinematic test, static torque/force test, path planning in the obstacle-free environment, path planning in the obstacle-laden environment, and optimal trajectory planning. The total number of compositions decrease after each test.

2.6.1 Numerical experiments for the virtual module set

To test our algorithm, we consider a pick and place task and use the virtual module set explained in Appendix A. We consider serially connected modular manipulators whose DOF varies from 2 to 5. The type of modules, the number of modules, and the combination of modules are the three main design parameters of our combinatorial problem.

In the given task, the robot is expected to take a predefined payload from a given initial position and place it at a given goal position in the shortest time without collision with any obstacles in the environment and the robot itself. Moreover, the robot should not exceed joint limits. Without loss of

Table 2.2: Comparison of the simulation time of the a 5-DOF module composition generated from the modules explained in Section 2.2.2 ($B - J1 - L1 - J1 - L2 - J1 - L3 - J1 - L2 - J1 - L3 - EE1$)

Test	Time [sec]
Reachability Test	0.007
Kinematic Test	0.143
Static Force / Torque Test	0.156
Path Planning in Obstacle-Free Environment	5.353
Path Planning in Obstacle-Laden Environment	5.871

generality, we test our algorithm by generating 10 random scenarios with the base module positioned at $p_b = (0; 0; 0)^T$ in all scenarios. For each scenario, the initial positions, goal positions, the obstacles' positions and their radii, and the number of the obstacles in the environment are generated using the uniform random number generator of MATLAB$^{\text{TM}}$ (with the *rand* function) in a 10x10x10 m environment. It should be noted that the base of the robot is placed at the center of the cube and all obstacles are static. All obstacles are considered as spheres and their radii are also randomly generated within the following limits: $r_o = [0.01 : 1]$ m. Besides, the number of obstacles for each scenario is also randomly generated to vary between 0 and 10 and the payload to vary between 1 kg and 7.5 kg.

All possible configurations from 2 DOF to 5 DOF are generated and 3108 different compositions are obtained following the predetermined structure in Section 2.3. Because the task requires a minimum of 3 DOF, 12 module compositions having 2 DOF are cancelled from all scenarios. As a first step, the reachability test detailed in Section 2.4.1 is applied to all module compositions, and module compositions whose lengths are greater than the distance between the base and the initial and goal positions ($\kappa_{\text{reachability}}$) is obtained. After that, the kinematic test detailed in Section 2.4.2 is applied to the remaining compositions and all compositions that can reach the initial and goal positions ($\kappa_{\text{kinematics}}$) are obtained for each scenario. Considering the given payload is in the $-z$-direction, a static test is applied as detailed in Section 2.4.3. To the compositions that can pass the static test (κ_{static}), collision and self-collision tests are applied to the initial and the goal positions as in Section 2.4.4. The path planning test is applied to the successful compositions from the previous test ($\kappa_{\text{collision}}$) as in Section 2.4.5. We implement the rapidly-exploring random trees-connect (RRT-connect) algorithm detailed in [83] as a path planning algorithm due to its simplicity and high success rate. Different from the standard RRT, two trees are simultaneously generated from the initial ($\mathbf{q}_s$) and goal positions ($\mathbf{q}_g$). A random point q_{rand} is generated and the closest points to the initial points of both trees are selected and called $q_{near,(\,\cdot\,)}$. New points $q_{new,(\,\cdot\,)}$ are simultaneously generated considering the given step size from both sides. Then, the algorithm is implemented as in standard RRT algorithms and the new generated points are checked for collisions. In case the generated point is collision free, it is added to the tree, otherwise another point is generated. When two trees intersect, a path is generated. We consider the step size as 0.1 and the number of maximum nodes is set to 100 for all scenarios. The RRT algorithm is applied a maximum of 5 times to each module composition and a module composition is discarded if a feasible path is not found. All compositions that can fulfil the given task and their corresponding paths are found after applying those tests ($\kappa_{\text{path planning}}$). It should be noted that the collision test is applied to all generated points on the path.

All successful compositions from the $\kappa_{\text{path planning}}$ test are examined in optimal trajectory planning as explained in Section 2.5. While implementing optimal trajectory planning, the positions generated by the path planning algorithm are set as an initial guess in the optimization to speed up the algorithm. The time is set 10 as an initial guess for all scenarios and the number of knots (generated points on the path) is set as the number of points defined from the path planning algorithm. Because the task is to be executed in a minimum amount of time, the objective function $L(\mathbf{x}_k(t), \mathbf{u}_k(t))$ is set as 1 where $(\,\cdot\,)_k$ indicates the k^{th} composition. Each joint modules' limits $\mathbf{q}_{k_{\text{min,max}}}$, $\dot{\mathbf{q}}_{k_{\text{min,max}}}$ and $\boldsymbol{\tau}_{k_{\text{min,max}}}$ define the lower and upper bounds of the optimization. The velocities of each joint at the initial and goal positions are set to zero. The multiple shooting method is applied to all remaining compositions and the trajectories are generated that integrate the equation $\dot{\mathbf{x}}_k(t) = g(\mathbf{x}_k(t), \mathbf{u}_k(t))$ for each knot starting from the initial position. The ODEs are solved for each of these points using the Runge-Kutta method from the library in [84].

The *fmincon* solver in MATLAB Optimization Toolbox$^{\text{TM}}$ is used and the *interior-point* method is selected. Tolerances for position and velocity are set to 0.001. All remaining compositions from the path planning test are considered in the optimization and the maximum iteration number and the maximum number of evaluations for each composition is set to 4 and 20000, respectively. The results for all compositions obtained at the end of the fourth iteration are compared to the best results obtained among the compositions. The composition is discarded in case its $c_{k,p}$ value calculated from (2.1) is above the predefined threshold that is set as 0.5 for the experiments.

This process is repeated in each iteration until the algorithm reaches the optimal cost value. In an average of all 10 scenarios, the total number of iterations is reduced by 56.1% and the total simulation time is reduced by 54.68% compared to the brute force algorithm. As an example, the task of moving the 5.1 kg payload from the initial position $p_i = (0.392, 0.097, 0.195)^T$ to the final position $p_d = (-0.371, 0.024, 0.266)$ in the environment with one obstacle is investigated and the position and velocity values of the best solution is given in Figure 2.2.

2.6.2 Numerical experiments for the extended SCHUNK LWA 4P robot

We also implement our algorithm on the second module set detailed in Appendix B and thus demonstrate the algorithm's results in a real environment. As opposed to the simulation experiment, only one piece from each module exists and the task is based on a real problem from the industry, which is defined as inserting insulation material into a car door within a minimum amount of time (see Figure 2.3).

The car door used in the experiment is provided by BMW and the given task is defined as following a predefined trajectory while pressing a mounted fabric on a door, which can be seen in Figure 2.3.

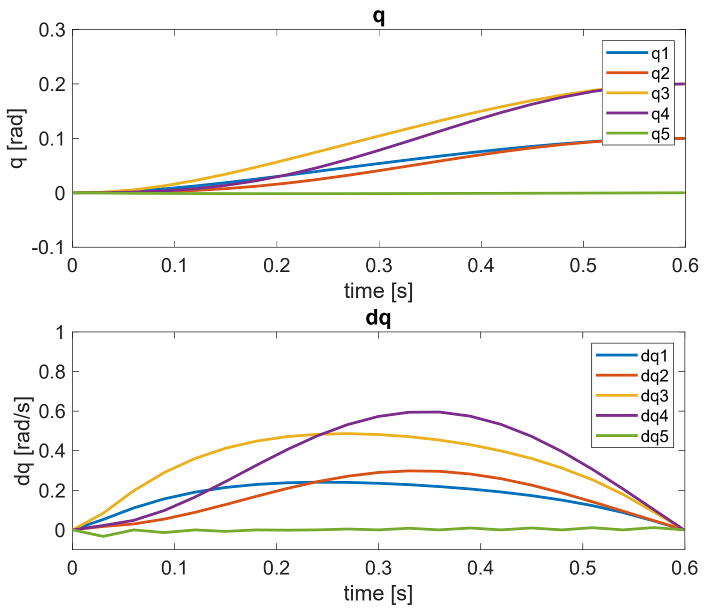


Figure 2.2: The position and velocity values of the cost-optimal solution for a given task.

The trajectory is obtained by Programming by Demonstration (PbD) [85] and a Vicon infrared motion tracking system is used to accurately track the position of the markers during the demonstration (see Figure 2.4(a)). More details on the generation of the points to be followed are detailed in [5]. Coordinates of all points that the robot has to follow are obtained by PbD and there are more than 15000 poses to be followed in sequence (see Figure 2.4(b)). However, implementing all tests on all obtained poses is time consuming and not practical. To decrease the computational work, we select 150 of the obtained points including all extreme poses in all directions. Because the positions to be followed are given in the task description, we only apply the following tests on these 150 predetermined poses: the reachability test, the kinematic test, the static test, and the self-collision test in sequence.

The production environment is generated in the laboratory and the base of the robot is placed in the position $p_b = (0; 0; 0)^T$. In total, 986409 different module combinations can be generated using the extended SCHUNK LWA 4P module set. To reduce the search space, we consider the following assumptions in addition to the ones given in Section 2.3: *i*) each module can be used once and *ii*) $L3$, $L4$, and $L5$ cannot be placed after the first joint module because it reduces robustness. Considering these assumptions and constraints, we reduce the number of possible compositions to 83. Afterwards, we implement the mentioned tests in sequence with the remaining module compositions. The desired task is completed in a 3D environment and requires at least 6 DOF, which means all joint modules

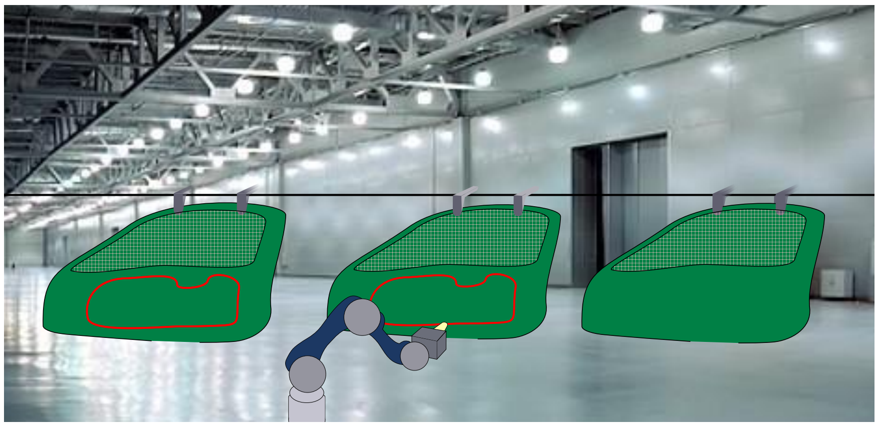
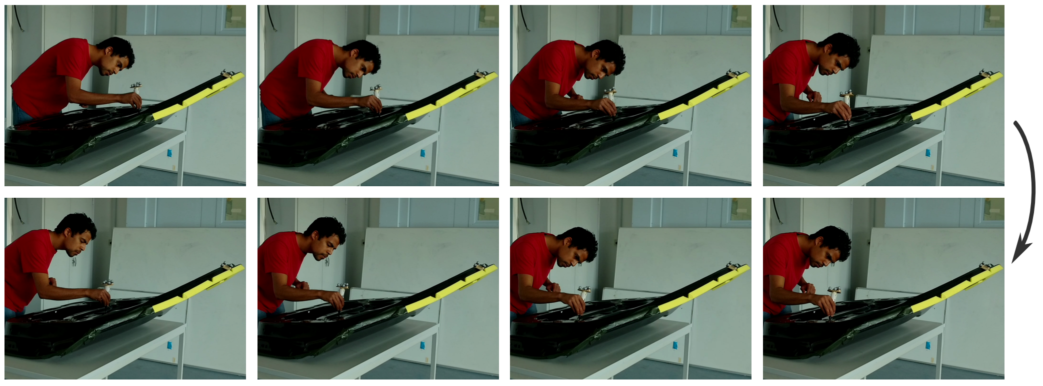


Figure 2.3: The illustration of the task given by the company [5].

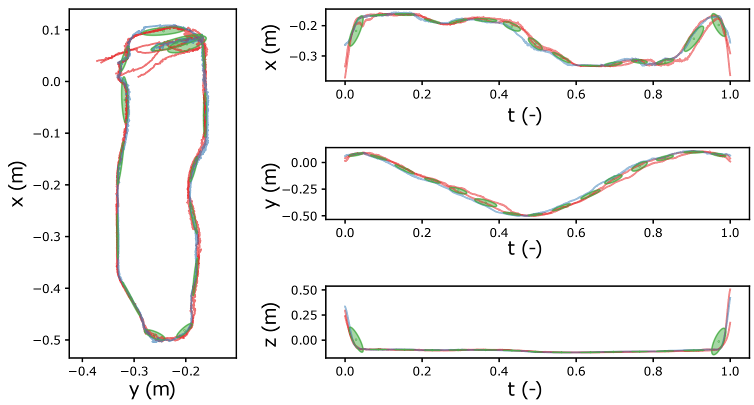
must be used. After implementing the reachability test, all 83 compositions followed the obtained positions. Then, we implement the kinematic and static tests. From 83 compositions, 70 are eliminated because they could not reach all given positions. The 9 remaining compositions could not pass the self-collision test leaving only 4 module compositions. For the remaining 4 compositions, namely $B - PB1 - L1 - PB2 - L3 - L2 - PB3 - E$, $B - PB1 - L1 - PB2 - L4 - L2 - PB3 - E$, $B-PB1-L1-PB2-L5-L2-PB3-E$, and $B-PB1-L1-L5-PB2-L2-PB3-E$, we follow their given trajectory point by point and calculate the minimum execution time for each one. We consider the maximum velocity limit for each joint (see Table B.1) during each interval while calculating the quickest execution time. As a result, the module composition $B-PB1-L1-PB2-L5-L2-PB3-E$ fulfills the task in the least amount of time. The optimal robot assembly can be seen in Figure 2.4(c). After the robot is assembled, the controller is generated as in [5] and the reproduction snapshots are shown in Figure 2.4(d).

2.7 Summary

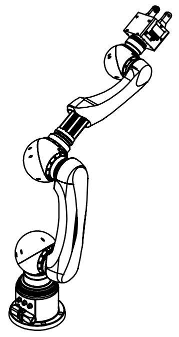
In this chapter, we review the proposed modular robot composition synthesis algorithms and propose a new, time-efficient composition synthesis algorithm that not only finds the cost-optimal solution but also shows how the task should be fulfilled. The proposed method is based on the elimination of unfeasible compositions starting from relatively easy and less time-consuming tests to more difficult and time-consuming ones. After obtaining all compositions that may fulfil the task, the proposed elimination method is implemented, which eliminates compositions with higher cost values than others in the early stages and applies the optimization to the more likely ones. As a result, the proposed algo-



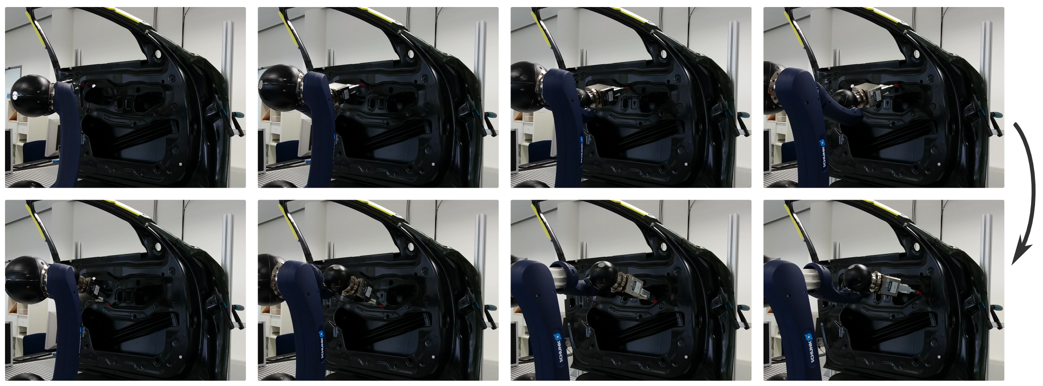
(a)



(b)



(c)



(d)

Figure 2.4: Overview of experimental results for inserting insulation material task with (a) demonstration of the task, (b) the task model obtained by PbD, (c) the optimal assembly, and (d) the snapshots of the reproduction [5].

rithm decreases the total computational time when compared to optimal trajectory planning tests on all remaining compositions. The results also prove that the proposed algorithms *i*) are computationally efficient, *ii*) applicable to different module types, *iii*) prevent the repetition of compositions, *iv*) consider not only robot kinematics but also robot dynamics if the task requires, and *v*) not only find the optimal composition but also compute how the task is fulfilled.

Chapter 3

Evolutionary Algorithm-Based Synthesis of Modular and Reconfigurable Robots for a Given Task

In this chapter, we present an evolutionary algorithm-based composition synthesis method for modular and reconfigurable robots that finds the cost-optimal module composition considering task-related objectives. As opposed to the previous chapter, an evolutionary algorithm is used to find the cost-optimal solution. Evolutionary algorithms are commonly used for many optimization problems because they provide faster solutions when compared to deterministic methods. Because the solution space of the problem is huge, we propose a method that guides the optimization algorithm while generating a set of intermediate points along the cost-optimal path. Only the end effector's position is taken into account while generating the set of intermediate points. The achievability of intermediate points is set as a parameter in the objective function. After that, the proposed two-step evolutionary algorithm-based method is implemented to find the set of best compositions. In the last step, the best solution is selected among the set of best compositions by applying an individual path planning algorithm to each of them. Simulations show that the presented algorithm provides a solution in a shorter simulation time than deterministic methods.

In addition to a literature review on the determination of task-based optimal module compositions explained in Section 2.1, a more detailed literature review on evolutionary algorithm-based optimal modular robot synthesis algorithms is given in Section 3.1. The proposed algorithm is outlined in

Section 3.2. The generation of a set of intermediate points is explained in Section 3.2.1 and the set of the best compositions is determined using evolutionary algorithms as detailed in Section 3.2.2. The selection of the cost-optimal solution is described in Section 3.2.3. Finally, the simulation results are shown in Section 3.3. The proposed method explained in this chapter was published in [40].

3.1 Introduction

Over the last few decades, optimization techniques based on evolutionary algorithms have gained considerable attention as an alternative to deterministic methods. Although they are not guaranteed to find a globally optimal solution, they are preferred due to offering solutions much faster than deterministic methods. Besides, it is easier to implement evolutionary algorithms when compared to deterministic methods. Numerous evolutionary optimization algorithms have been proposed so far, such as the genetic algorithm (GA) [86], memetic algorithm [87], ant colony algorithm [88], artificial bee colony algorithm [89], particle swarm optimization [90], and cuckoo search [91]. All these algorithms are biologically inspired and based on the principle of survival of the fittest. Each algorithm has its own advantages and disadvantages and more detail on each evolutionary algorithm can be found in [92].

These biology-based evolutionary algorithms have also been widely used to solve the task-optimal modular robot composition synthesis problem [64, 67, 93–96]. In particular, the genetic algorithm (GA) has been ubiquitous due to its simplicity and short computational times compared to the other methods [94]. For example, a GA-based optimal module composition synthesis algorithm is proposed in [64] in which a set of potential assemblies are encoded in a string. The proposed GA-based algorithm uses the reproduction, crossover, and mutation operators and finds the best solution for the given objective function, which checks the reachable number of given task points for each module composition. The same research group also introduces a minimized degrees of freedom (MDOF) approach in [93] to obtain the optimal module composition. The GA-based optimal module composition generation algorithm optimizes the given objective function that consists of reachability, joint range availability, manipulability, and feasibility of the mechanical structure. Another GA-based method that finds the best module composition for a given task is proposed in [72]. They first implement their hierarchical selection method to reduce their search space and then implement GA to select the best module composition. The authors also represent potential module compositions as chromosomes and implement GA to obtain the module composition providing the best fitness value among them.

Another automated task-based composition synthesis algorithm is presented in [50] and based on the combination of GA and an object-oriented structure. Each module is defined with information-containing parameters and are represented by two groups: binary values and real values. To generate new robots, they use different types of crossover and mutation operators for each group, which enables the separate exchange of modules and their parameters.

Besides implementing standard GA, some researches implement GA in several steps to find the cost-optimal module composition. The authors in [97] consider coupled joint and link modules as sub-assemblies and implement two-level GA to obtain the cost-optimal module composition. Each sub-assembly is represented by a chromosome and encoded in binary. Each chromosome has the following information: the joint type, the orientation of the joint, the link type, the link length, and the orientation of the link module. However, this is one of the drawbacks of this method since robots that only differ in the type of joints have the same structure in a chromosome. Furthermore, the posture of assemblies is represented with real coding because of the wide range of joint angles. The proposed method in [97] divides the objective function into three categories: hard requirements, soft requirements, and hard-soft requirements. While hard requirements must be achieved, the solutions closest to the soft ones are the better solutions and the hard-soft requirements must be given in predetermined thresholds. The proposed two-level algorithm first generates topologies and then it calculates the joint angles in the second level.

Another two-level GA-based composition synthesis algorithm for fault-tolerant modular robots is introduced in [26]. Their algorithm finds a set of robot compositions that fulfil the task considering a fault-tolerant workspace reachability in the first step. After that, the algorithm selects the optimal composition considering the failure possibility of each joint and joint limits among the successful compositions remaining from the first step.

The two-level genetic algorithm (TGA) detailed in [67] also determines the optimal module composition considering the specifications of a given task. The algorithm first finds a set of the best topologies in the upper-level GA and then it solves inverse kinematics of the obtained robot compositions in the lower-level GA. In this method, all information about a robot's composition is defined on a single binary chromosome that leads to a long-string large chromosome. Thus, the method is inefficient because changing a small part of the chromosome using GA operators affects a large part of the whole structure. As a consequence, the computational time of the process increases. To overcome this problem, the same authors propose a multi-chromosome algorithm (MEA) in [94]. The MEA differs from their previous work in the following way: *i*) the MEA has one evolution loop, which makes it faster than TGA, *ii*) the algorithm divides a chromosome into several chromosomes and this separation enables each chromosome to obtain remarkably related information without any effects from the global crossover, and *iii*) the algorithm divides the objective function into two sub-objective functions: topology and configuration. Thus, the global objective function is the product of these two sub-objective functions and this approach allows the user to optimize configurations while fixing module compositions. Their experiments show that the MEA is 100 times more computationally efficient than the TGA [94]. In addition to previously explained methods, another algorithm introduced in [98] also uses a two-step GA. As opposed to other studies, the authors consider adjustable link modules, and they aim to determine optimal lengths of the links. In this method, the configuration is determined in the first step of the algorithm considering robot kinematics and the second step is to find the optimal link length for the given task.

