Robot Trajectory Optimization for Relaxed Effective Tasks

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Ing. Sergey Alatartsev
geb. am 08.12.1988 in Chita, USSR

Gutachterinnen/Gutachter

Prof. Dr. Frank Ortmeier
Prof. Dr. Dmitry Berenson
Prof. Dr. Iacopo Gentilini
Prof. Dr. Nikos Aspragathos

Magdeburg, den 07.07.2015

Alatartsev, Sergey:
Robot Trajectory Optimization for Relaxed Effective Tasks
Dissertation, University of Magdeburg, 2015.

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, 07.07.2015

Sergey Alatartsev

Zusammenfassung

Industrielle Roboter sind flexible Maschinen, die aktuell in den verschiedensten Produktionsbereichen eingesetzt werden. Ihr Arbeitsablauf besteht hauptsächlich aus zwei abwechselnden Phasen. Die erste Phase berechnet effektive Bewegungen, die erforderlich sind, um eine Aufgabe auszuführen, wie z.B. Schweißen. In der zweiten Phase werden unterstützende Bewegungen bestimmt, welche für die Bewegung zwischen zwei Aufgaben benötigt werden. Insbesondere in der ersten Phase ist es jedoch möglich, dass z.B. das Werkzeug des Roboters einen gewissen Abstands- oder Winkelspielraum während des Schweißens haben darf. Diese Freiheit wird häufig vernachlässigt und Roboter sind manuell nach Intuition des Programmierers programmiert. Dennoch kann dieser Spielraum als ein zusätzlicher Freiheitsgrad für die Optimierung der Robotertrajektorie verwendet werden. In dieser Arbeit stellen wir eine Formalisierung dieser Freiheit für effektive Aufgaben vor. Wir bezeichnen eine effektive Aufgabe mit einer formalisierten Ausführungsfreiheit als eine gelockerte effektive Aufgabe.

Eine unendliche Anzahl an Möglichkeiten zu haben, eine Aufgabe auszuführen, lässt verschiedene Forschungsfragen aufkommen: (i) wie ist eine Eingangspunktsequenz für gelockerte effektive Aufgaben zu optimieren? (ii) wie sind Anfangskonfigurationen des Roboters für diese Aufgaben zu finden? (iii) wie ist eine Roboter-Trajektorie für eine bestimmte gelockerte Aufgabe zu optimieren? Wir stellen ein Konzept vor, welches die drei Problemstellungen in drei voneinander separaten Komponenten zu lösen vermag. In Kombination miteinander oder mit anderen dem Stand der Technik entsprechenden Ansätzen angewendet, ist die Berechnung der optimierten Roboter-Trajektorie möglich.

Die erste Komponente betrachtet das Problem, eine Sequenz für effektive Aufgaben und ihre Eingangspunkte zu finden. Dieses Problem ist als “Traveling Salesman Problem with Neighborhoods” (TSPN) bekannt, bei dem eine Tour durch eine Menge an Bereichen gefunden werden soll. Wir stellen “Constricting Insertion Heuristics” für die Konstruktion einer Tour und “Constricting 3-Opt” für die Optimierung der Tour vor. In der zweiten Komponente muss der Bewegungspfad angepasst und Anfangskonfigurationen für den Roboter gesucht werden. Dieses Problem ist als “Touring-a-Sequence-of-Polygons Problem” (TPP) bekannt, bei dem eine Tour durch eine gegebene Sequenz von Bereichen gefunden werden soll. Wir stellen eine Modifikation des “Rubber-Band Algorithmus” (RBA) vor und bezeichnen diese Erweiterung als “Nested RBA”. Die Optimierung von Robotertrajektorien in der dritten Komponente ist ebenfalls als TPP dargestellt. Dennoch stellen wir im Gegensatz zum klassischen RBA, bei dem Bereiche durch eine Polylinie beschränkt sind, eine Erweiterung des RBA namens “Smoothed RBA” vor, bei dem Bereiche durch eine glatte Kurve eingeschränkt sind, welche zu einer minimalen Kosten-Robotertrajektorie führt.

Abstract

Industrial robots are flexible machines that are currently involved in multiple production domains. Mainly their workflow consists of two alternating stages. The first stage is effective movements that are required to perform a task, e.g., welding a seam. The second stage is supporting movements that are needed to move from one effective task to another, e.g., movements between welding seams. Many effective tasks allow a certain freedom during their execution, e.g., the robot's tool might have a certain deviation during welding. This freedom is often ignored and robots are programmed manually based on the programmer's intuition. Nonetheless, this freedom can be used as an extra degree of freedom for robot trajectory optimization. In this thesis, we propose a formalization of this freedom for effective tasks. We refer to an effective task with a formalized freedom of execution as a relaxed effective task.