In addition to either implementing existing evolutionary algorithms as stand-alone or in sequence, a combination of the two has also been used to solve the task-based optimal module composition problem. An adapted simulated annealing and genetic algorithm (ASAGA)-based optimal composition synthesis algorithm for parallel modular robots is proposed in [96]. They consider the sum of the reachability of task points and the differences between joint angles in sequential task points as the objective function. In this algorithm, the mutation operator is replaced with a simulated annealing (SA) algorithm. This change makes the algorithm faster and increases its reliability.

Similar to [96], the authors in [49] also use a genetic-simulated annealing algorithm to solve the TBOD problem. The proposed algorithm combines the advantages of the searching capability of the GA and the superior local search capability of the simulated annealing algorithm. As a result, the proposed algorithm increases the accuracy and success rate of the search. As an objective function, the authors consider accessibility of the positions and orientations.

The novel task-based optimal design approach detailed in this chapter differs from the above-mentioned algorithm as the task is considered during the optimization. The search space is reduced considerably when taking the hard requirements of the task into account and thus the GA-based method is only applied to most likely compositions. The main novelty of the proposed method is the inclusion of the task-related part considering the previously defined guidance points in the objective function. The implementation of the optimization algorithm generates a set of best assemblies and then individual task planning is applied to them. After this, the composition that gives the best result is selected as the task-optimal solution.

3.2 Proposed Algorithm

As mentioned in previous sections, the search space must be narrowed as much as possible to obtain the cost-optimal solution in the shortest computational time. The same assumptions detailed in Section 2.3 are followed while generating module compositions. Instead of iterating all possible module combinations as in the previous chapter, an evolutionary algorithm-based composition synthesis method is

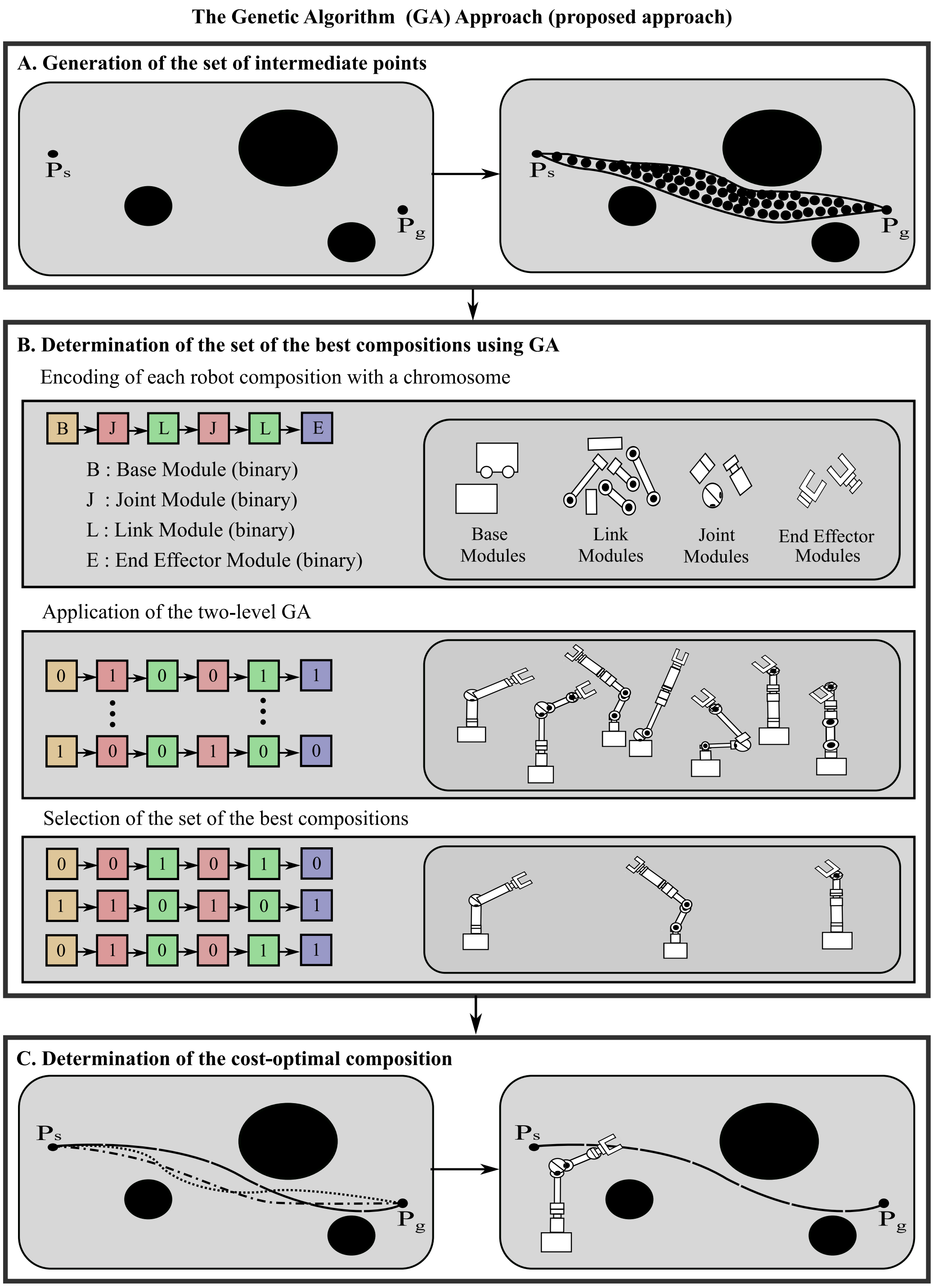

The Genetic Algorithm (GA) Approach (proposed approach)
A. Generation of the set of intermediate points
Ps
Pg
B. Determination of the set of the best compositions using GA
Encoding of each robot composition with a chromosome
B J L J L E
B : Base Module (binary)
J : Joint Module (binary)
L : Link Module (binary)
E : End Effector Module (binary)
Base Modules
Link Modules
Joint Modules
End Effector Modules
Application of the two-level GA
0 1 0 0 1 1
1 0 0 1 0 0
Selection of the set of the best compositions
0 0 1 0 1 0
1 1 0 1 0 1
0 1 0 0 1 1
C. Determination of the cost-optimal composition
Ps
Pg

Figure 3.1: The proposed composition synthesis method for modular robots

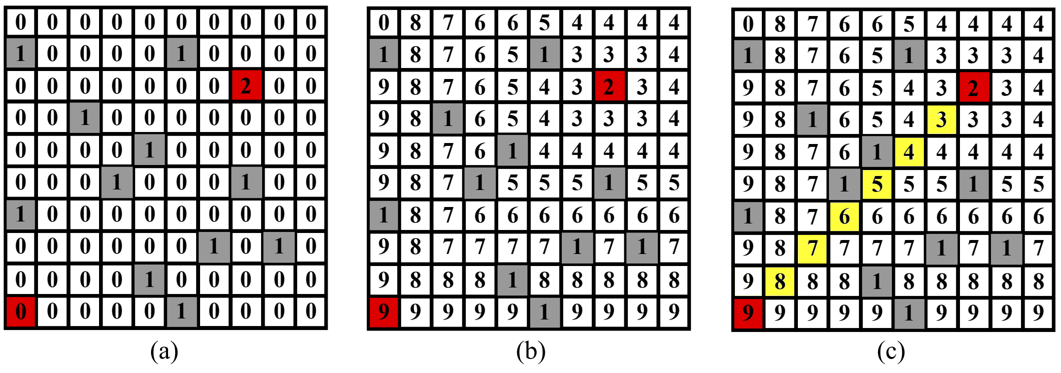


Figure 3.2: Wavefront algorithm (a) numeration of obstacles and the starting point (b) numeration of all cells, and (c) the set of intermediate points

proposed in this chapter. Considering the task requirements, the required minimum DOF, and the maximum DOF are defined by the user and a set of module compositions are randomly generated. As seen from the schematic representation of the presented method in Figure 3.1, it can be divided into three steps: *1*) the generation of the set of the guidance points, *2*) the determination of the best compositions using evolutionary algorithms with the help of the additional objectives, and *3*) the determination of the cost-optimal composition considering the given objective function.

3.2.1 Generation of the set of guidance points

As a first step of the algorithm, a set of intermediate points that are in the shortest geometric path between the initial position and the goal position are generated. Because each module composition has different solutions for a given task, a geometric path between the initial position and the goal position is created independently from the module composition. To achieve this, any geometric path planning method in the literature can be implemented (the most well-known geometric path planning algorithms can be found in [81]). We apply the wavefront algorithm in our simulations due to its simplicity [78]. The main idea behind the wavefront algorithm is to divide the environment into *cells* with a pre-defined *cell size*. All *cells* are initially numbered as 0. Then, all *cells* that contain obstacles are numbered by 1 and the goal position is numbered by 2 as in Figure 3.2(a). Starting from the goal position, all obstacle-free *cells* are numbered considering the distance to the goal position as in Figure 3.2(b). The path is generated starting from the goal position following the values in an increasing order and finishing at the initial position. Yellow *cells* in Figure 3.2(c) indicate the set of intermediate points to fulfil the given task with the shortest path.

3.2.2 Determination of the best compositions using evolutionary algorithms

As a second step, the set of the best compositions are defined considering the task-related constraints in the objective function. Any evolutionary algorithm can be implemented in this step. In the proposed method, the genetic algorithm (GA) is selected to solve the task-based cost-optimal composition synthesis problem as it is appropriate for multidimensional and nonlinear problems and it gives decent results for problems with a large search space [92].

Genetic algorithms are heuristic algorithms that are a subclass of evolutionary algorithms inspired by the principle of evolution proposed by Darwin [86]. The GA is based on a population of *chromosomes* in which potential solutions to the problem are encoded. Each *chromosome* has a *fitness* value that measures how good the solution is considering the given objective function. A *chromosome* (shown in a red frame in Figure 3.3) consists of a sequence of *genes* (shown in a blue frame in Figure 3.3) that are represented by a unique name (chromosomes can be seen in Figure 3.3 with the names Ai where i is a number). A *gene* can be encoded as binary, alphabetical, or floating-point values. Starting from an arbitrarily generated population of *chromosomes* (i.e. generation N in Figure 3.3), a process begins based on the selection of the best *fitness* values and then the successful ones are considered in the next genera-

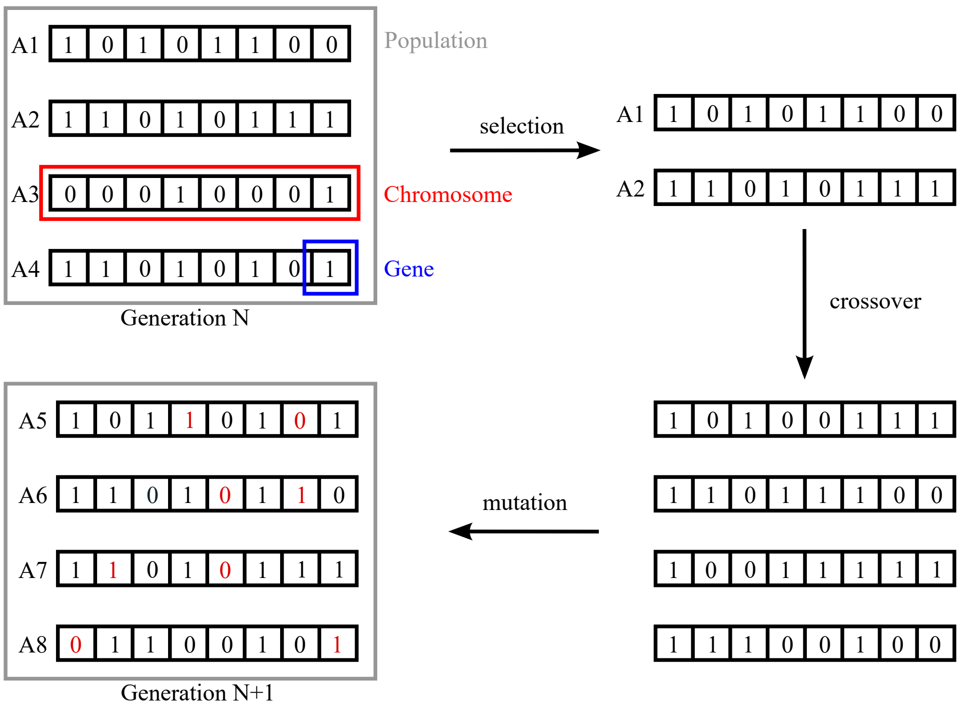


Figure 3.3: Genetic algorithm operations

tion. This evaluation is done by using the following three operators: *selection*, *crossover*, and *mutation*. The schematic representation of these operators can be seen in Figure 3.3. The *selection* operator of a GA guides the evolution process using the *fitness* values of the *chromosomes*. For the *selection* process, there are several algorithms to choose from such as roulette, tournament selection, random stochastic selection, etc. (for more details see [99]). As seen in Figure 3.3, chromosomes A1 and A2 are selected by the selection operator. The *crossover* operator creates *offspring chromosomes* by exchanging the genes from selected parents. Several algorithms for crossover operation exist in the literature such as one-point crossover, two-point crossover, multi-point crossover, cut and splice, three-parent crossover, etc. and their details can be found in [99]. Chromosomes in the bottom-right of Figure 3.3 are generated by implementing the crossover operator to the selected A1 and A2 chromosomes. To increase the diversity within a population, the *mutation* operator is used, which introduces random modifications to offspring chromosomes depending on the mutation rate. With the implementation of the mutation operator, chromosomes A5, A6, A7, and A8 in Figure 3.3 are generated and the genes changed by the mutation operator are seen in red in generation N+1.

Besides these four parameters, the *population size* and the *generation size* are also important in GAs. The *population size* gives the number of chromosomes in one generation, which is 4 for the example in Figure 3.3. The *generation size* gives the number of the generations in the algorithm. These two parameters are related to better fitness values. In the first generation, the fitness value generally shows higher improvements and then the changes decrease as the optimum solution is approached.

In each generation, random chromosomes are selected as parents and they produce the next generations of chromosomes. For each chromosome, the fitness value is calculated in every generation and this process is repeated until there is either no change in the population of fitness values, the maximum number of generations has been reached, or a certain time has elapsed [100].

In our composition synthesis problem, each module is represented by a gene and each assembly is represented by a chromosome. Modules are grouped as base modules, joint modules, link modules, and end effector modules and each module is encoded with its own binary numbers within itself. Then, compositions are randomly generated following the previously defined robot structure in Section 3.2. A representation of a module composition as a chromosome is shown in Figure 3.4.

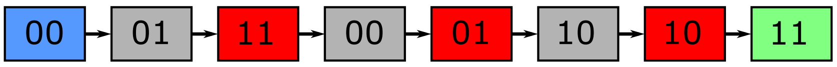


Figure 3.4: Representation of a module composition where blue box, gray boxes, red boxes and green box indicates base module, joint modules, link modules and end effector modules in the structure, respectively. The binary numbers in each group of module represents different modules.

The proposed optimization algorithm consists of two levels in a sequence to make the synthesis process faster. In the first level, task definitions (*TaskDef*) and a user-defined number of populations (*NumOfPop*) are given as an input. As seen in Algorithm 1, possible module compositions are generated with the *generateComps()* function following the rules explained in Section 3.2. It should be noted that the main purpose behind the implementation of this algorithm is to generate the initial generation and to discard unfeasible module compositions during the early stages. The first part of the algorithm checks whether the module composition can meet the hard constraints, which are the constraints that any module composition must achieve. For this algorithm, hard constraints are defined as the ability of the module composition to reach the given positions in the task definition such as the initial position, the goal position, and intermediate positions and whether or not there are any intermediate positions in the task description. To check whether the module composition meets the hard constraints, the following tests are implemented via the *tests()* function in Algorithm 1: the kinematic test (in Chapter 2.4.2), the static test (in Chapter 2.4.3) and the collision test (in Chapter 2.4.4). With these simple tests, the search space decreases as the unfeasible compositions are eliminated before applying the GA. In case the given positions are not reachable by a generated module composition, the module composition is penalized meaning that it is not considered for the next generation or in the second level of the algorithm. The penalized compositions can be summarized as compositions in *CheckedComps* but not in *PossibleComps* of Algorithm 1.

The output of the first level of the algorithm, *PossibleComps*, is a set of possible module compositions that will be considered as the first generation of the second level of the algorithm as in Algorithm 2.

Algorithm 1 $FirstLevel()$

```
1: Input: $TaskDef$, $NumOfPop$, $tests()$, $generateComps()$
2: Output: $PossibleComps$, $CheckedComps$
3: $PossibleComps := []$
4: $CheckedComps := []$
5: while $size(PossibleComps, 1) = NumOfPop$ do
6:     $Comp \leftarrow generateComps()$
7:     if $Comp \in CheckedComps$ then
8:         $go\ step\ 6$
9:     else
10:        $CheckedComps := [CheckedComps;\ Comp]$
11:        $Flag \leftarrow tests(Comp, TaskDef)$
12:        if $Flag$ then
13:            $PossibleComps := [PossibleComps;\ Comp]$
14:        end if
15:    end if
16: end while
```

Algorithm 2 $SecondLevel()$

```
1: Input: $PossibleComps$, $CheckedComps$, $TaskDef$, $targets$, $selectionPop$, $crossoverPop$, $mutationPop$, $genSize$, $terminationFlag$, $evalsol()$, $sort()$, $sortedPopOfSols()$, $ismember()$, $append()$, $bestSolOf()$, $encodedSol()$, $mutate()$, $crossover()$
2: Output: $ASetOfBestAssemblies$
3: while i=1:$genSize$ or $terminationFlag$ do
4:     $PossibleCompEvals \leftarrow evalsol(PossibleComps,\ TaskDef,\ targets)$
5:     $SortedPopOfSols \leftarrow sort(PossibleComps,\ PossibleCompEvals)$
6:     $j \leftarrow 1$
7:     $k \leftarrow 1$
8:     while $j \leq selectionPop$ do
9:         $newCandidate \leftarrow SortedPopOfSols(k)$
10:        if $ismember(newCandidate, newGeneration)$ then
11:            $k \leftarrow k + 1$
12:        else
13:            $newGeneration \leftarrow append(newCandidate)$
14:            $k \leftarrow k + 1$
15:            $j \leftarrow j + 1$
16:        end if
17:    end while
18:    $j \leftarrow 1$
19:    $k \leftarrow 1$
20:    while $j \leq crossoverPop$ do
21:        $parent1 \leftarrow encodedSol(SortedPopOfSols(k))$
22:        $parent2 \leftarrow encodedSol(SortedPopOfSols(k + 1))$
23:        $[offSpring1, offspring2] \leftarrow crossover(parent1, parent2)$
24:        $newGeneration \leftarrow append(bestSolOf(parent1, parent2, offSpring1, offSpring2))$
25:        $k \leftarrow k + 2$
26:        $j \leftarrow j + 1$
27:    end while
28:    $j \leftarrow 1$
29:    $k \leftarrow 0$
30:    $bool \leftarrow 1$
31:    while $j \leq mutationPop$ do
32:        if $bool$ then
33:            $newCandidate \leftarrow SortedPopOfSols(popSize - k)$
34:            $newCandidate \leftarrow mutate(newCandidate, mutationRate1)$
35:            if $newCandidate \in CheckedComps$ then
36:                $go:\ step:\ 34$
37:            end if
38:        else
39:            $newCandidate \leftarrow SortedPopOfSols(k)$
40:            $newCandidate \leftarrow mutate(newCandidate, mutationRate2)$
41:            $newCandidate \leftarrow mutate(newCandidate, mutationRate1)$
42:            if $newCandidate \in CheckedComps$ then
43:                $go:\ step:\ 41$
44:            end if
45:        end if
46:        $bool \leftarrow !bool$
```

```
47:         $k \leftarrow k + 1$
48:         $j \leftarrow j + 1$
49:         $newGeneration \leftarrow append(bestSolOf(evalsol(newCandidate(newCandidate,$
50:         $TaskDef, targets))))$
51:     **end while**
52:     $PossibleComps := newGeneration$
53: **end while**
54: $ASetOfBestAssemblies := PossibleComps$
```

In this part of the algorithm, first fitness values for all compositions generated in the first level are calculated using the *evalSol()* function in Algorithm 2. While calculating these values, the objective function seen in (3.1) is used (the objective function is given in *TaskDef* in Algorithm 2). The set of guidance points *targets* are also considered in the objective function. The obtained fitness values for each module composition are stored and sorted by the *sort()* function. The sorted compositions and their obtained fitness values are placed in *SortedPopOfSols*. The first *selectionPop* individuals (i.e., the user-defined number of individuals to be selected for the next generation) among *SortedPopOfSols* are selected. The algorithm checks whether they are in sets of already assigned or checked module compositions with the *ismember()* function. In case they are not, the selected individuals are assigned to the *newGeneration* with the *append()* function. Then, the crossover operator (*crossover()*) is implemented to a user-defined number of individuals (*crossoverPop*) using the uniform crossover technique. In this technique, two best individuals are selected as parents since the expectation of better offspring is higher for elite parents [99]. With the *encodedSol()* function, the binary representation of chromosomes are encoded. After offspring are generated (see *offspring1* and *offspring2*), their fitness values are obtained and compared with their parents using the *bestSolOf()* function and the best two fitness values out of the four chromosomes (*parent1*, *parent2*, *offspring1* and *offspring2*) are selected for the next generation. Then, the mutation operator (*mutate()*) is applied to a user-defined number of individuals (*mutationPop*) while generating a random change in random bits considering the user-defined mutation rate. As seen in the Algorithm 2, the mutation rate is different for the best and the worst candidates. The variable *mutationRate1* is higher than *mutationRate2* as a higher mutation rate enables larger changes in a chromosome, which increases the possibility of generating better candidates. It should be noted that ! means negation in Algorithm 2.

For each generation, the generated module compositions are compared with their ancestors and selected only if their fitness values are better than their ancestors. The fitness values of chromosomes generated via operators are calculated only if they are not in the set called *CheckedComps* and they fulfill the hard requirements. This process continues until the algorithm reaches the maximum number of generations (*genSize*) or there is no improvement in the population (*terminationFlag*).

The objective function considers soft constraints, which means that their fulfilment is not essential, but better results are obtained if the robot meets them. The objective function is defined as follows:

$$g = w_1 \cdot e^{-(k_1 \cdot R + k_2 \cdot L + k_3 \cdot A + k_4 \cdot D + k_5 \cdot I + k_6 \cdot V + k_7 \cdot \frac{1}{1+O})} + w_2 \cdot P \tag{3.1}$$

where k_i are the weighting values for the for the first part of the algorithm and w_i are the weighting values of each part of the objective function. In the first part, the algorithm considers criteria related to how the robot fulfils the given task such as reachability (*R*), linear distance (*L*), angular distance (*A*), dexterity (*D*), involved modules (*I*), joint value differences between the initial and the goal positions (*V*), and obstacle proximity (*O*). The criterion in the second part of the objective function considers the achievability of intermediate points (*P*) obtained from Section 3.2.1. It should be noted that the first part of the algorithm is taken from [67] and thus we use the normalized values of each sub-criterion.

The details of the criteria in the first part of the objective function are given as follows:

1. *Reachability (R):* Reachability is the ability to reach the given positions only considering the robot's length. The variable d_{d2b} indicates the distance between a given position to be followed ($\mathbf{p}_d$) and the base location of the robot ($\mathbf{p}_b$) and calculated as $d_{d2b} = \|\mathbf{p}_d - \mathbf{p}_b\|$. The total length of the robot is calculated by the sum of the length of each component where l_i is the length of the i^{th} component and n is the total number of DOF.

$$R = \frac{d_{d2b} - \sum_{i=1}^{2n} l_i}{d_{d2b}} \tag{3.2}$$

2. *Linear distance (L):* Linear distance is the normalized distance between the end effector's position $\mathbf{p}_e$ and the desired position $\mathbf{p}_d$ that gives the user how the robot far from the desired position in the work space.

$$L = \frac{\|\mathbf{p}_d - \mathbf{p}_e\|}{\|\mathbf{p}_d\|} \tag{3.3}$$

3. *Angular distance (A):* Angular distance is the normalized distance between the orientation of the end effector $\mathbf{r}_e$ and the desired orientation $\mathbf{r}_d$, which gives the user how the robot is far from the desired position in the joint space.

$$A = \frac{\|\mathbf{r}_d - \mathbf{r}_e\|}{\|\mathbf{r}_d\|} \tag{3.4}$$

4. *Dexterity (D):* Dexterity is the ability of a robot manipulator to easily move and apply force in arbitrary directions. The Yoshikawa manipulability index y [101] is used to define dexterity and

it is calculated as

$$y = \sqrt{det(\mathbf{J} \cdot \mathbf{J}^T)}, \tag{3.5}$$

where **J** is the Jacobian matrix and $(J)^T$ is its transposed matrix. Then, dexterity is calculated by

$$D = \frac{1}{1+y}. \tag{3.6}$$

5. *Involved modules (I):* The mass and complexity of the robot structure is measured by involved modules that are calculated as in

$$I = \frac{\sum_{i=1}^{n} l_i + \zeta \cdot l_{max,i}}{d_{e2b}}. \tag{3.7}$$

where the variable l_i is the length of the i^{th} module in the composition and ζ is a parameter depending on the joint type with a value of 0 for link modules and revolute joints and 1 for prismatic joints. The variable $l_{max,i}$ is the maximum limit of the i^{th} joint and d_{e2b} is the distance between the end effector and the base module. This parameter is used to minimize the complexity and the total mass of the robot. A small value of this parameter gives better solutions.

6. *Joint value differences between the initial and the goal positions (V):* Joint value differences are the total angular distance between the initial and the goal positions of all joints. It is calculated as the sum of all the joint position differences for each joint as in

$$V = \sum_{i=1}^{n} |q_i(0) - q_i(t_f)|. \tag{3.8}$$

Smaller values of V indicate that the robot is closer to the desired position.

7. *Obstacle proximity (O):* Obstacle proximity is the closest distance between the robot and an obstacle in the environment. The larger the distance between the robot and obstacle, the safer it is. The variable d_{i2O_j} indicates the distance between the i^{th} component and the j^{th} obstacle. The variable r_{O_j} and the variable r_i indicate the radius of the j^{th} obstacle in the environment and the i^{th} component of the robot, respectively.

$$O = \underset{i \in 1 \cdots 2n, j \in 1 \cdots s}{argmin} (d_{i2O_j} - (r_{O_j} + r_i)) \tag{3.9}$$

The optimization is performed in the second step and it is expected to minimize the objective function, which is the weighted sum of these seven criteria.

The first part of the objective function is very similar to the method proposed in [67]. The criteria R, L, A, D, and I in this part of the equation are also considered in [67] and the criteria O is modified from [67]. The criteria V is added in our approach. The proposed approach in [67] only considers the initial and goal positions and not how the robot moves between them. The main novelty of the proposed method is the second part of the equation (3.1), which enables the selection of the compositions taking the task-related objectives into account. With this additional criteria, we consider the ability of the robot to reach all possible intermediate points in the optimization. These points are the common points for all possible compositions that are close to the optimal solution and the higher the possibility of reaching these points, the higher the possibility of the composition being the optimal solution. Fitness values of compositions are obtained using equation (3.1). The algorithm runs until it reaches one of the stopping criteria, which are *i*) reaching the maximum number of generation, *ii*) a certain time has elapsed, and *iii*) no change in the population fitness value.

3.2.3 Determination of the cost-optimal composition

The set of module compositions that give the best fitness values (*ASetOfBestAssemblies*) are obtained from a common geometric path. While finding the common geometric path, we only consider the end effector's positions and do not take the movement of each joint into account. Because the geometric path is only used as guidance for the algorithm, an additional path planning algorithm that is specific to each module composition must be individually implemented to all compositions among the set. This additional step generates finer paths for all selected module compositions. Any path planning algorithm in the literature can be implemented to find finer paths. Many widely used path planning algorithms can be found in [81]. Considering the desired cost function, cost is calculated for the module compositions and the cost-optimal solution is selected among them as in (1.3).

3.3 Numerical Experiments

We implemented our algorithm in Matlab R2019b running on the Intel® Core™ i7 processor with 2.30 GHz and 16 GB of memory. The virtual module set detailed in Appendix A is used in the simulations and it is encoded as follows:

i) the base module (B) is coded as 0,

ii) revolute joint (J_R) is coded as 0 and prismatic joint (J_P) is coded as 1, *iii*) link modules with $\alpha = 90^\circ$ whose length along y-direction of the previous coordinate system (L_1), $\alpha = 0^\circ$ whose length is along x-direction of the previous coordinate system (L_2) and $\alpha = 90^\circ$ whose length along the z-direction (L_3) are encoded as 00, 01 and 10, respectively, and

iv) end effectors E_R and E_P are encoded as 0 and 1, respectively.

The base module is positioned at point $\mathbf{p}_b = (0, 0, 0)^T$ and only the robot compositions with 5-DOF are generated using the assumptions given in Section 3.2. In total, 2592 different robot compositions can be generated following those criteria.

The task is defined by picking a randomly generated payload that varies between 0.1 kg and 5 kg and moving it from the initial position $\mathbf{p}_s$ to the goal position $\mathbf{p}_g$ along the shortest possible route without collision with any obstacles in the environment or the robot itself. It should be also noted that the robot cannot violate its joint limits during the execution. We consider the environment as a cube whose edges are 7 m in length and centered at the position $\mathbf{p}_b$. In total, 20 different scenarios were randomly generated using a standard uniform distribution within the aforementioned environment. In each scenario, we consider spherical obstacles with radii varying between $r_0 = [0.05, 0.4]$ m and the number of obstacles varying between 1 to 6. While generating these scenarios, we check whether the generated initial positions and goal positions are within the obstacles and generate new ones in case they are.

As a first step, we generate the shortest geometric path for all scenarios as explained in Section 3.2.1. The step size for the wavefront algorithm is set to 0.1. Cells with an obstacle are numbered with 1 and the numbering is implemented starting from the initial position. The shortest geometric paths are generated for all 20 scenarios. All points on the shortest geometric paths in each cell are considered as possible intermediate points.

The parameters for the GA are tested while fixing *mutation rate*, *selection rate* and *crossover rate* as in Table 3.1 with different values for different groups. The variables *population size* and *generation size* are closely related to each other in terms of the exploration and diversification of the solutions. Experiments are done in order to determine the optimal population size and generation size by comparing the best fitness value obtained as well as the required computation time. We tested values for different scenarios and found all tests provided similar results. The results of simulations with varying *population size* and *generation size* for Scenario 1 demonstrate that the best fitness value remains the same (Figure 3.5(a)) even as the simulation time increases (Figure 3.5(b)) after a certain value of each parameter. Considering these results, the *population size* and *generation size* are chosen as 100 and 150, respectively.

Table 3.1: The GA parameters.

	Group I	Group II
the *mutation* rate	60%	30%
the *selection* rate	10%	10%
the *crossover* rate	30%	60%

Two different groups are employed during the execution of the GA. The first group is used in the first few steps to prioritize the diversity of the population, so the *mutation* rate is selected to be a high number. This process prevents the algorithm from being stuck at a local minima. The second group is used after a few generations and its purpose is to place more importance on the best individuals to increase exploitation.

While defining the weighting values of the objective function, we treat all variables as equally important. Because each criterion is equally important, they receive the same priority (k_i) in the first part of (3.1) as detailed in Section 3.2.2 with each k_i value set as $1/7$.

The same priority is also given to each criterion of the objective function with both w_1 and w_2 in (3.1) set to 0.5. The best 15 individuals reached in the last generation of the GA (called *the set of the best compositions*) are considered for each scenario and the method explained in Section 3.2.3 is applied only the set of the best compositions. We implemented RRT as detailed in [81] to generate finer paths.

The simulation results for each scenario are given in Table 3.2 in which the success ratio (*success ratio*) is defined as the ratio of the number of compositions that can find a feasible path to the number of sets of the considered compositions (all compositions for the method in [67] and the set of best compositions for the proposed method). The maximum step size and the maximum number of points was set to 0.1 and 100, respectively. We ran the RRT algorithm 5 times for each module composition

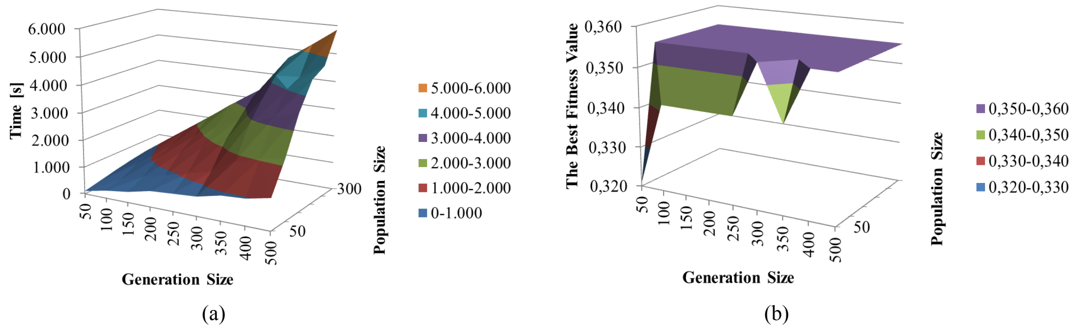


Figure 3.5: The simulation results for different population size and generation size: (a) simulation time and (b) the best fitness value for each simulation

Table 3.2: Experimental Results for each Scenario

	Proposed Method		Method in [67]		Method in [24]	
Scenario	*success ratio*	*comp. time ratio*	*success ratio*	*comp. time ratio*	*number of $\mathbf{p}_s$ and $\mathbf{p}_g$ feasible compositions*	*ratio of feasible compositions*
1	0.81	0.21	0.26	0.31	134	0.77
2	1	0.45	0.67	0.98	26	0.91
3	0.74	0.22	0.12	0.60	53	0.91
4	0.83	0.48	0.49	1.18	24	0.85
5	N/A	0.98	N/A	1.24	0	N/A
6	1	0.68	0.50	1.24	9	0.89
7	1	0.14	0.90	0.11	171	0.89
8	0.96	0.46	0.30	0.66	74	0.92
9	0.93	0.24	0.78	0.28	92	0.90
10	0.89	0.21	0.66	0.36	120	0.84
11	0.95	0.37	0.59	0.36	119	0.92
12	0.88	0.74	0.02	1.58	12	0.83
13	0.95	0.27	0.43	0.28	107	0.93
14	0.89	0.26	0.51	0.52	13	0.77
15	0.72	0.43	0.62	0.64	76	0.88
16	0.56	0.28	0.41	0.33	162	0.85
17	1	0.35	0.77	0.21	232	0.93
18	0.95	0.35	0.81	0.44	97	0.91
19	1	0.23	0.80	0.18	207	0.98
20	0.8	0.50	0.65	0.25	92	0.94

and considered their average as *success ratio* because the RRT algorithm gives different results for each run. It was found that 88.71% compositions of the set were able to find a collision-free path on average (see Table 3.2).

To see the advantages of our algorithm, we compare our proposed method with the method proposed in [67]. To make a fair comparison, we also selected sets of the best compositions obtained from the algorithm and implement the same path planning methods with the same parameters. We compare the results obtained from the RRT algorithm and an average of 54.6% compositions from the set were able to find a collision-free path. However, the best compositions obtained from [67] could not find a solution for 7 scenarios since the robots collided with obstacles in the environment while performing the task. Even though the computational time of [67] is shorter than the proposed method, the simulations show that the best compositions, only taking the initial and goal positions into account (called as $\mathbf{p}_s$ *and* $\mathbf{p}_g$

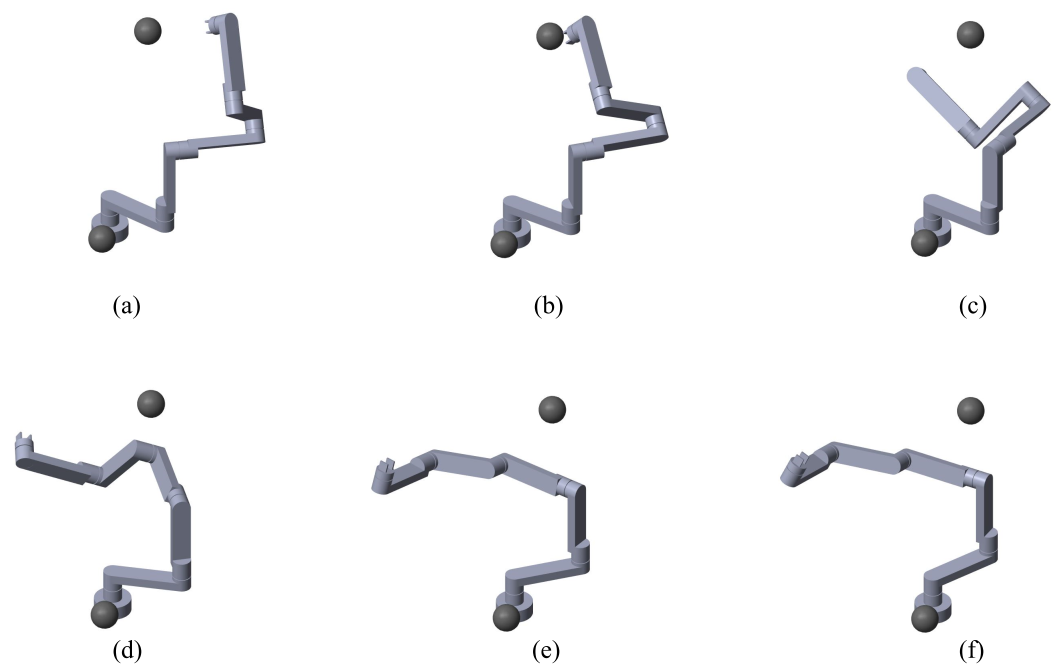


Figure 3.6: Screen shots for the composition $B-J1-L2-J1-L1-J1-L3-J1-L2-J1-L3-EE1$ for the scenario. $t_f = 1$ and (a) $t = 0$, (b) $t = 0.2t_f$, (c) $t = 0.4t_f$, (d) $t = 0.6t_f$, (e) $t = 0.8t_f$, (f) $t = t_f$

feasible compositions), are not the cost-optimal composition when the full task is considered. Moreover, one cannot guarantee that $\mathbf{p}_s$ and $\mathbf{p}_g$ feasible compositions are able to perform the task.

We also compare the proposed method with our previously published method detailed in [24] and compare the total computation time. To make a better comparison, we implement the same path planning algorithms on the remaining compositions, calculate the ratio of the simulation time for each scenario, and show the computation time ratio (*comp. time ratio*) in Table 3.2. Compositions that can find a collision-free path between $\mathbf{p}_s$ and $\mathbf{p}_g$ are called as *feasible compositions*. The ratio of $\mathbf{p}_s$ and $\mathbf{p}_g$ feasible compositions and feasible compositions (*ratio of feasible compositions*) are also given in Table 3.2. The simulation results show that the proposed algorithm is computationally more efficient when there are multiple solutions for the given task. The computational efficiency increases with the number of feasible compositions of the initial and goal positions. Furthermore, simulations show that compositions that can reach the $\mathbf{p}_s$ and $\mathbf{p}_g$ may not find a collision-free path between them.

The screenshots of the best composition generated by using the visualization tool explained in Appendix C for the following scenario are given in Figure 3.6. In this scenario, the robot is moving the 5 kg payload from $\mathbf{p}_s = (1.5, 1.5, 2)^T$ to $\mathbf{p}_g = (-2, -1.2, 0.6)^T$ with two obstacles defined in the environment as $\mathbf{p}_{o,1} = (1.5, -1.3, 1.1)^T$, $r_{o,1} = 0.34$ and $\mathbf{p}_{o,2} = (-1.2, 2.1, 0.8)^T$, $r_{o,2} = 0.39$.

3.4 Summary

In this chapter, a task-based optimal composition synthesis method for modular and reconfigurable robot manipulators was presented. To the best of our knowledge, none of the evolutionary algorithm-based approaches presented in the literature so far has considered task-related constraints in the evaluation function. This idea enables the user to generate the optimal composition of the modules in a time-efficient way, which makes modular and reconfigurable robots a more promising technology in the industrial environment. Comparisons show that our method is more efficient than the current methods in the literature, which is based on eliminating compositions from a brute-force algorithm in case there are many compositions that can fulfil the task when only considering the initial and goal positions. This proposed algorithm also shows that consideration of only the initial and goal positions does not provide the cost-optimal solution for the given task. Although the proposed method does not guarantee that the best composition is the one obtained from this heuristic algorithm, the obtained solution does have a higher chance of being the best as it is selected using the fitness values.

The main advantages of the proposed optimal composition synthesis algorithm can be summarized as follows: *i*) it is applicable to all types of modules, *ii*) it only generates cost-optimal obstacle-free paths for the set of the best compositions, which makes it computationally efficient, and *iii*) it provides a faster solution when compared to finding assemblies by optimizing trajectories for each assembly individually.

Chapter 4

A Time-Efficient Collision-Free Motion Planners for Redundant Modular Manipulators

The generation of an optimal modular robot composition not only requires efficient algorithms, but also computationally efficient motion planners that perform the task in a short time. One test that composition synthesis algorithms spend a significant amount of time on is motion planning. Indeed, the focus of this chapter is computationally fast and efficient motion planning of several module compositions as this test significantly contributes to the total computational time. The main idea behind the proposed motion planners is the implementation of a two-step path planner. First, a common geometric path for all possible compositions is generated, which is followed by an individual path planner for each composition. These proposed path planning methods are investigated and their results are given at the end of the chapter.

First, a literature review on current workspace-based path planning algorithms (Section 4.1) is given. Then, the proposed algorithms are detailed in Section 4.2 and Section 4.3. The implementation for the proposed algorithms in Section 4.2 and Section 4.3 are given in Section 4.4.1 and Section 4.4.2, respectively. The proposed method explained in Section 4.3 was published in [6].

4.1 Introduction

Collision-free path planning for robots can be summarized as finding a path between an initial and goal position without collision with any obstacles in the environment. It is a well-known problem in the area of robotics and, as a result, it has been intensively studied by many approaches [78, 102–110]. Depending on the environment, the path planning problem can be divided in two groups: static path planning and dynamic path planning. The environment that we consider in this thesis is static and therefore has obstacles in stationary, pre-determined positions so there is no need for updates in the environment. The proposed approaches for static path planning can be grouped into bug algorithms, graph search-based methods (such as A*), potential field algorithms, sampling-based methods, roadmap algorithms, cell decomposition methods, Voronoi diagrams, and heuristic approaches (for more details see [78]).

All of the approaches above are classified as either configuration space (C-space) or workspace approaches. In C-space approaches, the manipulator is represented as a point in an n-dimensional space where n is the number of degrees of freedom (DOF) of the manipulator, which requires the transformation of task points and obstacles (the latter are known as C-obstacles) from the workspace to the C-space [102]. However, the generation of C-space and C-obstacles exponentially increases the complexity by a factor of n that will obviously increase the computational time. In contrast, workspace-based approaches directly generate a path in the physical environment without requiring mapping from one space to another. These algorithms not only result in more geometric and intuitive paths, they also decrease the computational work, especially for high-DOF systems, because they do not require mapping of the whole environment.