Having an infinite number of ways to execute a task raises several research questions: (i) how to optimize a sequence of entry points for relaxed effective tasks? (ii) how to find starting robot configurations for these tasks? (iii) how to optimize a robot trajectory for a certain relaxed task? We propose a solution concept that decomposes a problem containing all three questions into three components that can be applied in combination with each other or with other state-of-the-art approaches.

The first component considers the problem of finding a sequence of effective tasks and their entry points. This problem is modeled as the Traveling Salesman Problem with Neighborhoods (TSPN) where a tour has to be found through a set of areas. We propose a Constricting Insertion Heuristic for constructing a tour and a Constricting 3-Opt for improving the tour. In the second component, the problem of adapting a tour for a robot to execute and searching for starting robot configurations is modeled as a Touring-a-sequence-of-Polygons Problem (TPP) where a tour has to be found through a given sequence of areas. We propose a modification of the Rubber-Band Algorithm (RBA). We refer to this extension as a Nested RBA. Optimization of a robot trajectory in the third component is also represented as a TPP. However, in contrast to the classic RBA where areas are constricted with a polyline, we propose an extension of the RBA called Smoothed RBA where areas are constricted with a smooth curve which leads to a minimal cost robot trajectory.

Acknowledgements

This thesis would not be possible without many people. Firstly, I would like to express my deepest gratitude to supervisor Prof. Frank Ortmeier who gave me a chance to work in his department and provided the conditions in which it was possible to finish this project. I would like to thank my reviewers and committee members: Prof. Nikos Aspragathos, Prof. Dmitry Berenson, Prof. Iacopo Gentilini, Prof. Rudolf Kruse, Prof. Stefan Schirra, and Jun.-Prof. Sebastian Zug for the time they invested in this thesis.

During this PhD project, I had the honor of working with many great people. It is not enough space to list everyone, therefore, please excuse me if you cannot find your name here. It was a pleasure to work with Marcus Augustine, who shared many stories about Australia with me and introduced me to Flight of the Conchords. It was great to share the office with Tanja Hebecker, who always gave me some time to shutdown the computer. Matthias Güdemann supported me greatly at the very beginning of my PhD and was very patient when correcting my first paper. Although he graduated soon, he continued to support me throughout the whole PhD study and we had a lot of fun during our cross-Russia travel. I want to express my deepest gratitude to Michael Lipaczewski who helped me a lot in difficult life situations and proved to be a very faithful friend and a very kind man. Davai-Davai, Michael! If there would exist a best secretary award, I would definitely give it to Marianne Schulze who used her expertise to make my life much easier. Also, I would like to thank Anton Belov for a valuable help with trajectory optimization, Sebastian Stellmacher for his contribution to sequencing approaches analysis, Kelsey Elder for her comments on the thesis. I thank Anton Ivanov for the fun we had, and for his a very special point of view on life, which heavily influenced my attitude to many things. Going to pre-PhD time, I would like to express my gratitude to the people who taught me to express myself – Marina Morozova, to work – Dmitry Makarov and to think – Valery Gordon. All these people enriched my life in the past years.

And finally, very special thanks go to my parents Oleg and Svetlana who encouraged and supported me throughout all my life. Their influence on me is hard to overestimate, as well as the one of my partner Vera Mersheeva. Besides, she was always there when I needed her, she has made 3278 corrections to the thesis. I feel extremely grateful for the hours of thesis discussions and support. At last, I am also grateful to the shower at my apartment, as the vast majority of ideas I got there.

Contents

1. Introduction

Please do not shoot the pianist.
He is doing his best.

Oscar Wilde

Industrialized countries with high labor costs have to rely on production automation to keep their competitive advantage. One of the most flexible and powerful automation technologies available today is industrial robotics. Equipped with the right tool, standardized industrial robots can perform numerous production tasks. Since acquisition and programming of an industrial robot are very expensive, the feasibility of using robots in production facilities depends on the efficiency with which the robot can perform its task. The more production steps a robot can perform in a given time interval, the higher the production rates and, as a result, the faster the robot can compensate for its initial acquisition and programming costs, and the higher the competitive advantage it provides to the company. Therefore, robot trajectory optimization is one of the most important problems in industrial robotics.