A common approach to workspace algorithms is to first find a collision-free space for the end effector and then to calculate the inverse kinematics for certain points along the path. In [111], a workspace path planner that is based on mapping links from 3D to 2D environments is proposed. A motion planner for redundant manipulators using a bi-level optimization algorithm is presented in [112]. In this method, the path is partitioned into small pieces and the next position of the end effector is generated in workspace. The optimal configuration of the generated position is calculated by using the redundancy. A repulsive potential field-based approach is proposed in [113]. The method considers that surfaces of obstacles are charged and thus generate forces and torques between obstacles and manipulators. The algorithm finds a path using the potential field generated by the above-mentioned forces and torques. However, this algorithm also has a major drawback: the potential field-based algorithm can stack local minima. Some algorithms using C-space path planning approaches have proposed a partial transformation of the environment: they do the planning in C-space but do not map the whole environment to C-space. For example, the probabilistic roadmap method (PRM) proposed in [107] introduces a two-phase roadmap-based planner that does not require the mapping of the whole environment into the C-space. In PRM, a graph (called *roadmaps*) consisting of collision-free configurations represented by edges and nodes are generated and stored in the first phase (called the *learning phase*). In the second phase (called the *query phase*), the method searches for a path from any given start and goal position to the nodes of roadmaps. Then, it finds a set of edges connecting the previously found nodes using graph search. Even though PRM has been very successful for many problems (e.g. for manipulators with high DOF), it may terminate without finding a feasible path because of its time constraint.

In another approach, the rapidly-exploring random tree (RRT) algorithm uses a workspace heuristic function as a guide to the generated trees in C-space to reach the target [114]. The main advantages of this algorithm are the non-essential explicit IK calculations and the fact that generated paths are already nearly the shortest possible paths because they are in the same topological class as the shortest path solution. Another RRT-based workspace path planner called Jacobian transpose-directed rapidly exploring random tree (JT-RRT) is proposed in [115]. In JT-RRT, the authors use the Jacobian transpose (JT) method while growing the trees in the workspace. In [79], an RRT algorithm in task-space is proposed to find an obstacle-free path for redundant manipulators in high-dimensional space. In JT-RRT approaches, inverse kinematics are solved independently, which may lead to collisions between two sequential arbitrary points. To solve this problem, a hierarchical path planner consisting of a global path planner (GPP) and a local motion planner (LMP) is introduced in [9]. The algorithm finds a geometric path in GPP using a workspace planner and the genetic algorithm-based LMP finds a finer path between the positions determined by the GPP. Aa genetic algorithm with non-random initial population dramatically decreases the neighboring points in C-space. However, it selects only one path generated for the end effector and the planner fails if the algorithm could not find a collision-free path within the predefined path. Because a large number of different compositions exist and mapping each composition to find C-obstacles for each composition is computationally expensive, this thesis focuses on motion planners that find solutions in the workspace of the robot.

For the proposed planners, we consider two different approaches. The first method focuses on finding a trajectory for a robot following the gaps in the environment from largest to smallest. In the first part of the algorithm, a path is found without considering any objective function. Then, a cost-optimal trajectory for the generated path is created. The second method is also a two-step algorithm. In the first step of this algorithm, the planner finds several shortest geometric paths for a robot. Then, in the second step, the best path using the best inverse kinematics solutions along the geometric path are selected.

4.2 Extended Follow-the-Gap Method

The proposed approach is a workspace-based, two-step motion planner. In the first step, a geometric obstacle-free path is generated considering the size of the gaps between obstacles in the environment until the goal position is reached. Then, trajectories are generated between the path points obtained in the first step. The obtained points along the splines are checked against possible collisions and then collision-free trajectories are generated. While checking for collisions, the same assumptions of Section 2.4.4 are also considered here. As in Section 2.4.4, each module and obstacle are defined by 4 variables: (x, y, z, r) where x, y and z are the Cartesian coordinates of the centers of each module or obstacle and r is the radius of the component or obstacle. For the collision check, we follow the algorithms detailed in Section 2.4.4.

The proposed method is an extension of the Follow-the-Gap Method (FGM) in [116]. In [116], the authors consider autonomous vehicles in a 2D environment. They consider the mobile robot as a point, modelled obstacles in 2D, and used sensor data to generate a path in real-time. In contrast to this approach, this work adapts FGM to manipulators in a previously known 3D environment. The end effector is directed towards the goal position following the largest gap between obstacles or boundaries of the environment in its immediate surroundings. The main idea behind the proposed algorithm is to keep the end effector as far away as possible from obstacles or boundaries in the environment thus reducing the risk of end effector collisions.

The motion planning methods mentioned in the previous sections target the generation of an optimal path for individual solutions (or a common optimal solution). Contrary to these methods, the main motivation behind the proposed extended FGM is to obtain a feasible path in the shortest computational time. As a result, the generated path may not be the optimal solution, however, decreasing computational time is one of the main challenges to task-based design and this algorithm achieves that.

The proposed algorithm works as follows:

1. Obstacles and the end effector are represented by spheres and the distance between the center of an obstacle and the current position of the end effector is calculated from $d_{O2r} = \|\mathbf{p}_r - \mathbf{p}_O\|$ as explained in Section 2.4.4.1, where subscripts $(\cdot)_O$ and $(\cdot)_r$ represent an obstacle and the robot, respectively.

2. A distance between two obstacles is calculated as in Section 2.4.4.1 and the vector that represents the distances between all obstacles is shown as $\mathbf{d}_{O2O,b}$. In addition to that, the distances between each obstacle's border and the limits in the environment, denoted by $\mathbf{d}_{O2L,b}$, are calculated. In the following, the vector $\mathbf{d}_{gap} = [\mathbf{d}_{O2O,b}; \mathbf{d}_{O2L,b}]$ is obtained.

3. The components of the $\mathbf{d}_{gap}$ vector are sorted in descending order.

4. Starting from the maximum d_{gap} value, the midpoint of the gap is obtained. Then, the existence of an obstacle between the robot and the midpoint of the gap is checked. Furthermore, it is checked whether the robot is closer to the goal position than the midpoint of the gap $\|\mathbf{p}_{mid} - \mathbf{p}_g\| < \|\mathbf{p}_i - \mathbf{p}_g\|$, where the distance between the midpoint of the gap $\mathbf{p}_{mid}$ and goal position $\mathbf{p}_g$ is denoted by $\|\mathbf{p}_{mid} - \mathbf{p}_g\|$ and the distance between the robot's current position $\mathbf{p}_i$ and goal position $\mathbf{p}_g$ is denoted by $\|\mathbf{p}_i - \mathbf{p}_g\|$.

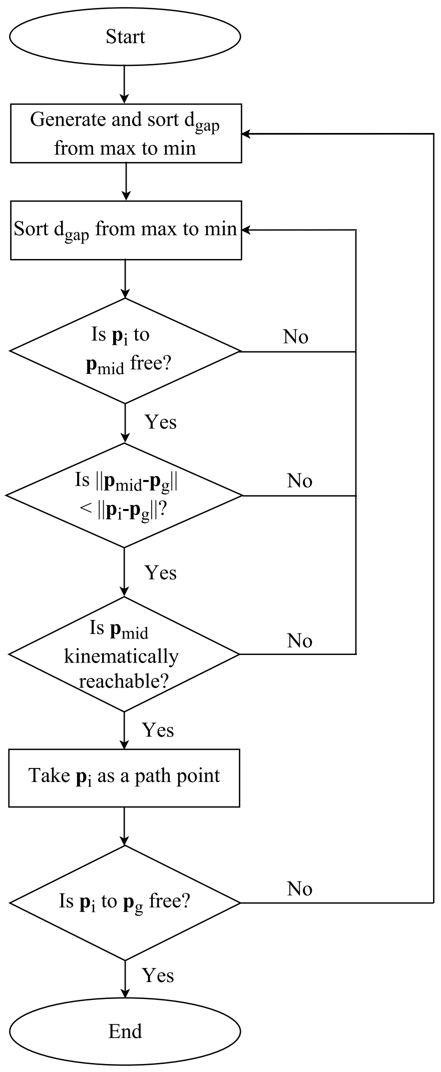

Start
Generate and sort d_gap from max to min
Sort d_gap from max to min
Is p_i to p_mid free?
No
Yes
Is ||p_mid-p_g|| < ||p_i-p_g||?
No
Yes
Is p_mid kinematically reachable?
No
Yes
Take p_i as a path point
Is p_i to p_g free?
No
Yes
End

Figure 4.1: Flow chart of the proposed method.

5. In case *i*) there is an obstacle between $\mathbf{p}_i$ and $\mathbf{p}_{mid}$ and *ii*) $\mathbf{p}_{mid}$ is closer to $\mathbf{p}_g$ than $\mathbf{p}_i$, the point $\mathbf{p}_{mid}$ is checked to determine whether it is kinematically reachable or not. To this end, while using the inverse kinematic algorithm, the accuracy is defined by the length of the gap. When $\mathbf{p}_{mid}$ satisfies these conditions, it is added as a path point. If it does not satisfy the conditions, step number 4 is repeated as in Figure 4.1.

6. In case the robot cannot reach $\mathbf{p}_{mid}$, the algorithm checks the next largest gap and repeats step numbers 4 and 5 as in Figure 4.1.

This procedure is repeated until the manipulator reaches the goal position. After finding a collision-free path for the end effector, the entire robot is then considered and the algorithm checks if the generated path is collision-free or not. To check the collisions between the robot and obstacles while the robot is moving from one point to another, a trajectory using cubic spline functions for each joint between two sequential path points is generated. The motivation behind selecting the cubic spline is its simplicity. Due to the huge computational effort of mapping each obstacle into C-space, each spline is divided into pieces of finite length that is defined by the radius of the smallest obstacle in the environment. The algorithm checks for collision at each piece in the workspace using the methods explained in Section 2.4.4. The position of each robot's components are obtained from the transformation matrices detailed in Appendix D. Considering the velocity and acceleration limits of the joints, the cost optimal trajectories between the points obtained by the geometric path are generated as in [117]. After obtaining all trajectories for each feasible composition, the results are compared and the module composition with the minimum cost value is selected as the optimal module composition.

4.3 Hierarchical Genetic Path Planner for Highly Redundant Manipulators

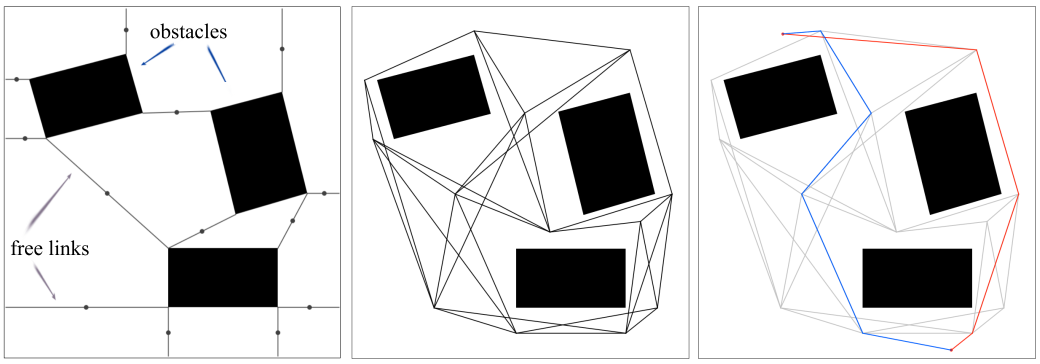
This approach is based on task-space-based two-step path planner consisting of a global path planner (GPP) and a local motion planner (LMP). In the GPP, a collision-free path in the workspace is obtained and the LMP selects intermediate points for each position obtained from the GPP. The LMP is based on a genetic algorithm that reuses the population from one intermediate goal to the next, which reduces joint angle variations. Different from the previous chapter, the GA is used while calculating the inverse kinematics of the robot. It should also be noted that the proposed algorithm is applied only for 2D path planning problems.

4.3.1 Global Path Planner (GPP)

The proposed global path planner uses binary space partitions to generate several collision-free paths in the workspace. The obstacles in the environment are represented by polygons and their vertices are named as *obstacle vertices*. The algorithm divides the free space into convex areas using obstacle vertices as seen in Figure 4.2(a). Obstacle vertices are connected to each other or to workspace boundaries with lines that are called *free links*. In the proposed algorithm, we firstly select candidate free links to connect obstacle vertices with other vertices or to the boundaries of the environment. It should be noted that only segments that do not intersect with the edges of obstacles are selected. The candidates are sorted from the shortest to longest length.

As a next step, the candidates are selected one-by-one and the algorithm checks whether it is a convex hull of vertices (a vertex is defined as a convex if all angles formed by adjacent segments are less than 180° [118]). The candidate segment is selected as a free link if it makes a convex vertex or reduces the largest angle. This step is repeated until all obstacle vertices are convex. Redundant free links are removed if their removal does not change the convexity of the vertices that it connects with. The midpoints of all remaining free links are selected (see Figure 4.2(b)) and visibility graphs that connect all midpoints without intersecting with obstacle edges are generated (see Figure 4.2(c)). We generate multiple collision-free paths from the graphs for given initial and end points as in Figure 4.2(c).

Because manipulators have a fixed base and cannot freely move in a workspace, paths generated by MAKLINK (proposed for mobile robots) are not always feasible. As seen in Figure 4.3, the path found by the algorithm cannot be followed by the manipulator without collision with the obstacles in the environment. The path planner in [9] follows the shortest path in the graph, but it is not always possible



(a) Free link division of the free space into convex areas (b) Visibility graph for midpoints (c) Collision-free paths in the visibility graph

Figure 4.2: Global Path Planner [6]

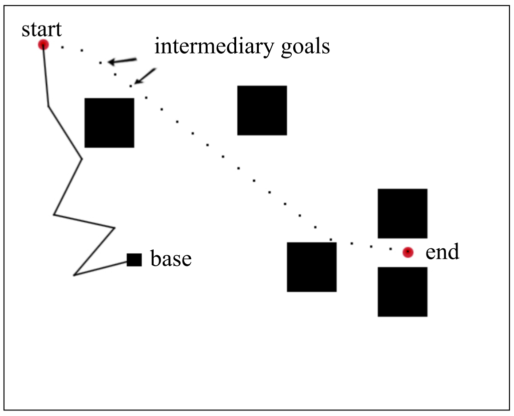


Figure 4.3: Shortest path in the visibility graph [6]

to achieve that without any collision with obstacles in the environment, so the LMP fails for these cases (see Figure 4.3). In the proposed algorithm, we solve this problem by selecting multiple paths and trying other solutions if the algorithm fails to find a feasible path. Other important advantages of the proposed planner is that it only generates non-homotopic paths, which means that a path does not deform the other paths. For each new path, the homotopy of every path in the set is computed. A set of homotopic paths is called as *homotopic group*. If the new path is non-homotopic with the rest of the generated paths, it is added to the set. In case there is a homotopy, the shortest path is selected. As a result, the generated non-homotopic paths are the shortest paths from their homotopic groups. Because the shortest path in each homotopic group is relevant, this approach reduces the computational effort. Even if another path in the homotopic group is feasible, a robot that can follow the shortest path is preferred over one that requires a longer homotopic path.

The paths are generated using a depth-first traversal algorithm with a cut-off condition that ensures that the path's length is less than 2.5 times that of the robot's length. This number for the maximum length is arbitrarily chosen to limit the path that robot will follow. In the proposed approach, the GPP provides the LMP with each non-homotopic path starting from the shortest path to the longest one.

4.3.2 Local Motion Planner (LMP)

To obtain a finer path, equally-spaced points along the global path are linearly interpolated by the MAK-LINK algorithm. These points are called intermediate goals and the end effector is expected to follow these points from the initial position to the goal position. To find robot configurations at intermediate points, GA is used due to its ability to find minima or maxima of non-differentiable functions. Details of the GA are explained in Section 3.2.2. The LMP with non-random initial population uses the Algorithm 3 that is proposed in [9]. The sequence of joint angles $[q_1, q_2, \ldots, q_n]$ in n-dimensional space generates a chromosome while the gene values are floating point numbers from the interval $[0, 2\pi)$. The fitness function is defined as follows:

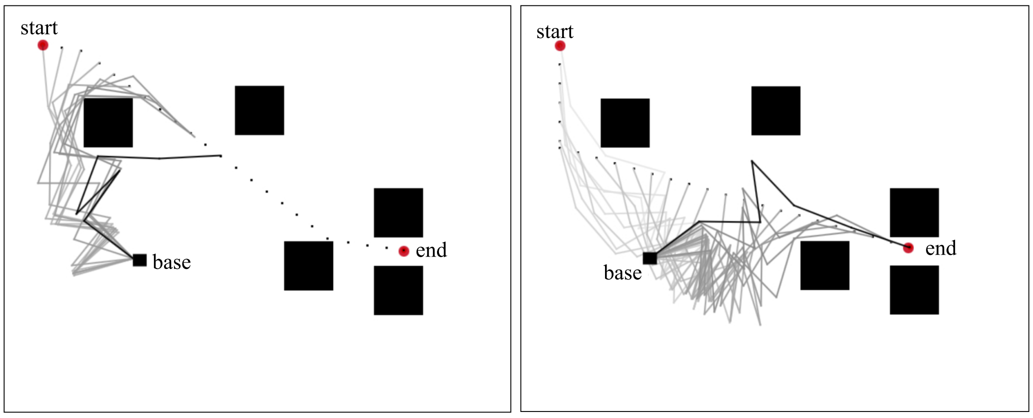
$$F = f_{collision} \cdot D_s \tag{4.1}$$

where $f_{collision}$ is the collision coefficient and D_s is the Euclidean distance between the position of the end effector and the intermediate points. The variable $f_{collision}$ equals to 1 if the robot is collision-free in the given configuration, otherwise, the given configuration is a high-value constant (V_{max}).

When an intermediate goal is reached, the final population is set as the initial population for the next generation. The reason behind this is to ensure continuity. The randomly-generated initial population increases the probability of the next solution being far away from the previous goal's solution, which also might lead to incorrect solutions. Figure 4.4(a) can be given as an example of this. The incorrect solution led by a random search is non-homotopic and the manipulator reached the next intermediate point that was not collision-free. The proposed planner checks the existence of the obstacle's vertices in the area swept by the manipulator. To do this, it constructs a polygon using the manipulator in two sequential configurations. In case there is an obstacle vertex in the polygon, the path is unfeasible and the GPP selects the next shortest path as an alternative (see Figure 4.4(b)).

Algorithm 3 Genetic algorithm with non-random initial population [9]

```
1:  $i \leftarrow 1$ {Initialize the first intermediate goal}
2:  $t \leftarrow 1$ {Initialize the genetic generations}
3:  randomly generate an initial population $P_i(t)$
4:  compute fitness $F_i(t)$
5:  **repeat**
6:      **repeat**
7:          select $P_i(t+1)$ from $P_i(t)$
8:          crossover $P_i(t+1)$
9:          mutate $P_i(t+1)$
10:         compute fitness $F_i(t+1)$
11:         $t \leftarrow t+1$
12:     **until** reached intermediate goal $q_i$
13:     $P_{i+1}(t) \leftarrow P_i(t)$
14:     $i \leftarrow i+1$
15: **until** reached final goal $q_n$
```



(a) Incorrect avoidance of an obstacle (b) Alternative path around the obstacle

Figure 4.4: The proposed path planner

4.4 Simulation Results

4.4.1 Simulation Results for the Extended Follow the Gap Method

To demonstrate the applicability of our proposed algorithm, we implement it in MATLAB R2016a on an Intel® Core™ i7 processor with 2.30 GHz and 16 GB of memory. We use the virtual module set explained in Appendix A. The task is defined as carrying the given payload from the initial position to the goal position in the shortest time without colliding with obstacles in the environment and without violating the joint limits of the 5-DOF robot. The payload is 5 kg and the initial and goal positions are defined as $\mathbf{p}_s = (1.95, -1, 1.25)^T$ m and $\mathbf{p}_g = (-1.55,\ -1.75,\ 1.25)^T$ m, respectively. It is assumed that there are four static obstacles in the environment whose center points are defined as $\mathbf{p}_{1,O} = (0.75,\ -1,\ -0.6)^T$ m, $\mathbf{p}_{2,O} = (1,\ -2.3,\ 2)^T$ m, $\mathbf{p}_{3,O} = (0,\ -0.6,\ 1)^T$ m, and $\mathbf{p}_{4,O} = (-0.5,\ -1.5,\ -0.9)^T$ m. The radii of the obstacles are given as $r_1 = 0.1$ m, $r_2 = 0.15$ m, $r_3 = 0.15$ m, and $r_4 = 0.1$ m. We also define the limits of the environment as $x_{min} = -3$ m, $y_{min} = -2$ m, $z_{min} = -3$ m, $x_{max} = 2$ m, $y_{max} = 3$ m, and $z_{max} = 3$ m.

As a first step, we apply the composition synthesis algorithm in [24] and obtain 57 compositions that are feasible at the initial and goal positions. As a second step, we apply the proposed method to all 57 compositions. First, all distances between the obstacles and all distances to each limit in the environment are calculated and sorted from the largest gap to the shortest gap. Starting from the largest one, the midpoints between the obstacles are found and the planner checks whether *i*) there is any obstacle between the current point and determined point, *ii*) the midpoint is kinematically reachable, and *iii*) the midpoint is closer to the goal point than the current point. If the point satisfies these three conditions, the planner considers it as a path point and this procedure is repeated until the end effector reaches to the goal point. After path points are generated, trajectories between two pre-determined path points are generated using cubic splines and divided into pieces. The planner checks each piece for collision in workspace. It was observed that using 100 pieces to approximate the path yields sufficiently accurate results.

We compare our method against the RRT algorithm [109], wavefront algorithm [78], and task space-RRT algorithm (TS-RRT) [79]. The wavefront and TS-RRT algorithms could not find a collision-free path between the initial and goal position for most of the remaining compositions and they were computationally expensive. The RRT algorithm, which is a popular path planning approach due to its fast convergence and high success probability [109], employs random points generated in the configuration space to find a collision-free path from the initial position to the goal position. Although random points are generated in configuration space, the collision check is done in workspace. To be consistent with the proposed algorithm, the maximum displacement at each iteration (*step size*) and the maximum number of iterations (*max number of trees*) were set to 0.1 and 200, respectively. The proposed algorithm obtains the trajectory in an 84.6% shorter time than the RRT algorithm. When comparing the computational times of both methods to perform the task, the trajectory obtained by the proposed method is 30.8% less than that of the RRT algorithm in terms of total CPU time. The cost-optimal paths obtained from the both algorithms are given in Figure 4.5. As seen from Figure 4.5, the trajectories obtained from the proposed method are smoother than that of the RRT algorithm.

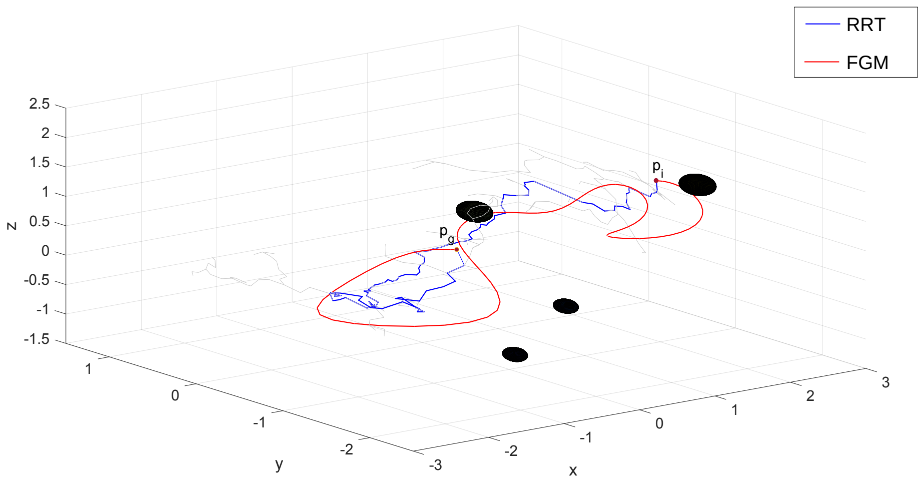


Figure 4.5: Comparison of the trajectories obtained by CS-RRT and FGM.

4.4.2 Simulation Results for the Hierarchical Path Planner

To demonstrate the applicability of our path planner, we implement it in Java on a Mac mini with a 2.6 GHz Intel Core i5 processor. We use a 6-DOF manipulator in a workspace with several rectangular obstacles as in [9]. We used the same scenarios generated in [9] to make a better comparison between planners. The environments given in Figure 4.6(a) and in Figure 4.6(b) are Figure 15 and Figure 10 in [9], respectively. We first compare the proposed planner with another hierarchical planner in [9] whose GPP only considers the shortest path. For the comparisons, the LMP algorithm and its parameters are considered to be the same for both planners. We perform the same experiments in the same environment with 1000 randomly generated scenarios using standard uniform distribution. It should be noted that base location, initial point, and final point are different for each scenario. While generating these points, we ensure that all points are in *i*) the workspace, *ii*) the obstacle-free areas, and *iii*) the initial and the final points are reachable for the manipulator (implementing GA-based inverse kinematic algorithm).

We count the number of solved path planning problems for each planner. The planner is considered to be successful if it finds a collision-free path from the initial point to the final point within a radius of 0.1 cm. It should be noted that the dimensions of the environment in Figure 4.6(a) are 67 x 53 cm, which demonstrates how small the considered radius is. We also set another condition for early detection of unfeasible paths. Based on that condition, the maximum distance between the end effector and the goal should not exceed twice the distance between two intermediate goals. The reason behind this is to prevent unfeasible paths as in Figure 4.4(a). In the given scenario, the potential solution will be discarded due to this condition because the manipulator is fully extended and it is wrapped around

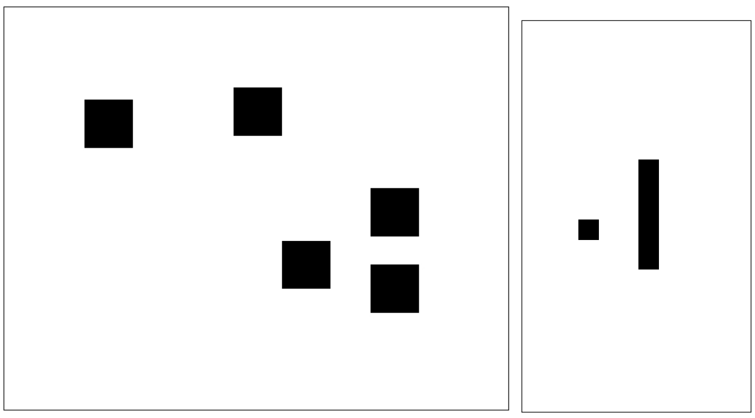

(a) Scenario 1 (b) Scenario 2

Figure 4.6: The scenarios used in the experiments.

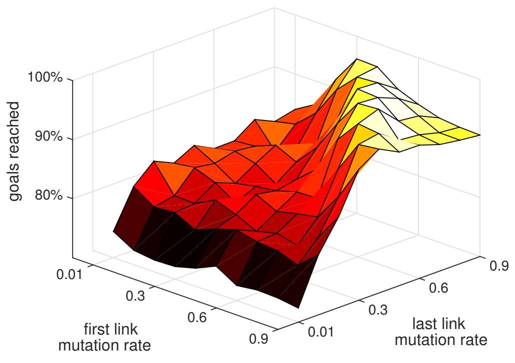
Table 4.1: The comparison of the method in [9] and the proposed method in 1000 scenarios.

	method in [9]	proposed method
number of solutions found for the environment in Figure 4.6(a)	771	987
number of solutions found for the environment in Figure 4.6(b)	906	977

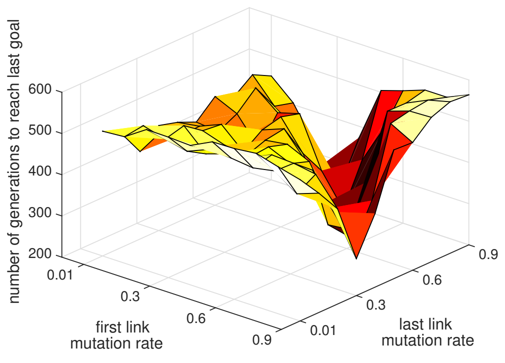
the obstacle on the left. In this case, no further progress is possible along the prescribed path and this solution is directly discarded.

In the environment of Figure 4.6(a), the planner [9] that only considers the shortest path found solutions for 771 scenarios while the proposed path planner found 987. The average execution time for each case was 8.07 sec. We also initiated both algorithms for the environment in Figure 4.6(b) and the planner that only considers the shortest path found solutions for 906 scenarios while the proposed path planner found 977 scenarios. The average execution time for the proposed planner for each of these cases was 0.089 sec. The summary of the results are given in Table 4.1.

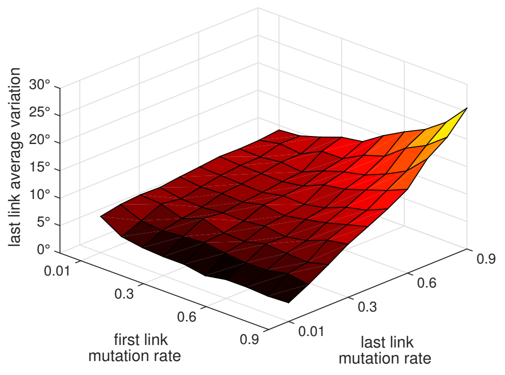
To find optimal parameters for the genetic algorithm, we consider the following criteria: *i*) the percentage of the reached intermediate goals, *ii*) the required number of the generations to reach the intermediate goals, and *iii*) the average joint variations of every joint for each adjacent intermediate goals. Apparently, the higher the percentage of reached intermediate goals, the lower the number of generations and lower joint variation values. Reusing the population from the previous intermediate goal as an initial population helps the user achieve lower joint variations. To use the same parameters as in [9], we consider the following values: the population size is 100, the maximum number of generations is 600, the selection takes the fittest 50% chromosomes to generate two offspring, the crossover exchanges one gene between two parents, and the mutation changes one gene in a chromosome. For the missing parameters, we consider the following assumptions: *i*) the module compositions (represented by chromosomes) whose end effectors collide with obstacles or boundaries are penalized and the collision punishment cost (V_{max}) is taken as 1000 (where higher values give safer solutions), *ii*) the distance between two adjacent intermediate goal points is 2.5 cm and is proportional to the width and the height of the environment to the robot's length, and *iii*) the parent chromosomes are randomly chosen between the fittest 50% of the chromosomes obtained from the selection step. It should be noted that the most dominant parameter in path planning is the mutation rate, however, the mutation rate is not given in [9]. To define it, we analyzed all possible combinations for the mutation rate of the base and distal joint from 0 to 0.9 in 0.1 intervals. A mutation rate of 0 or 1 is not useful because 0 means no changes in genes (so, we take 0.01 instead of 0 in our simulations) and 1 means changing all genes all the time. Assigning the same mutation rate to all joints is also not realistic and we consider different mutation rates for the joint



(a) Percentage of intermediate goals reached



(b) Number of generations to reach the last goal



(c) Last link angle variation

Figure 4.7: Mutation rate analysis for the workspace in Figure 4.6(a)

close to the base and end effector. The intermediate joints between them are calculated by an arithmetic progression. For instance, if the mutation rate for the base joint is 0.2 and that of the distal joint is 0.6, the mutation rates for all joints is the array $[0.2, 0.28, 0.36, 0.44, 0.52, 0.6]$. The mutation rates can also be decreasing. For example, if the base joint mutation is 0.8 and the distal mutation rate is 0.3, then the mutation rates for all joints is the array $[0.8, 0.7, 0.6, 0.5, 0.4, 0.3]$. The algorithm is run 100 times for each pair and the averages for the quality criteria are gathered. Results for the environment in Figure 4.6(a) can be seen in Figure 4.7. The best values for the mutation rate in terms of efficiency, accuracy, and low value of joint variations can be found along a line going from $(0.4, 0.9)$ to $(0.9, 0.4)$ in Figure 4.7. These graphs also prove that using the same mutation rate for all joints is not the best solution. Results for the other scenario in 4.6(b) are similar. As seen from the comparison with a conventional GA implementation in Figure 4.8, the average joint variations are reduced by 4 to 8 times,

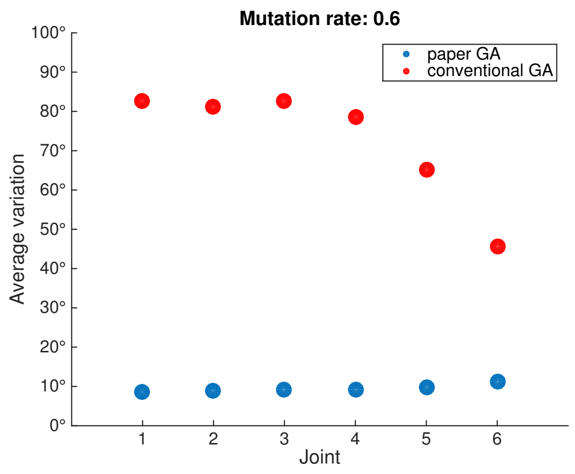


Figure 4.8: Average joint variations for the workspace from Fig. 4.6(b)

which increases the probability of finding a feasible C-space path.

The proposed path planner shows significant improvements and is more robust when it is compared to the planner in [9] because it considers paths other than the shortest one. The proposed path planner improves the percentage of successful path planners from 77% to 99% and from 91% to 98% for the scenarios in Figure 4.6(a) and Figure 4.6(b), respectively. Completing a mutation rate analysis also allowed us to find proper GA parameters making 96% of the cases successful. The presented GA provides better solutions than the conventional GA, which creates feasible C-space paths because it reduced the average variations by 4 to 8 times. Moreover, the experiments showed that the genetic algorithm is only slightly more efficient when using non-random initial populations compared to random ones, which contradicts the claim in [9]. We also verify the viability of the solution directly in the workspace by checking that the space swept by the manipulator does not contain any obstacle vertices. If it does, the solution is not viable and another path must be selected.

4.5 Summary

As opposed to the previous chapters, this chapter focuses only on the path planning part of a robot's task fulfilment. A computationally efficient hierarchical task space path planning algorithm was given. After a literature review, two novel path planning approaches for manipulators were presented.

The method presented in Section 4.2 generated a path following the largest gaps in the environment and thus enabled the user to obtain a path in a short computational time. Taking into consideration that one can generate thousands of different robot compositions using modular robots, any level of reduction in path planning simulation time could significantly decrease the total simulation time. It should be noted that the proposed algorithm does not guarantee the cost-optimal path, but it does give the cost-optimal solution for the generated path. The main advantage of this planner is that it remarkably reduces the total computational time. The experiments also show that the proposed algorithm gives better results when compared to other algorithms.

In the second planner, detailed in Section 4.3, a fast and robust two-step path planner was presented. The method divided the path planning problem into two steps. It found a set of geometric paths in the first step and a finer module composition-dependent path planner in the second step. It used the genetic algorithm in the second step that considers the previous intermediate point's joint angles as initial guesses. The robustness of the proposed algorithm and its short computational time while finding feasible paths encouraged the authors to use it for the automatic search for modular robot assemblies.

Chapter 5

Conclusions

The task-based composition synthesis problem of modular robot manipulators was investigated in this thesis. The available composition synthesis methods for modular robots in the literature only consider if the user-defined positions are achievable and notably do not consider how the task should be fulfilled. In contrast to other works, task constraints and the manner of task fulfilment are considered in this thesis while finding potential module compositions. In addition, both robot kinematics and robot dynamics were considered during the search for those compositions. The simulations and experiments in this thesis demonstrated that the proposed methods not only enable the user to find feasible module compositions, but also provide the cost-optimal solution for a given task.

The main ideas behind the proposed algorithms are grouping the task requirements and systematically discarding the unfeasible or least-likely optimal module compositions. Then, these grouped task requirements are ordered based on their computational time. In the exhaustive search-based method detailed in Chapter 2, a new hierarchical composition synthesis method was proposed in which all task requirements are sequentially checked by first using fast and easy tests and then moving to more complex and computationally expensive ones. After each test, compositions that do not pass are eliminated leaving the more complex and computationally expensive tests for the fewer remaining compositions. This method allows all potential modular robot compositions for a given task to be generated with details of how the task can be fulfilled. The first results of this algorithm are shown in [24] which is implemented into a framework proposed for flexible manufacturing as published in [5].

In addition to finding solutions to the task-based composition synthesis problem, the task-based optimal composition synthesis problem is investigated using both deterministic and evolutionary algorithms. The deterministic method detailed in Chapter 2 was the best solution for the predetermined objective function, which is based on the elimination of the least-likely compositions during optimization. The proposed method can also be applicable to different optimality conditions and task requirements. Indeed, simulations showed that the proposed method provides a solution over a shorter computational time when compared to a brute-force search and these results are shared in [39].

The implementation of deterministic methods can be time-consuming when compared to evolutionary-based algorithms. To handle this problem, we proposed a GA-based approach that considers task requirements in the objective function. Detailed performance analysis of each method has been completed and compared to exhaustive search-based methods. Experiments showed that the proposed methods found the cost-optimal solution in a shorter computational time when compared to the brute-force algorithm. The simulations that prove the applicability of the proposed method are shown in [40].

Besides composition synthesis algorithms, path planning is one of the sub-tasks that synthesis algorithms spend an excessive computational time on. Available path planning methods are time-consuming or not suitable to modular robots because thousands of different compositions must be investigated. To combat this problem, two new planners that provide faster solutions are proposed. Both of these are two-step path planners where a geometric path for all compositions in a workspace is first calculated and then a local motion planner is implemented to find individual solutions for all potential module compositions. In the first method, a path is generated following the largest gaps between obstacles. Then, individual planning is done for each potential composition. In the second method, multiple collision-free paths are generated and considered for all module compositions. The inverse kinematics of each module composition are individually solved for all generated intermediate goals. While solving the inverse kinematics problem, the GA-based IK solver uses the population from the previous intermediate point as an initial guess. Thus, the non-random initial population technique not only decreases the variations of joint angles but also reduces the risk of collisions. The first results of this method are shown in [6] and then the improved algorithm is published in [41]. These proposed path planning methods can be also applied to non-modular robots.

All proposed algorithms are shown by simulations or experiment. The proposed approaches enable users without any knowledge in robotics to reconfigure a modular robots for different production scenarios. The developed visualization tool also helps users quickly see the algorithm's results and foresee how the robot will move while performing the task before reassembling the modules. Based on these results, we can say that our approaches are mature enough to be applied to real robots.

5.1 Future Directions

The task-based composition synthesis approach presented in this thesis serves as a foundation for future advanced research in modular robots. The continuation of this work may involve learning-based composition synthesis algorithms. Machine learning algorithms may guide the composition synthesis algorithm with eliminating modules or module compositions that are the least-likely to fulfil a given task based on previous simulations. Cancelling the least-likely combinations dramatically reduces the search space, which decreases total computational time. In addition, learning-based methods may also help the user determine a set of module compositions that have a higher chance of fulfilling the task. Algorithms can be applied only to this set, which considerably reduces the computational time as it shrinks the search space.

Another potential continuation may be to improve module design. The commercially available modular robots are not robust enough and their payload capacities are too limited for most industry tasks. More complex and robust designs that improve their capabilities are required. Moreover, for tasks that require longer link modules, the user should assemble two link modules to extend the robot's length, which decreases the load capacity. Robust telescopic link modules with strong connectors will solve this problem and help widen their usage in industry.

In addition, the application of different optimization methods (especially for the different evaluation algorithms) may be considered to speed up the process and to avoid stacked local minima. The genetic algorithm is selected for the evolutionary-based composition synthesis algorithm based on its superior features to other algorithms. New evolutionary-based optimization algorithms have been proposed and may give better results for this problem.

Appendix A

Virtual Module Set

To demonstrate our proposed algorithms, we design simple modules in the Computer-Aided Design (CAD) environment considering the specifications of the available modular and reconfigurable manipulator designs detailed in Section 1.2. To increase the variety of the modules, we design one type of 0-DOF base module, two types of joint modules, namely, 1-DOF revolute joints (J_R) and 1-DOF prismatic joints (J_P), and three types of 0-DOF link modules which are *i)* $\alpha = 90°$ link module whose length is along the y-direction of the previous coordinate system (L_1), *ii)* $\alpha = 0°$ link module whose length is along the x-direction of the previous coordinate system (L_2), and *iii)* $\alpha = 90°$ link module whose length is along the z-direction of the previous coordinate system (L_3) where α represents the angle between the input and output in the z-direction, and two types of 1-DOF end effector modules that are the revolute end effector module (E_R) and the prismatic end effector module (E_P) (see Figure A.1).

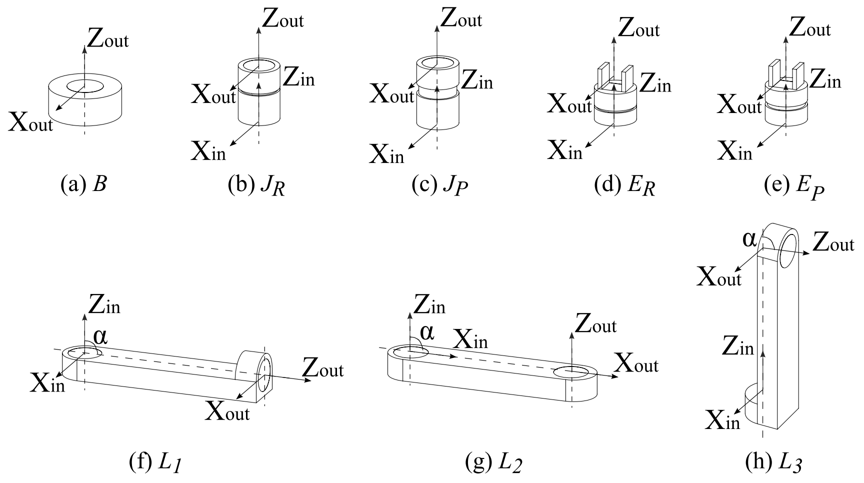


Figure A.1: Virtual module set

Table A.1: The module parameters of virtual modules involved in simulations (see Figure A.1)

	Length [m]	Diameter [m]	Max τ [Nm]	Joint Limits [rad or m]
L_1	0.75	0.2	–	–
L_2	0.75	0.2	–	–
L_3	0.75	0.2	–	–
J_R	0.25	0.2	80	[$-\pi$,π]
J_P	0.25	0.2	75	[0,0.2]
E_R	0.2	0.2	75	[$-\pi$,π]
E_P	0.2	0.2	70	[0,0.1]

The specifications of each module are given in Table A.1. The connection mechanism allows the user to attach two modules in only one orientation and the user has unlimited numbers from each module to generate their robot.

The parameters used to generate kinematic and dynamic models of both virtual module set are given in Table A.2 below. Besides, the dynamic parameters of the virtual module set can be seen in Table A.3.