Virtually all robotic scenarios consist of two types of robotic movements. The first category includes movements that are specifically required for a job, e.g., welding a seam, deburring a sharp edge or cutting a shape. Typically these are the movements during which tools (e.g., a welding torch) are applied. We call this category *effective movements* or *effective tasks*. Another category – *supporting movements* or *supporting tasks* – are the movements between effective tasks. Supporting movements are not directly needed for a given job. However, they are necessary to sequence one effective movement after another. For example, in seam welding, a supporting movement would be to move the robot from one welding seam to another. An example of a simple welding application is depicted in Figure 1.1. The two welding seams – (2) and (4) – are effective movements, whereas (1), (3) and (5) are supporting movements, which are only necessary to execute the effective movements.

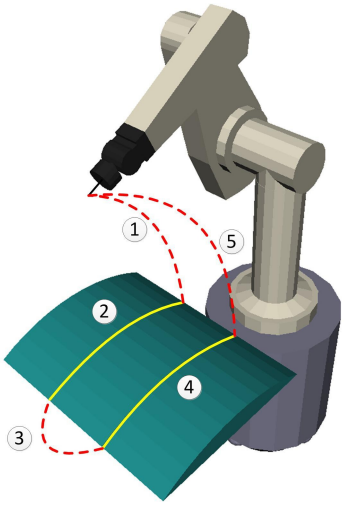

Figure 1.1: Example of the alternating effective and supporting movements

The major characteristics that affect the efficiency of a given robot are how fast the robot can perform its effective tasks (effective movements), and how long it takes the robot to move between effective tasks (supporting movements).

Conventional industrial programming can be compared to public tram connections. The tram infrastructure is created in the existing streets and often consists of straight and circular segments. Path smoothing depends on the tram specification and the environment. The path that a robot's tool has to follow is also constructed from the pattern pieces, e.g., linear or point-to-point movements. The connection points between these pieces, e.g., point P_2 in Figure 1.2, are smoothed by manually inserting intermediate nodes, i.e., points P_1 and P_3 in Figure 1.2. The robot has to strictly follow the defined geometry of the task similar to a tram that follows its rails.

The means of transportation have made a huge step forward, which is currently used in entertainment sporting events. The most famous example is Formula 1 racing. In this case, the time it takes to finish a track is important. Drivers apply known techniques to make turns as efficiently as possible, e.g., at which point of time and space to decelerate or accelerate. Unlike trams, a racing car makes full use of the whole width of the track, see Figure 1.2.

At first sight, many tasks that a robot has to perform, e.g., welding or cutting a line segment or a closed-contour, are similar to tram rails, as they have strictly defined geometry. However, that is misleading. Industrial tasks often allow a certain freedom during execution. For example, the orientation of the knife can have a certain deviation during cutting. For spot drilling, orientation about the drilling axis is not important. Due to the possibility of multiple ways of execution, industrial robot effective tasks are more similar to racing tracks. Thus, we refer to the effective tasks that allow a certain freedom for execution as relaxed effective tasks. However, this freedom is often ignored and the robot path is chosen based on the programmer's intuition [79][19].