Table A.2: The D-H parameters for the Virtual Module Set (see Figure A.1)

	a_{pl} [m]	α_{pl} [rad]	p_{pl} [m]	n_{pl} [m]	δ_{pl} [rad]	a_{dl} [m]	α_{dl} [rad]	p_{dl} [m]	n_{dl} [m]	δ_{dl} [rad]
L_1	0	$-\pi/2$	0	0.75	0	0	0	0	0	0
L_2	0.75	0	0	0	0	0	0	0	0	0
L_3	0	$-\pi/2$	-0.75	0	0	0	0	0	0	0
J_R	0	0	-0.15	0	0	0	0	-0.1	0	0
J_P	0	0	-0.15	0	0	0	0	-0.1	0	0
E_R	0	0	-0.1	0	0	0	0	-0.1	0	0
E_P	0	0	-0.1	0	0	0	0	-0.1	0	0

Table A.3: The dynamic parameters for the Virtual Module Set (see Figure A.1)

	m_{pl}	I_{pl}	$rcom_{pl}$	m_{dl}	I_{dl}	$rcom_{dl}$	I_m	jbv	jbc	k_{tau}	k_r
	$[kg]$	$[kgm^2]$	$[m]$	$[kg]$	$[kgm^2]$	$[m]$	$[kgm^2]$	$-$	$-$	$-$	$-$
L_1	1.62	$[743381-6; 1171323742; -6374274338]\,10^{-6}$	$[0; -175; 13.2]\,10^{-3}$	0	$zeros(3:3)$	$zeros(3:1)$	0	0	0	0	0
L_2	1.62	$[743381-6; 1171323742; -6374274338]\,10^{-6}$	$[0; -175; 13.2]\,10^{-3}$	0	$zeros(3:3)$	$zeros(3:1))$	0	0	0	0	0
L_3	1.62	$[743381-6; 1171323742; -6374274338]\,10^{-6}$	$[0; -175; 13.2]\,10^{-3}$	0	$zeros(3:3)$	$zeros(3:1)$	0	0	0	0	0
J_R	2	$diag([0.0200.0206.5\,10^{-3}]\,10^2)$	$([0; 0; 0.05])$	2	$diag([0.0200.0206.5\,10^{-3}]\,10^2)$	$([0; 0; -0.05])$	$0.09\,10^{-4}$	0	0	0.04	160
J_P	2	$diag([0.0200.0206.5\,10^{-3}]\,10^2)$	$([0; 0; 0.05]$	2	$diag([0.0300.0306.5\,10^{-3}]\,10^2)$	$([0; 0; -0.04])$	$0.09\,10^{-4}$	0	0	0.04	160
E_R	1.1	$diag([0.0040.0040.0005]\,10^2)$	$[0; 0; 50]\,10^{-3}$	1.1	$diag([0.0040.0040.0005]\,10^2)$	$[0; 0; -50]\,10^{-3})$	0	0	0	0	0
E_P	0	$zeros(3:3)$	$zeros(3:1)$	1.1	$diag([0.0040.0040.0005])$	$[0; 0; 50]\,10^{-3}$	0	0	0	0	0

Appendix B

Extended Schunk LWA 4P Robot

The SCHUNK LWA 4P robot is designed as a lightweight, 6-DOF industrial manipulator whose standard assembly can be seen in Figure B.1(a). It weighs 12.7 kg (without gripper) and has a high-resistance aluminium structure, high torque capacity, and relatively low energy consumption (more details can be found in [7]). The commercially available modules designed by SCHUNK can be seen in Figure B.1(b) and their details are given in Table B.1. The joint modules called *powerballs* have two perpendicular revolute joints with different connectors. Although the available robot gives the user modularity and reconfigurability, its reconfigurability is very limited since input and output connectors of each module are different. The possible connections of the modules are as follows: i) the larger powerballs ($PB1$ or $PB2$) can be assembled with the base module, input and output ports of $L1$, and the input port of $L2$,

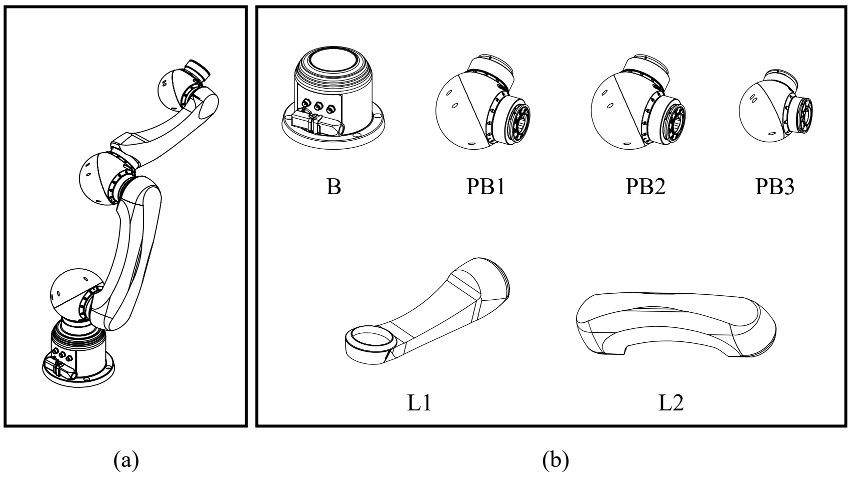


Figure B.1: (a) The commercially available SCHUNK LWA 4P robot and (b) its available modules in the market

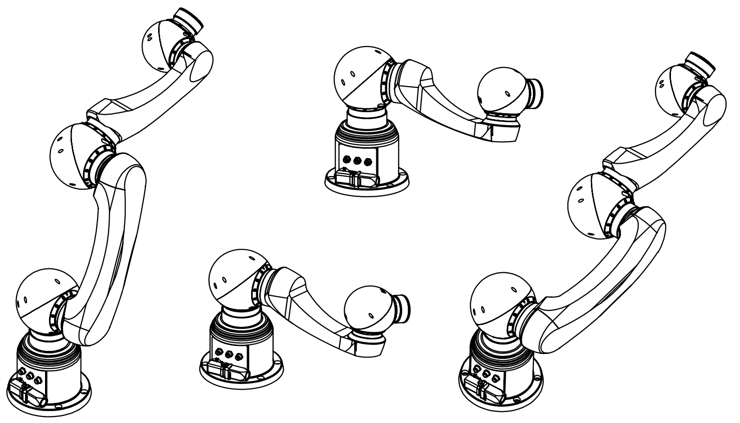


Figure B.2: All possible compositions obtained from available modules. Because there are several different commercially available end effectors compatible with the robot, the number of possible compositions can be increased by $4 \times e$ where e is the number of different end effector modules.

and *ii*) the smaller powerball ($PB3$) can only be connected with the output port of $L2$ and the input port of the end effector.

The only reconfigurability that the robot offers is changing its end effector with another end effector that SCHUNK produces [35] or by changing the place of $PB1$ and $PB2$ (which forms almost the same robot only diverging joint limits of the first and the second joint), or by reducing the DOF of the robot (removing $PB1$ or $PB2$ from the original assembly). All possible module compositions that can be obtained from the available modules produced by SCHUNK can be seen in Figure B.2. To increase reconfigurability, we design link modules with different lengths as detailed in Section B.2.

Table B.1: The module parameters of the available modules produced by SCHUNK (see Figure B.1) [7]

	Length [m]	Diameter [m]	Max τ [Nm]	Joint Limits [rad]	Velocity Limits [rad/s]	Acceleration Limits [rad/s^2]
$L1$	0.35	0.068	-	-	-	-
$L2$	0.305	0.068	-	-	-	-
$PB1_1$	0.1013	0.1013	64	[-2.967,2.967]	1.257	2.618
$PB1_2$	0.1013	0.1013	64	[-2.967,2.967]	1.257	2.618
$PB2_1$	0.1013	0.1013	64	[-2.731,2.731]	1.257	2.618
$PB2_2$	0.1013	0.1013	64	[-2.967,2.967]	1.257	2.618
$PB3_1$	0.0748	0.0748	19	[-2.967,2.967]	1.257	2.618
$PB3_2$	0.0748	0.0748	19	[-2.967,2.967]	1.257	2.618

B.1 Design Constraints

We consider the technical data given by the manufacturer in [7] while designing new modules. Based on the user manual, the maximum load capacity of the robot is considered as 6 kg (even though it is given as 7 kg in [7], it is recommended that it should not exceed 6 kg in the same document) and the maximum load value against the distance from the center of the first powerball, $PB1$ (called axis A2) is given in Figure B.3. As seen from Figure B.3, the maximum load capacity (7 kg) can be applied until a horizontal projected distance of 450 mm from the axis A2, which is considered as a reference point. Starting from this point, the load capacity falls linearly and it decreases by 3 kg at a length of 800 mm from the reference point as in Figure B.4. The end effector's weight that we have in the chair (SCHUNK parallel gripper in [119]) is 1.2 kg and thus strict the maximum load capacity is 1.8 kg at the full extension at 800 mm. The reason why the graph drops to zero at 800 mm in Figure B.3 is that the manufacturer does not consider the extension of the robot. Based on the linear decrease in Figure B.3, we consider the maximum and useful load capacities as in Table B.2, which shows that 150 mm is considered as the maximum extension for the design although the weight of the possible module slightly reduces this value. While designing new modules, we consider the following requirements: *i*) mechanical robustness, *ii*) limited strain, *iii*) full functionality, and *iv*) variable extension. The material is defined as PA *2200 Performance 1.0* from the supplier EOS whose nominal yield strength and ultimate strength are 40 MPa and 50 MPa, respectively [120]. Parameters detailed in Table B.3 are used for representing the kinematic and dynamic parameters throughout the simulations and finite element (FE) analysis. We consider the following conditions (called *normal severe dynamic conditions*) for simulations and FE analysis: *i*) all tests are completed at the position in which the robot is fully extended horizontally (as in Figure B.4), *ii*) maximum accelerations of joints are applied to joints $PB1_1$ and $PB2_2$ and the maximum acceleration against gravity is applied to $PB1_2$, *iii*) in addition to the end effector's weight, a 0.5-kg payload is considered as a wrist load, *iv*) all modules are placed between $PB2$ and $L2$ for

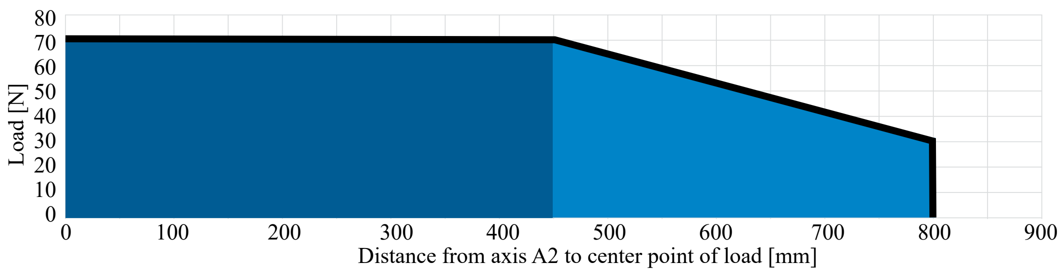


Figure B.3: Robot's load capacity as a function of its length [7]

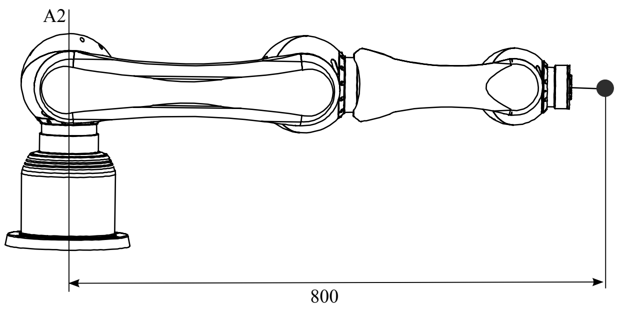


Figure B.4: Representation of the robot's position used in Figure B.3.

the original SCHUNK robot, *v*) maximum extension is considered 150 mm (see Table B.2), and *vi*) the safety factor is taken as 2. Accelerations at the joints $PB1_1$ and $PB1_2$ mostly generate bending and the acceleration at $PB2_2$ mostly causes torsion. To validate the dynamic scenarios, the Newton-Euler method (for more details see [38]) is used.

B.2 Designed Modules

While designing new components to extend the robot's length, the first focus is on basic tubular module designs that can be easily mounted or dismounted with 8 screw-flanges located at both ends as in Figure B.5. Cables are considered along the inside of the tube and original printed circuit boards (PCBs) provided by the company are mounted on both ends. The finite element analysis of the first design is done under the previously mentioned normal severe dynamic conditions with the selected PA *2200 Performance 1.0* material. The maximum Von Misses stress occurs at the fillet radius between the base flange and the cylindrical wall and around the holes in the base flange, which is caused by bending. The results show that the obtained maximum stress is less than the critical stress value and the safety factor is around 8.5 for this design. Pin holes at the base flange and the upper flange are also exposed to torsion but the maximum stress there is insignificant when compared to critical values. The stress analysis of

Table B.2: Possible load capacities and their corresponding potential extension

Extension [mm]	Maximum Load Capacity [kg]	Useful Load Capacity [kg]
0	3	1.8
30	1.66	1.4
60	2.31	1.1
90	1.97	0.7
120	1.63	0.4
150	1.29	0.1

Table B.3: Kinematic and dynamic parameters representing normal severe conditions

Joint number	Joint position [rad]	Joint velocity [rad/s]	Joint acceleration [rad/s^2]	Torque [Nm]
$PB1_1$	0	0	5.2	18.8
$PB1_2$	$\pi/2$	0	-3	-64.1
$PB2_1$	0	0	0	-26.3
$PB2_2$	0	0	5.2	0.7
$PB3_1$	0	0	0	-3.0
$PB3_2$	0	0	0	0

the first design under the given condition is also shown in Figure B.5. In addition, we analyze this design in terms of strain under the applied conditions and it is seen that the maximum strain is around 0.3 mm, which occurs at the upper flange. This is such a large displacement especially when precision is needed for a task. Moreover, the displacement of the wrist would be even larger than this value. For example, this strain value corresponds to a 1.4-mm displacement at the wrist for the pre-defined robot assembly used in the simulation (in case that we mount the module after $PB2$ and before the $L2$ in the standard robot assembly) and this displacement is caused by the bending of the designed prototype.

Because precision is important for trajectory planning and the aim is to design a robust component, the above-mentioned design should be improved. To solve the problems in the first design, the cylindrical wall is thickened from the inside because it is not possible to thicken it on the outside due to the screw holes. Moreover, fins are added to the outer surface of the cylinder in the middle of screw holes between both flanges as in Figure B.6. Stress and strain analyses are done for the new design under the same conditions, which are normal and severe dynamic conditions, and the simulations show that the

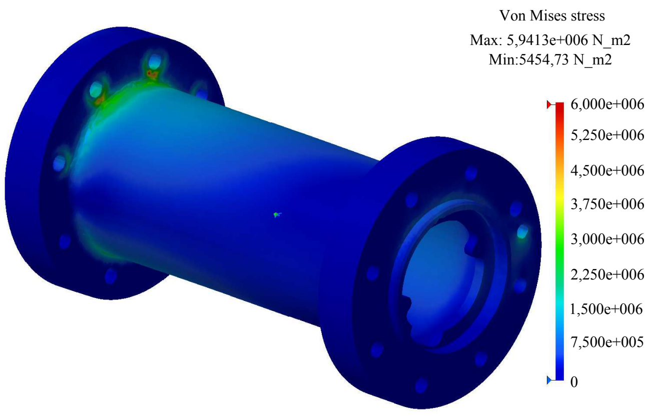
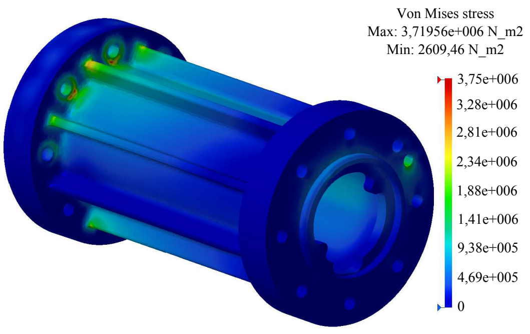


Figure B.5: Stress analysis of the basic tubular design under normal severe dynamic conditions.



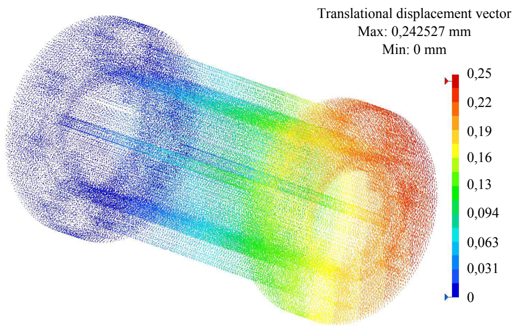
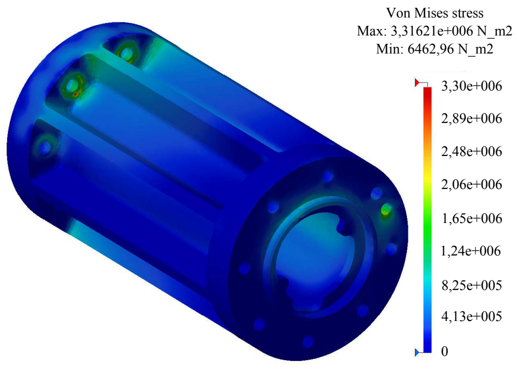


Figure B.6: (a) Stress and (b) strain analysis of the second design under normal severe dynamic conditions.

fins improve the stress distribution as they allow traction from bending to flow through the fins. The maximum stress for this design occurs at the external fillet radius, between the base flange and fins, as in Figure B.6(a). When compared to the first design, the maximum stress is reduced by 40%. In addition, it also shows better stress distribution around the screw holes with respect to the previous design. The safety factor under the applied conditions is calculated around 10. The current design not only improves the stress distribution but also improves the strain behavior of the component. The simulations show that the maximum displacement is 0.24 mm, which occurs at the top flange (see Figure B.6(b)) and it is 20% less than the previous design.

Although the second design gives a feasible solution for the applied conditions, its robustness and strain behavior should still be improved. As a final design, the fins are extended to the outer diameter of both flanges and their thicknesses are increased in both directions. The design should allow the user to tighten the screws from the outside, which could potentially interfere with the signal connector on the



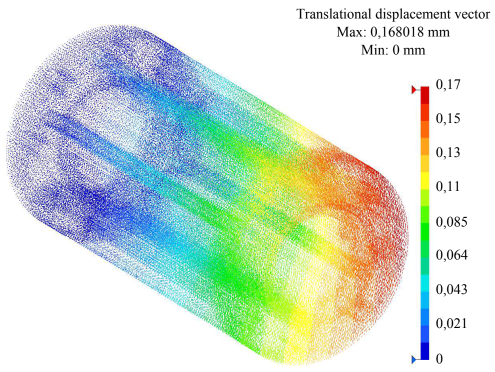
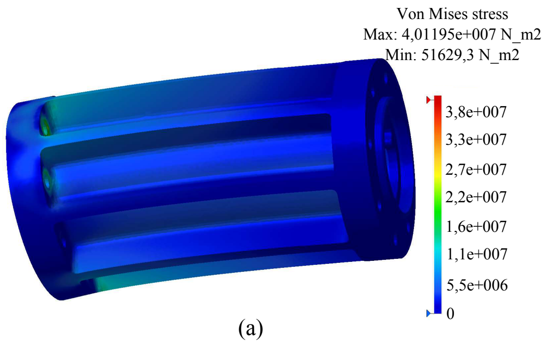


Figure B.7: (a) Stress and (b) strain analysis of the final design under normal severe dynamic conditions.

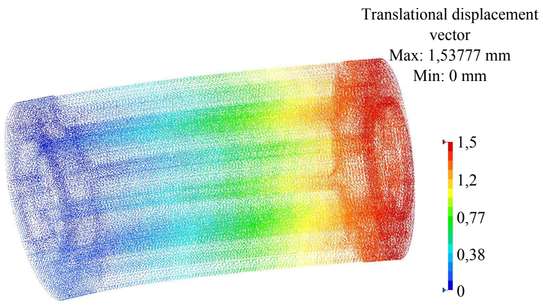
PCBs. Considering these constraints, the thickness of the design is optimized to 4.5 mm. Simulations are run under the same conditions and they show that the final design significantly improves the strength especially for bending. The maximum stress value is 11% less than the previous design and it occurs at the pin hole. The stress distribution of the final design is shown in Figure B.7(a). Contrary to the previous designs, the maximum stresses are at screw holes and result from torsion rather than bending and thus show that the final design is much stronger against bending. The safety factor for the stress of the final design is around 12. Since the bending characteristics of the final design are better than the previous ones, the final design also shows better strain behavior. Simulations show that the maximum strain is 0.17 mm for the same conditions, which corresponds to around 30% improvement in strain when compared to the second design (see Figure B.7(b)).

To see the detailed bending and torsion characteristics of the final design, pure bending and pure torsion cases were also simulated. Under pure bending, an estimated yield stress of 40 MPa reached 1660 N transversal force applied at the top flange and occurred at the fillet radii between the base flanges. Under this condition, the maximum displacement was measured as 3 mm at the top flange. The simulation results under pure bending conditions are shown in Figure B.8(a) and Figure B.8(b).

For pure torsion, the fins and walls were barely stressed while pin holes were mainly affected by torsion. The critical stress value of 40 MPa at the pin holes reached just under 21 Nm applied torque from the top flange and the analysis is shown in Figure B.9(a). However, it should be noted that the described situation at the pin is a worst-case scenario and the static friction produced by the joint pressure of the bolts is capable of transmitting a significant torque and thus reduces the stress value. The torque transmitted via friction can be estimated by using the recommended tightening torque of the joint bolts. The thread dry friction coefficient between the M4 coarse threaded bolts and a head friction coefficient between the heads of screws and the PA extension are taken as 0.2 and 0.2, respectively, based on [121].



(a)



(b)

Figure B.8: (a) Stress and (b) strain analysis of final design under pure bending.

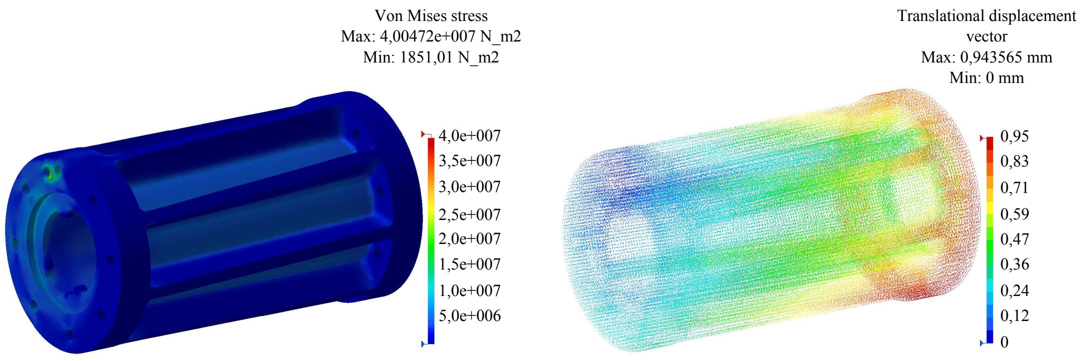


Figure B.9: (a) Stress and (b) strain analysis of final design under pure torsion.

The stress values are calculated using the online tool in [122], which uses the following formula:

$$M_T = K_{friction} \cdot \sigma_{preload} \cdot d_N \tag{B.1}$$

where M_T is the tightening torque, $K_{friction}$ is a friction coefficient obtained from the type of the thread and the friction coefficients, $\sigma_{preload}$ is the desired tensile preload, and d_N is the diameter of the bolt. Considering the tightening torque as 0.53 Nm (as recommended in [120]), the tensile stress in each bolt is calculated as 57.6 MPa and the effective thread stress is calculated as 95.4 MPa in the case where shear effects in the thread are considered. The preloading is limited by the aluminium thread as the effective thread stress is close to the alloy's yield strength, which results in a compression of 510 N from each bolt or 4080 N at the total joint. A maximum static force is obtained as 816 N while multiplying the obtained total thread stress with the static friction coefficient between the aluminium and PA. The screws are positioned at 28-mm distances from the center, which results in an estimated frictional torque transmission of up to 22.8 Nm. Therefore, it can be assumed that friction transmits around half of the torsion effort while the static friction is still valid. In case the tightening torque is guaranteed as 0.53 Nm at mounting, 40 Nm, which is approximately the half of the torsion effort (static friction is still valid), is a more realistic pure torsion resistance for the component. The maximum strain under pure torsion is calculated to be around 0.58 mm and it occurs at the outer radius of the top flange (see Figure B.9(b)). The tangential displacement is calculated to be around 0.52 mm at the flange screw holes, which corresponds to a 1.2° angular displacement.

In case the critical situation occurring at the pin holes is disregarded due to the torsion resistance of the part, a failure torque is calculated around 120 Nm that corresponds to a 6° angular displacement. The FE analysis of the realistic scenario is also done in which tangential forces at pins are considered. It

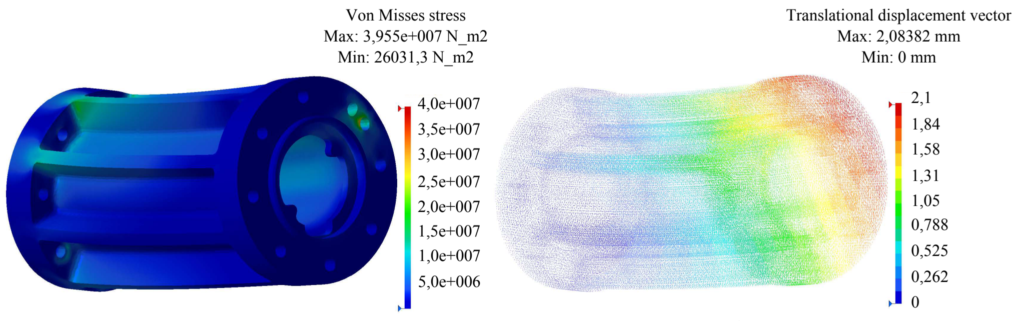


Figure B.10: (a) Stress and (b) strain analysis of the final design under realistic scenario.

is seen that this scenario mainly causes bending and the failure torque is calculated as 20 Nm, which is similar to the scenario where the friction from the bolts' pressure is discarded. The corresponding bending force for the worst-case scenario is 715 N. In case friction resulting from the screws is considered, the torque capacity is increased up to 40 Nm and the bending force is increased to 1430 N. As seen in the FE simulation of the final design in Figure B.10, the simulations show that the final design is very strong against bending and torsion, while bending plays a more important role than torsion when the transverse displacement is considered.

B.3 Produced Prototypes

We printed three components with different lengths based on the final design (see Figure B.11). The lengths of the components are set to 64 mm for $L3$, 84 mm for $L4$ and 104 mm for $L5$. The PCB plates

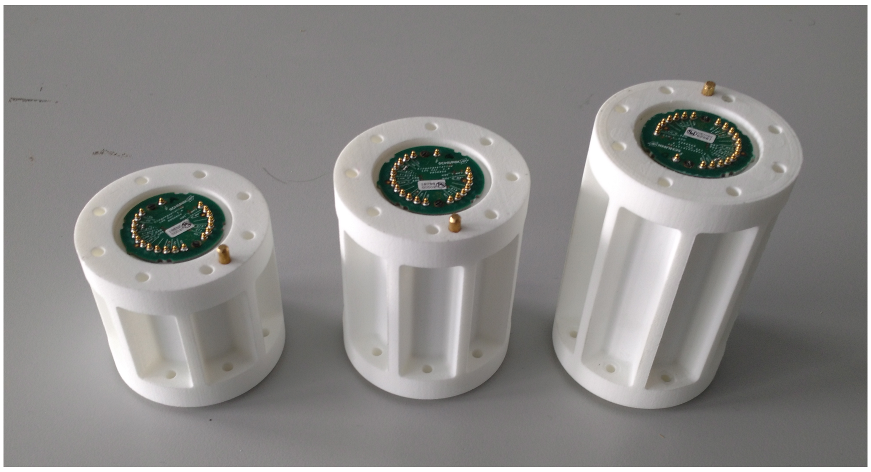


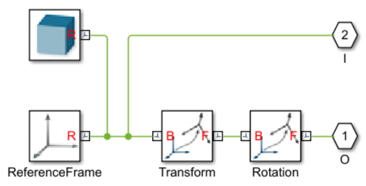
Figure B.11: Produced modules

are bought from the manufacturer SCHUNK GmbH & Co. KG and all modules are electrically tested. Although the larger extensions are possible while assembling two printed modules randomly, it is not recommended since it exceeds the maximum allowed length given in Figure B.3. Since the modules are easily mountable, the replacement procedure takes around half an hour based on where the new module is mounted.

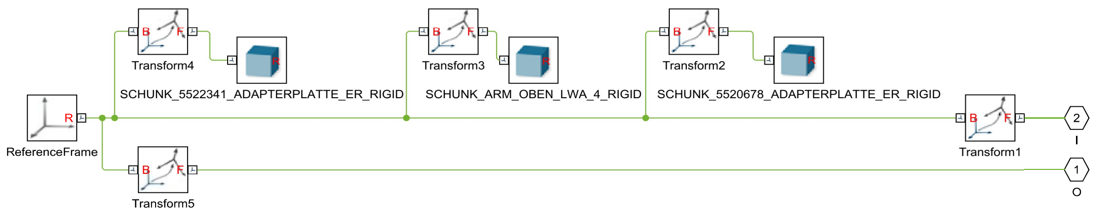
Appendix C

Visualization Tool

Various assemblies can be generated by the above-detailed sets of modules and it is not practical to implement the results obtained from our algorithms without visualizing them in the simulation environment. The use of current visualization toolboxes is limited for our case as all different assemblies cannot be defined in their libraries. Motivated by this fact, we generated our own visualization tool based on MATLAB® SimMechanics™ [123]. SimMechanics™ is a Simulink® [124] based software used for modeling and simulating 3D, multi-body, dynamic mechanical systems. The SimMechanics™ software generates the model using interconnected blocks considering their geometric and kinematic relations. It enables the user to solve the equation of motion for the whole system and automatically visualize the system. The basic idea behind the generated visualization tool is to store the module data in a custom library that has details about the kinematics of modules (length, transformations, etc.) and the Computer Aided Design (CAD) models of modules in the Standard for the Exchange of Product model data (STEP) format. We generate two different libraries for each module set in which parameters of all modules are defined. For both sets of modules, their CAD models are individually drawn in SolidWorks® [125] (for original modules from SCHUNK LWA 4P are downloaded from the company's website [35]) and they are assembled to define their mate types. The generated CAD assemblies are exported using the *SimMechanicsLink* tab and their XML (Extensible Markup Language) and STEP files are obtained automatically. The XML files include information about how the modules are assembled, spatial relationships of modules, and couplings between modules and material properties of modules. The STEP files have the information about module geometries. Subsystems including a set of blocks described in the Simulink® environment are used to define all modules. Each module is given a unique name. Each module's solid model, reference frame, transformations, and its input and output ports are defined as in Figure C.1 in a subsystem. For the definition of each component the following Simulink®



(a)



Transform4
SCHUNK_5522341_ADAPTERPLATTE_ER_RIGID
Transform3
SCHUNK_ARM_OBEN_LWA_4_RIGID
Transform2
SCHUNK_5520678_ADAPTERPLATTE_ER_RIGID
ReferenceFrame
Transform1
2
I
Transform5
1
O

(b)

Figure C.1: (a) The Simulink® model of module $L1$ in the virtual module set and (b) the Simulink® model of module $L2$ for the SCHUNK LWA 4P robot.

blocks are used:

1. *Solid* has a solid element with its geometry, inertia and color.
2. *Reference Frame* defines the local reference frame for each component and indicates how to locate the component in the robot structure.
3. *Rigid Transform* defines the translations and rotations between two sequential frames.
4. *Connection Port* defines the physical modeling connection port block for subsystems.

The model definition of $L1$ for the virtual module set and the model definition of $L2$ for the SCHUNK LWA 4P robot are given as examples in Figure C.1. All modules are defined with the above-explained Simulink® blocks and stored in a library called *ModRobLib*. A user interface is generated to start the SimMechanics™-based visualization tool which calls the *ModRobLib* library considering the user-defined task and user-defined module composition. For each simulation, this code generates the robot model and defines how the robot moves using the following Simulink® blocks:

1. *World Frame* generates a unique, motionless, orthogonal and right-handed coordinate frame for the whole mechanical model.
2. *Mechanism Configuration* defines mechanical and simulation parameters for the entire model.
3. *Block Parameters* sets the solver settings used in the simulation.

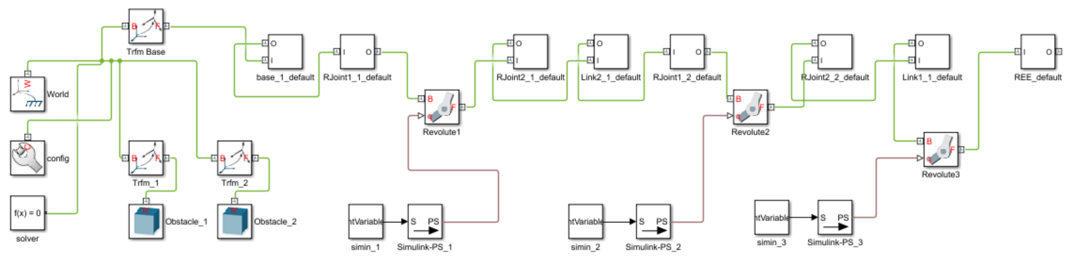


Figure C.2: The automatically generated Simulink® Model of module composition $ModRob = \{B; J1; L2; J1; L1; EE1\}$ and obstacles.

4. *Revolute Joint* represents a revolute joint movement between two sequential frames which has one-DOF. The z-axes of the previous and the following frames are considered as coincident and the joint can rotate around the x-axis and the y-axis.

5. *Prismatic Joint* represents a prismatic joint movement between two sequential frames which has one-DOF. The z-axes of the previous and the following frames are considered as coincident and the joint can move along the z-axis.

All above-mentioned features are pre-defined in the visualization tool and the tool requires the following inputs from the user to visualize the requested movement:

i) *the obstacle-laden environment*: the coordinates of obstacles and coordinates of the base module should be defined by the user.

ii) *the structure of the robot*: the robot composition in the following format: $ModRob = \{C_1; C_2; \ldots C_{end}\}$ where $ModRob$ and $C_{(\,\cdot\,)}$ represents the structure of robots and $(\,\cdot\,)^{th}$ component of the robot composition, respectively, and

iii) *trajectory*: the desired trajectory in the following form

$$TrajData = \begin{bmatrix} t_0 & t_1 & t_2 & \ldots & t_m \\ q_{1,0} & q_{1,1} & q_{1,2} & \cdots & q_{1,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{n,0} & q_{n,1} & q_{n,2} & \cdots & q_{n,m} \end{bmatrix}$$

where the variables n and m define the DOF and knots, respectively.

For the generated visualization tool, the center of the base module is automatically placed to the position $(0, 0, 0)$ and the place of obstacles must be determined with respect to this position. Then, the visualization tool sequentially adds the components of the robot from the library using user-defined $ModRob$.

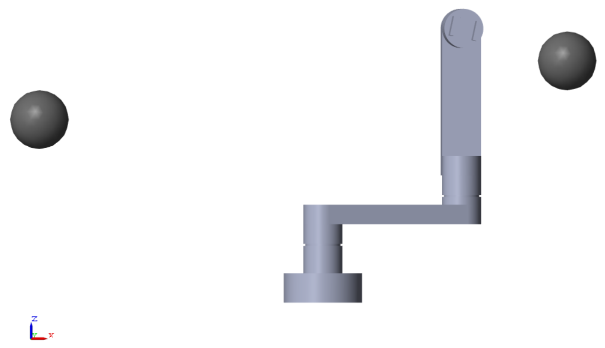


Figure C.3: The visualization of robot the same module composition and obstacles in Figure C.2.

Each component is defined with a block and lines connect these blocks considering their input and output ports. The Simulink® model for the robot composition $ModRob = \{B; J1; L2; J1; L1; EE1\}$ (from the virtual module set) with two generated obstacles in the environment and the screenshot of the same module composition and obstacles generated in visualization tool are shown in Figure C.2 and Figure C.3, respectively. Similarly, the Simulink® model for the standard assembly of SCHUNK LWA 4P robot $ModRob = \{B; PB1; L1; PB2; L2; PB3\}$ can be seen in Figure C.4.

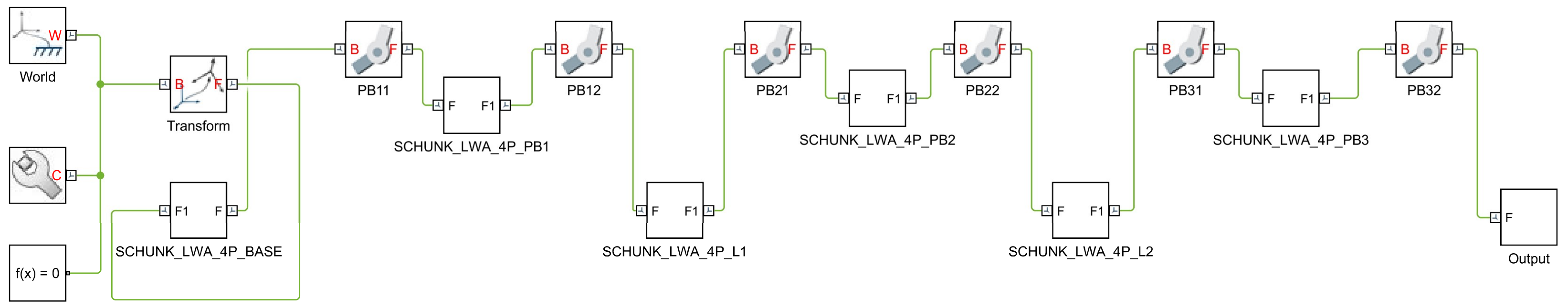


Figure C.4: The automatically generated Simulink® model of the module composition $ModRob = \{B; J1; L2; J1; L1; EE1\}$ and obstacles.

Appendix D

Extended D-H Convention

Standard D-H parameters consist of four variables, which are a_i, d_i, α_i, and θ_i. The variable a_i denotes the distance along the x_i axis between the origins of the frames o_i and $o_{i'}$, the variable d_i denotes the distance along z_{i-1} between the origins of the frames o_{i-1} and $o_{i'}$, the variable α_i denotes the angle between the axis z_{i-1} and z_i around the x_i axis, and the variable θ_i denotes the angle between the axis x_{i-1} and x_i around the z_{i-1} axis. In the method proposed in [8], the authors extend the standard D-H convention with the following parameters: a_i, α_i, p_i, n_i, and γ_i where p_i represents the distance along z axis between the origin $o_{i'}$ and joint connection PJ_{i-1} (joint between $link_i$ and $link_{i-1}$), n_i represents the distance along the z axis between o_i and the joint connection PJ_i (joint between $link_i$ and $link_{i+1}$), and γ_i is the parameter that shows the angular offset between the sequential x-axes at the time the joint is in its zero position ($q_i = 0$) (see FigureD.1). These three variables are used to determine d_i and θ_i considering the joint type as follows:

$$d_i = \begin{cases} n_{i-1} - p_i & \text{(revolute joint)}, \\ n_{i-1} - p_i + q_i & \text{(prismatic joint)}, \end{cases} \tag{D.1}$$

$$\theta_i = \begin{cases} \gamma_i + q_i & \text{(revolute joint)}, \\ \gamma_i & \text{(prismatic joint)}. \end{cases} \tag{D.2}$$

A typical joint module presented in [8] consists of three elements: an input connector, a joint, and an output connector, which are shown in Figure D.1(a). Each joint module is considered 1-DOF and is divided into distal $((\cdot)^{dl})$ and proximal $((\cdot)^{pl})$ parts as in Figure D.1(b) and Figure D.1(c), respectively. The x-axes of the input and output connectors are located along a particular direction on the connection plane and the z-axes are located towards and outwards through the input and output connectors, respectively (see Figure D.1). The y-axes of both connectors are defined considering the

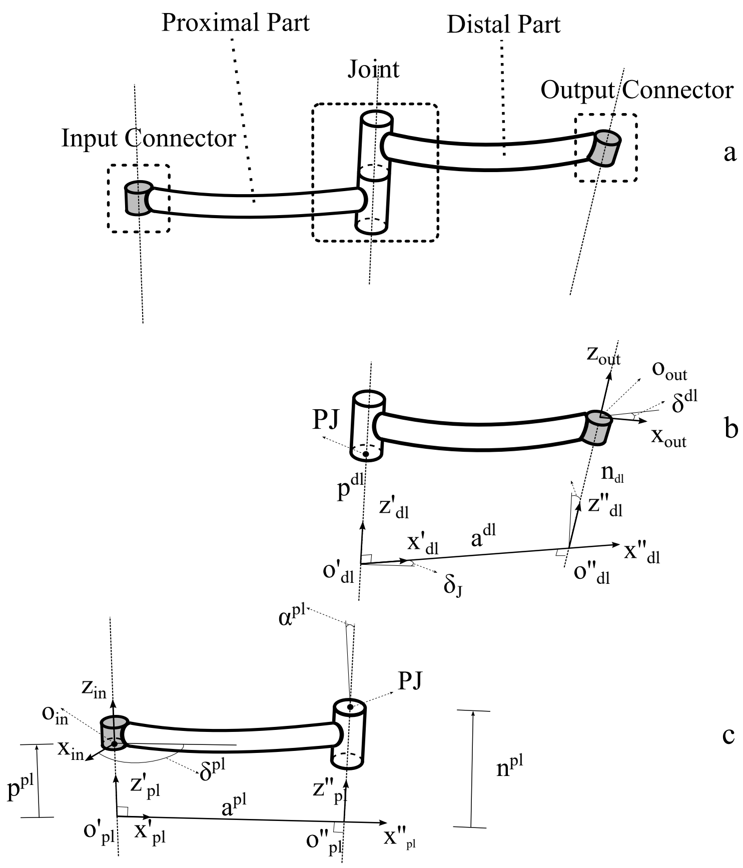


Figure D.1: The representation of kinematic notation for joint module where the connectors are indicated in light-grey color, (a) is the entire module, (b) the distal part and (c) the proximal part [8].

right-hand rule for Cartesian coordinate systems. Both proximal and distal parts are shown by four parameters $a^{(pl)}$, $\alpha^{(pl)}$, $p^{(pl)}$, $n^{(pl)}$ and $a^{(dl)}$, $\alpha^{(dl)}$, $p^{(dl)}$, $n^{(dl)}$, respectively. All of these parameters are detailed in [8] and can be seen in Figure D.1(b) and Figure D.1(c). Besides these, three more parameters namely δ^{pl}, δ^{dl}, and δ^{J} are introduced in [8]. The variable δ^{pl} represents the angle between the axes $x^{'}_{pl}$ and x_{in}, the variable δ^{dl} represents the angle between the axes $x^{''}_{dl}$ and x_{out}, and the variable δ^{J} represents the angle between $x^{''}_{pl}$ and $x^{'}_{dl}$ when the joint is in its zero position. A typical link module presented in the same approach can be seen in Figure D.2. The frames are considered similar to the distal or proximal parts of the joint modules and the details on how the axes are placed are given in [8]. The link modules are also represented with the same parameters as joints (a^{l}, α^{l}, p^{l}, n^{l}) and these parameters can be summarized as follows: a^{l} is the distance between two origins ($o_{l'}$ and $o_{l''}$) along the common normal, α^{l} is the angle between the z-axes (z_{in} and z_{out}), p^{l} is the z coordinate of the origin of the input (o_{in}) from $o^{'}$, and n^{l} is the the z coordinate of the origin of the output connection point from $o^{''}$. In addition, the angles $\delta^{l,in}$, and $\delta^{l,out}$ are defined as the angles between $x_{l'}$ and x_{in} and $x_{l''}$ and x_{out}, respectively. It should be noted that the end effector module is considered as a link module [8]. Using the above-defined parameters, the kinematic model of a robot can be generated automatically using the homogeneous transformation matrices.