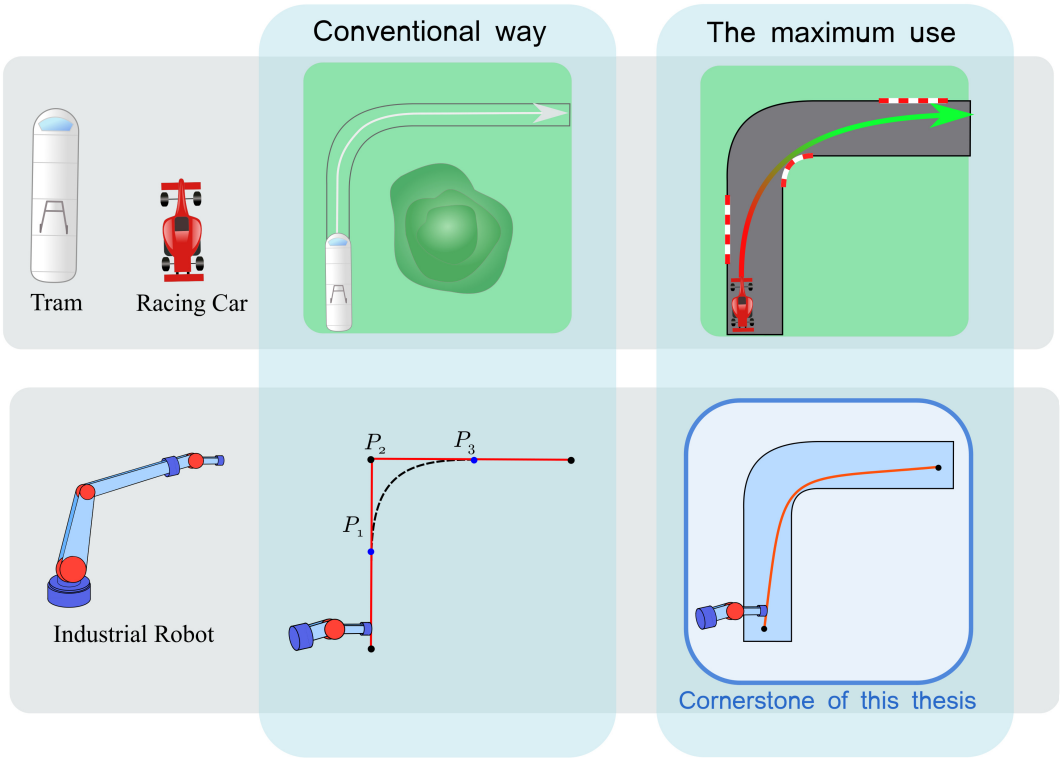

Figure 1.2: Comparison of robot programming with means of transportation

Building a near-optimal trajectory for an industrial robot to perform a relaxed effective task is a challenging problem. The first reason is that a relaxed effective task often represents 6D volume that has to be followed by the tool. In addition, for a typical industrial robot, there are often up to 8 different kinematic solutions to reach a certain pose of its tool. Furthermore, a robot usually has to perform not just a single effective task, but rather a set of them. Therefore, one also has to find their sequence and the entry points of each task. This leads to large search space that is time consuming to explore.

The task relaxation raises multiple research problems. In this thesis, we answer the following two main questions:

- Given: a set of relaxed effective tasks. How to find a sequence of entry points for the tasks such that it has the minimal trajectory cost of the supporting movements?
- Given: a single relaxed effective task. How to find a path through the task for a robot's tool such that it has the minimal robot trajectory cost?

Summarizing, the goal of the thesis is to develop algorithms to optimize robot trajectory by making use of the freedom for task execution. This will increase the effectiveness of industrial robots and can even enable the use of industrial robots in new

manufacturing processes where the costs of human labor are currently lower than or equal to those of robots. It would reduce production costs, and thus give a competitive advantage.

1.1 Contribution

This dissertation presents approaches for optimization of effective and supporting robot trajectories. The main contributions are the following:

- Tour construction approach "Constricting Insertion Heuristic" for the Traveling Salesman Problem with Neighborhoods for constructing an initial tour. A tour is a sequence of effective tasks and their entry positions.
- Tour improvement approach "Constricting 3-Opt" for the Traveling Salesman Problem with Neighborhoods that takes an initial tour as input and iteratively improves it.
- Decomposition approach to optimize entry-points of a given relaxed effective task sequence using robot trajectory costs.
- Approach to optimize robot trajectory for a given effective task based on exploiting the freedom in the robot end-effector path.

1.2 Publication Note

Part of the work presented in this thesis has already been published. The concept of robot trajectory description that allows multiple ways of execution was published in [5]. The analysis and categorization of state-of-the-art approaches were presented in [9]. The ideas of involving extra degrees of freedom available from the tasks were first presented in [3]. Tour-construction and tour-improvement heuristics presented in Chapter 4 were published in [6] and [7], respectively. Robot tour adaptation approach from Chapter 5 first appeared in [4]. The problem of trajectory optimization for the end-effector path that allows the freedom of execution in Chapter 6 was covered in [8]. The stated publications are listed below:

- **2015**:

  [9] Robotic Task Sequencing Problem: A Survey. Sergey Alatartsev, Sebastian Stellmacher and Frank Ortmeier. In Journal of Intelligent & Robotic Systems, Springer, 2015.