The connection between the $(i-1)^{th}$ and i^{th} module, called the i^{th} connection, and the synthesis matrix $\mathbf{F}_i$ are calculated as in (D.3) where $\mathbf{F}^{'}_{i}$ is the auxiliary matrix calculated as in (D.4), the variable ϕ_i is an additional rotation around z_i and $T_m(\cdot)$, and $R_m(\cdot)$ are the homogeneous transformation and rotation matrices along/around the axis m. The details of the calculation of ϕ_i can be seen in [8].

$$\mathbf{F}_i = \mathbf{F}^{'}_{i} R_z(\phi_i) \tag{D.3}$$

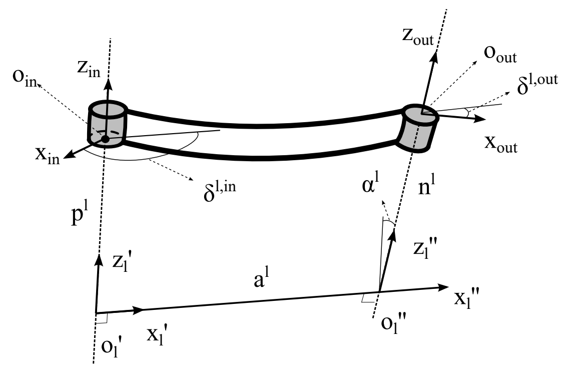


Figure D.2: The representation of kinematic notation for link modules where the light-grey colored parts are connectors [8].

$$\begin{aligned}\mathbf{F}_i^{'} =&T_z(-p_{i-1}^{dl})T_x(a_{i-1}^{dl})R_x(\alpha_{i-1}^{dl})T_z(n_{i-1}^{dl})\\ &R_z(\delta_{i-1}^{dl}-\delta_i^{pl})T_z(-p_i^{pl})T_x(a_i^{pl})R_x(\alpha_i^{pl})T_z(n_i^{pl})\end{aligned} \tag{D.4}$$

References

[1] "Reconfigurable modular manipulator system (rmms).." http://www.cs.cmu.edu/~paredis/graphics/rmms_hardware.jpg. Accessed: 2017-10-16. ix, 6

[2] "Modular reconfigurable robots (mrrs) .." http://155.69.254.10/users/risc/www/img/img-workcell-ia2.jpg. Accessed: 2017-10-16. ix, 6

[3] S. Tabandeh, "Development of novel task-based configuration optimization methodologies for modular and reconfigurable robots using multi-solution inverse kinematic algorithms," 2010. ix, 6, 7

[4] "Schunk lwa 4p modular and lightweight robot .." http://www.robotnik.es/web/wp-content/uploads/2014/04/03_000261_02_es1.jpg. Accessed: 2017-10-16. ix, 6

[5] A. Giusti, M. J. Zeestraten, E. Icer, A. Pereira, D. G. Caldwell, S. Calinon, and M. Althoff, "Flexible automation driven by demonstration: Leveraging strategies that simplify robotics," *IEEE Robotics & Automation Magazine*, vol. 25, no. 2, pp. 18–27, 2018. ix, 10, 34, 35, 36, 75

[6] G. Mesesan, E. Icer, and M. Althoff, "Hierarchical genetic path planner for highly redundant manipulators," in *Proc. of the Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*, 2015. x, 10, 59, 65, 66, 76

[7] "Schunk modular and mobile gripping systems. 100% flexibility through modularity." https://schunk.com/fileadmin/pim/docs/IM0019885.PDF. Accessed: 2017-12-20. x, xiii, 83, 84, 85

[8] A. Giusti and M. Althoff, "Automatic centralized controller design for modular and reconfigurable robot manipulators," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 3268–3275, IEEE, 2015. xi, 16, 17, 24, 99, 100, 101

[9] C.-C. Lin, "Hierarchical path planning for manipulators based on genetic algorithm with non-random initial population," *International Journal of Innovative Computing, Information and Control*, vol. 8, no. 3, pp. 1773–1786, 2012. xiii, 61, 65, 67, 70, 71, 73

[10] J. Liu, X. Zhang, and G. Hao, "Survey on research and development of reconfigurable modular robots," *Advances in Mechanical Engineering*, vol. 8, no. 8, p. 1687814016659597, 2016. 2, 5

[11] K. Stoy, D. Brandt, D. J. Christensen, and D. Brandt, *Self-reconfigurable robots: an introduction*. Mit Press Cambridge, 2010. 2

[12] M. Yim, "Modular self-reconfigurable robot systems: Challenges and opportunities for the future," *IEEE Robotics Automat. Mag.*, vol. 10, pp. 2–11, 2007. 2

[13] S. Murata and H. Kurokawa, "Self-reconfigurable robots," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 71–78, 2007. 2

[14] S. Chennareddy, A. Agrawal, and A. Karuppiah, "Modular self-reconfigurable robotic systems: A survey on hardware architectures," *Journal of Robotics*, vol. 2017, 2017. 2

[15] R. J. Alattas, S. Patel, and T. M. Sobh, "Evolutionary modular robotics: Survey and analysis," *Journal of Intelligent & Robotic Systems*, pp. 1–14, 2018. 2

[16] T. Fukuda and Y. Kawauchi, "Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 662–667, IEEE, 1990. 2

[17] T. Fukuda and S. Nakagawa, "Dynamically reconfigurable robotic system," in *International Conference on Robotics and Automation (ICRA)*, pp. 1581–1586, IEEE, 1988. 2

[18] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson, "Self-reproducing machines," *Nature*, vol. 435, no. 7039, pp. 163–164, 2005. 2

[19] V. Zykov, A. Chan, and H. Lipson, "Molecubes: An open-source modular robotics kit," in *IROS-2007 Self-Reconfigurable Robotics Workshop*, pp. 3–6, 2007. 2

[20] P. White, K. Kopanski, and H. Lipson, "Stochastic self-reconfigurable cellular robotics," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 3, pp. 2888–2893, IEEE, 2004. 2

[21] Z. Bi and W.-J. Zhang, “Concurrent optimal design of modular robotic configuration,” *Journal of Robotic systems*, vol. 18, no. 2, pp. 77–87, 2001. 2, 18

[22] Z. Bi, W. A. Gruver, W.-J. Zhang, and S. Y. Lang, “Automated modeling of modular robotic configurations,” *Robotics and Autonomous Systems*, vol. 54, no. 12, pp. 1015–1025, 2006. 2, 16

[23] “Sme definition.” `http://ec.europa.eu/growth/smes/businessfriendly-environment/sme-definition_en`. Accessed: 2018-04-11. 3

[24] E. Icer, A. Giusti, and M. Althoff, “A task-driven algorithm for configuration synthesis of modular robots,” in *International Conference on Robotics and Automation (ICRA)*, pp. 5203–5209, IEEE, 2016. 4, 10, 55, 56, 68, 75

[25] J.-O. Kim and P. K. Khosla, “Design of space shuttle tile servicing robot: an application of task based kinematic design,” in *Proceedings International Conference on Robotics and Automation*, pp. 867–874, IEEE, 1993. 4

[26] Q. Li and J. Zhao, “A universal approach for configuration synthesis of reconfigurable robots based on fault tolerant indices,” *Industrial Robot: An International Journal*, vol. 39, no. 1, pp. 69–78, 2012. 5, 41

[27] J. Neumann, A. W. Burks, *et al.*, *Theory of self-reproducing automata*. University of Illinois Press Urbana, 1966. 5

[28] S. Donald and K. Pradeep, “The cmu reconfigurable modular manipulator systems,” in *International Symposium and Exposition on Robots*, pp. 473–488, 1988. 5, 7

[29] D. B. Stewart, R. A. Volpe, and P. K. Khosla, “Design of dynamically reconfigurable real-time software using port-based objects,” *IEEE Transactions on software engineering*, vol. 23, no. 12, pp. 759–776, 1997. 5

[30] I. M. Chen, *Theory and applications of modular reconfigurable robotic systems*. PhD thesis, California Institute of Technology, 1994. 5, 7

[31] S. Tabandeh, C. Clark, and W. Melek, “Task-based configuration optimization of modular and reconfigurable robots using a multi-solution inverse kinematics solver,” *Toronto, Canada*, 2007. 6, 14, 15, 17, 21

[32] T. Matsumaru, "Design and control of the modular robot system: Tomms," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 2, pp. 2125–2131, IEEE, 1995. 6, 7

[33] H. Ulbrich, J. Baur, J. Pfaff, and C. Schuetz, "Design and realization of a redundant modular multipurpose agricultural robot," in *Proceedings of the XVII International Symposium on Dynamic Problems of Mechanics (DINAME), Natal, Brazil*, 2015. 6, 7

[34] C. Schütz, J. Pfaff, J. Baur, T. Buschmann, H. Ulbrich, and G. Idea, "A modular robot system for agricultural applications," in *Proceedings International Conference of Agricultural Engineering, Zurich*, 2014. 6, 7

[35] "Schunk mobile gripping systems." `http://www.schunk-modular-robotics.com/en/content/home/products.html`. Accessed: 2017-12-18. 7, 84, 93

[36] "igus® official website - modular robotic components robolink®." `https://www.igus.com/info/robotic-arm-and-components`. Accessed: 2019-09-01. 7

[37] "e.do people learn robotics e.do." `https://edo.cloud/`. Accessed: 2019-09-01. 7

[38] J. J. Craig, "Introduction to robotics. mechanics and control. series in electrical and computer engineering: Control engineering," *ed: Addison-Wesley, Reading, MA, USA*, 1989. 9, 86

[39] E. Icer and M. Althoff, "Cost-optimal composition synthesis for modular robots," in *Conference on Control Applications (CCA)*, pp. 1408–1413, IEEE, 2016. 10, 76

[40] E. Icer, H. A. Hassan, K. El-Ayat, and M. Althoff, "Evolutionary cost-optimal composition synthesis of modular robots considering a given task," in *International Conference on Intelligent Robots and Systems (IROS)*, pp. 3562–3568, IEEE, 2017. 10, 40, 76

[41] G. Mesesan, M. A. Roa, E. Icer, and M. Althoff, "Hierarchical path planner using workspace decomposition and parallel task-space rrts," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, IEEE, 2018. 10, 76

[42] H. Al-Dois, A. Jha, and R. Mishra, "Task-based design optimization of serial robot manipulators," *Engineering Optimization*, vol. 45, no. 6, pp. 647–658, 2013. 14

[43] S. Patel and T. Sobh, "Task based synthesis of serial manipulators," *Journal of advanced research*, vol. 6, no. 3, pp. 479–492, 2015. 14

[44] S. H. Patel and T. M. Sobh, “Using task descriptions for designing optimal task specific manipulators,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 3544–3551, IEEE, 2015. 14

[45] B. Rout and R. Mittal, “Optimal manipulator parameter tolerance selection using evolutionary optimization technique,” *Engineering Applications of Artificial Intelligence*, vol. 21, no. 4, pp. 509–524, 2008. 14

[46] I.-M. Chen and J. W. Burdick, “Enumerating the nonisomorphic assembly configurations of modular robotic systems,” in *Proceedings International Conference on Robotics and Automation*, pp. 1985–1992, IEEE/RSJ, 1993. 14, 15, 16, 21

[47] Z. M. Bi, Y. Lin, and W. Zhang, “The general architecture of adaptive robotic systems for manufacturing applications,” *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 5, pp. 461–470, 2010. 14, 15, 21

[48] L. Xuan, Z. Minglu, and L. Wei, “Methods to modular robot design,” in *2008 Second International Symposium on Intelligent Information Technology Application*, vol. 1, pp. 663–668, IEEE, 2008. 14

[49] B. Dong and Y. Li, “Multi-objective-based configuration generation and optimization for reconfigurable modular robot,” in *International Conference on Information Science and Technology*, pp. 1006–1010, IEEE, 2011. 15, 42

[50] C. Leger and J. Bares, “Automated synthesis and optimization of robot configurations,” in *ASME Design Engineering Technical Conferences*, 1998. 15, 40

[51] R. W. Brockett, “Robotic manipulators and the product of exponentials formula,” in *Mathematical theory of networks and systems*, pp. 120–129, Springer, 1984. 16

[52] I.-M. Chen and G. Yang, “Configuration independent kinematics for modular robots,” in *Proceedings International Conference on Robotics and Automation*, vol. 2, pp. 1440–1445, IEEE, 1996. 16

[53] I.-M. Chen and G. Yang, “Automatic model generation for modular reconfigurable robot dynamics,” *Journal of dynamic systems, measurement, and control*, vol. 120, no. 3, pp. 346–352, 1998. 16

[54] X. Pan, H. Wang, and Y. Jiang, “Research on the kinematic calibration of a modular reconfigurable robot,” in *International Conference on Mechatronics and Automation (ICMA)*, pp. 562–568, IEEE, 2013. 16

[55] J. Denavit, “A kinematic notation for lower-pair mechanisms based on matrices,” *ASME J. Appl. Mech.*, pp. 215–221, 1955. 16

[56] Z. Bi, W. Zhang, I.-M. Chen, and S. Lang, “Automated geneartion of the d–h parameters for configuration design of modular manipulators,” *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 5, pp. 553–562, 2007. 16

[57] L. Kelmar and P. Khosla, “Automatic generation of kinematics for a reconfigurable modular manipulator system,” in *International Conference on Robotics and Automation (ICRA)*, pp. 663–668, IEEE, 1988. 16

[58] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer Science & Business Media, 2008. 17

[59] L. Kelmar and P. K. Khosla, “Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system,” *Journal of Field Robotics*, vol. 7, no. 4, pp. 599–619, 1990. 17

[60] I.-M. Chen, G. Yang, and I.-G. Kang, “Numerical inverse kinematics for modular reconfigurable robots,” *Journal of Robotic Systems*, vol. 16, no. 4, pp. 213–225, 1999. 17

[61] W. Wu, Y. Guan, H. Li, M. Su, H. Zhu, X. Zhou, and H. Zhang, “Task-oriented inverse kinematics of modular reconfigurable robots,” in *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pp. 1187–1192, IEEE, 2013. 17

[62] S. Tabandeh, C. M. Clark, and W. Melek, “A genetic algorithm approach to solve for multiple solutions of inverse kinematics using adaptive niching and clustering,” *Computer Science and Software Engineering*, p. 63, 2006. 17

[63] A. K. Dash, I.-M. Chen, S. H. Yeo, and G. Yang, “Task-based configuration design for 3-legged modular parallel robots using simplex methods,” in *International Symposium on Computational Intelligence in Robotics and Automation*, vol. 2, pp. 998–1003, IEEE, 2003. 17, 19

[64] I.-M. Chen and J. W. Burdick, "Determining task optimal modular robot assembly configurations," in *Proceedings International Conference on Robotics and Automation*, pp. 132–137, IEEE, 1995. 17, 40

[65] J.-O. Kim and P. Khosla, "A multi-population genetic algorithm and its application to design of manipulators," pp. 279–286, 1997. 17

[66] C. J. Paredis and P. Khosla, "Synthesis methodology for task based reconfiguration of modular manipulator systems," 1993. 17, 18

[67] O. Chocron and P. Bidaud, "Genetic design of 3d modular manipulators," in *International Conference on Robotics and Automation*, vol. 1, pp. 223–228, IEEE, 1997. 17, 40, 41, 50, 52, 54, 55

[68] C. J. Paredis and P. K. Khosla, "Kinematic design of serial link manipulators from task specifications," *The International Journal of Robotics Research*, vol. 12, no. 3, pp. 274–287, 1993. 18

[69] C. J. Paredis and P. Khosla, "Design of modular fault tolerant manipulators," 1995. 18

[70] C. J. Paredis, H. B. Brown, and P. K. Khosla, "A rapidly deployable manipulator system," *Robotics and Autonomous Systems*, vol. 21, no. 3, pp. 289–304, 1997. 18

[71] C. J. Paredis and P. K. Khosla, "Task-based design of manipulators: An agent-based approach," in *Proceedings of ASME Design Engineering Technical Conference*, pp. 14–17, 1997. 18

[72] S. Farritor, S. Dubowsky, N. Rutman, and J. Cole, "A systems-level modular design approach to field robotics," in *IEEE International Conference on Robotics and Automation*, vol. 4, pp. 2890–2895, IEEE, 1996. 18, 40

[73] S. Farritor and S. Dubowsky, "On modular design of field robotic systems," *Autonomous Robots*, vol. 10, no. 1, pp. 57–65, 2001. 18

[74] A. K. Dash, I.-M. Chen, S. H. Yeo, and G. Yang, "Task-oriented configuration design for reconfigurable parallel manipulator systems," *International Journal of Computer Integrated Manufacturing*, vol. 18, no. 7, pp. 615–634, 2005. 19

[75] A. Valente, "Reconfigurable industrial robots: A stochastic programming approach for designing and assembling robotic arms," *Robotics and Computer-Integrated Manufacturing*, vol. 41, pp. 115–126, 2016. 19

[76] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi, “Minimum cost trajectory planning for industrial robots,” *European Journal of Mechanics-A/Solids*, vol. 23, no. 4, pp. 703–715, 2004. 22

[77] S. Chiaverini, “The unit quaternion: A useful tool for inverse kinematics of robot manipulators,” *Syst. Anal. Model. Simulation*, vol. 35, pp. 45–60, 1999. 24, 25

[78] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005. 26, 28, 44, 60, 69

[79] A. Shkolnik and R. Tedrake, “Path planning in 1000+ dimensions using a task-space voronoi bias,” in *IEEE International Conference on Robotics and Automation*, pp. 2061–2067, 2009. 26, 61, 69

[80] A. P. Del Pobil, M. A. Serna, and J. Llovet, “A new representation for collision avoidance and detection,” in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pp. 246–251, IEEE, 1992. 26

[81] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006. 28, 44, 52, 54

[82] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, “Fast direct multiple shooting algorithms for optimal robot control,” in *Fast motions in biomechanics and robotics*, pp. 65–93, Springer, 2006. 29

[83] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *IEEE International Conference onRobotics and Automation*, vol. 2, pp. 995–1001, 2000. 32

[84] M. Kelly, “An introduction to trajectory optimization: How to do your own direct collocation,” *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017. 33

[85] A. G. Billard, S. Calinon, and R. Dillmann, “Learning from humans,” in *Springer handbook of robotics*, pp. 1995–2014, Springer, 2016. 34

[86] L. Davis, “Handbook of genetic algorithms,” 1991. 40, 45

[87] P. Moscato *et al.*, “On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms,” *Caltech concurrent computation program, C3P Report*, vol. 826, p. 1989, 1989. 40

[88] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, vol. 2, pp. 1470–1477, IEEE, 1999. 40

[89] D. Karaboga and C. Ozturk, "A novel clustering approach: Artificial bee colony (abc) algorithm," *Applied soft computing*, vol. 11, no. 1, pp. 652–657, 2011. 40

[90] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Citeseer, 1995. 40

[91] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 210–214, IEEE, 2009. 40

[92] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen, *et al.*, *Evolutionary algorithms for solving multi-objective problems*, vol. 5. Springer, 2007. 40, 45

[93] G. Yang and I.-M. Chen, "Task-based optimization of modular robot configurations: minimized degree-of-freedom approach," *Mechanism and machine theory*, vol. 35, no. 4, pp. 517–540, 2000. 40

[94] O. Chocron, "Evolutionary design of modular robotic arms," *Robotica*, vol. 26, no. 03, pp. 323–330, 2008. 40, 41, 42

[95] J. B. Pollack and H. Lipson, "The golem project: Evolving hardware bodies and brains," in *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, pp. 37–42, IEEE, 2000. 40

[96] J. Lemay and L. Notash, "Configuration engine for architecture planning of modular parallel robots," *Mechanism and Machine Theory*, vol. 39, no. 1, pp. 101–117, 2004. 40, 42

[97] W. Gao, H. Wang, Y. Jiang, and X. Pan, "Task-based configuration synthesis for modular robot," in *International Conference on Mechatronics and Automation (ICMA)*, pp. 789–794, IEEE, 2012. 41

[98] W. K. Chung, J. Han, Y. Youm, and S. Kim, "Task based design of modular robot manipulator using efficient genetic algorithm," in *Proceedings International Conference on Robotics and Automation*, vol. 1, pp. 507–512, IEEE, 1997. 42

[99] T. Bäck, D. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997. 46, 49

[100] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Springer Science & Business Media, 2007. 46

[101] T. Yoshikawa, "Manipulability of robotic mechanisms," *The international journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985. 50

[102] T. Lozano-Perez, "A simple motion-planning algorithm for general robot manipulators," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 3, pp. 224–238, 1987. 60

[103] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*, pp. 396–404, Springer, 1986. 60

[104] O. Takahashi and R. J. Schilling, "Motion planning in a plane using generalized voronoi diagrams," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 2, pp. 143–150, 1989. 60

[105] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 2, pp. 224–241, 1992. 60

[106] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 5, pp. 1179–1187, 1989. 60

[107] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. 60

[108] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*, pp. 43–57, Springer, 2004. 60

[109] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006. 60, 69

[110] F. Fahimi, "Autonomous robots," *Modeling, Path Planning and Control*, 2009. 60

[111] J. Yu and P. Müller, "An on-line cartesian space obstacle avoidance scheme for robot arms," *Mathematics and Computers in Simulation*, vol. 41, no. 5, pp. 627–637, 1996. 60

[112] R. Menasri, A. Nakib, B. Daachi, H. Oulhadj, and P. Siarry, "A trajectory planning of redundant manipulators based on bilevel optimization," *Applied Mathematics and Computation*, vol. 250, pp. 934–947, 2015. 60

[113] C.-C. Lin and J.-H. Chuang, "A potential-based path planning algorithm for hyper-redundant manipulators," *Journal of the Chinese Institute of Engineers*, vol. 33, no. 3, pp. 415–427, 2010. 60

[114] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1874–1879, IEEE, 2006. 61

[115] M. V. Weghe, D. Ferguson, and S. S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pp. 477–482, IEEE, 2007. 61

[116] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm:follow the gap method," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012. 62

[117] C. Lin, P. Chang, and J. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial robots," *IEEE Transactions on Automatic Control*, vol. 28, no. 12, pp. 1066–1074, 1983. 64

[118] A. Alexandrov, "Basic concepts and simplest properties of convex polyhedra," *Convex Polyhedra*, pp. 7–86, 2005. 65

[119] "Schunk parallel gripping system." `https://schunk.com/de_en/gripping-systems/series/mpg/`. Accessed: 2017-12-18. 85

[120] "Eos e-manufacturing solutions, material data center." `http://eos.materialdatacenter.com/eo/de`. Accessed: 2017-10-18. 85, 90

[121] "Maryland metrics." `https://mdmetric.com/fastindx/TI-168.pdf`. Accessed: 2017-02. 89

[122] "Engineering-abc, tightening torque to preload a bolt." `www.tribology-abc.com/calculators/e3_6a.htm`. Accessed: 2017-10-18. 90

[123] "Simmechanics multibody - matlab-simulink." `https://de.mathworks.com/products/simmechanics.html`. Accessed: 2016-08-11. 93

[124] "Matlab-simulink." `https://de.mathworks.com/products/simulink.html`. Accessed: 2016-08-11. 93

[125] "Solidworks." `https://www.solidworks.com/`. Accessed: 2016-08-11. 93