- **2014**

  [4] Improving the Sequence of Robotic Tasks with Freedom of Execution. Sergey Alatartsev and Frank Ortmeier. In Proceedings of the International Conference on Intelligent Robots and Systems (**IROS**), USA, 2014

  [8] Robot Trajectory Optimization for the Relaxed End-Effector Path. Sergey Alatartsev, Anton Belov, Mykhaylo Nykolaychuk and Frank Ortmeier. In Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics (**ICINCO**), Austria, 2014

- **2013**

  [6] Constricting Insertion Heuristic for Traveling Salesman Problem with Neighborhoods. Sergey Alatartsev, Marcus Augustine and Frank Ortmeier. In Proceedings of the 23rd International Conference on Automated Planning and Scheduling, (**ICAPS**), Italy, 2013

  [7] On Optimizing a Sequence of Robotic Tasks. Sergey Alatartsev, Vera Mersheeva, Marcus Augustine and Frank Ortmeier. In Proceedings of the International Conference on Intelligent Robots and Systems (**IROS**), Japan, 2013

  [3] Path planning for industrial robots among multiple under-specified tasks. Sergey Alatartsev and Frank Ortmeier. In Proceedings of the Magdeburger-Informatik-Tage 2. Doktorandentagung (MIT), Germany, 2013
  Best-Upcoming-PhD-Thesis Award 2013

- **2012**

  [5] Trajectory Description Conception for Industrial Robots. Sergey Alatartsev, Matthias Güdemann and Frank Ortmeier. In Proceedings of the 7th German Conference on Robotics (**ROBOTIK**), Germany, 2012

2. Problem Specification

I keep six honest serving-men
(They taught me all I knew);
Their names are What and Why and When
And How and Where and Who.

Rudyard Kipling

In this chapter, we provide the background information required for this thesis. This includes both frequently used terms, e.g., robotic spaces, kinematics and trajectory definition, as well as proposed formalization of the tasks with freedom. We illustrate the thesis objectives using the example from the cutting-deburring domain. Then a solution concept and research problems are defined. As involving many degrees-of-freedom causes a large search space that is computationally expensive to explore, we have to set the scope of the thesis. To that end, we present assumptions that make a solution feasible.

Formal definitions of robotic spaces, kinematics and formalization of task freedom are presented in Section 2.1. Objectives that are covered in this thesis are discussed in Section 2.2. The thesis problem is stated in Section 2.3. In Section 2.4, we present a decomposition concept and possible solution strategies as well as sub-problem formalization. The solution conception works under several assumptions that are discussed in Section 2.5.

2.1 Formal Definitions

2.1.1 Task and Configuration Spaces

An industrial robot consists of a kinematic chain, i.e., a sequence of connected links. The last link of a kinematic chain is called end-effector. An end-effector can be a drill, brush, camera, etc. An end-effector is often specified with a point called tool center point (TCP), see Figure 2.1. For example, the tip of a drill or the focus point of a camera are TCPs. We refer to the point where the end-effector is attached to the robot as end of arm (EOA). Further, by referring to end-effector, we will refer to the TCP.

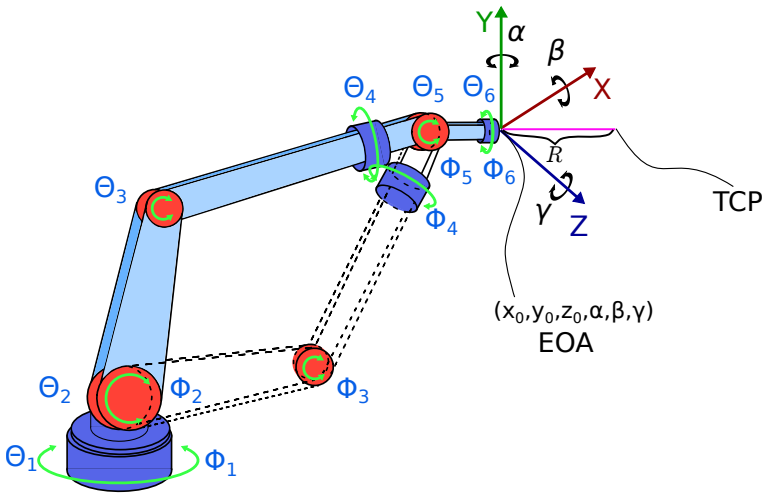

Figure 2.1: The T-space point is $(x_0, y_0, z_0, \alpha, \beta, \gamma)$. It can be reached with two C-space points, i.e., robot configurations: $(\Theta_1, \Theta_2, \Theta_3, \Theta_4, \Theta_5, \Theta_6)$ and $(\Phi_1, \Phi_2, \Phi_3, \Phi_4, \Phi_5, \Phi_6)$. R is the length of tool.

In order to describe a robot pose two spaces are used: T-space for describing the pose of the robot end-effector and C-space for describing the configuration of the robot. Sometimes, C-space is also called joint or axis space.

Definition 1. *T-space—task space $SE(3)$—is used to designate the position $\mathbb{R}^3$ and the orientation of an end-effector $SO(3)$, i.e., $SE(3) = \mathbb{R}^3 \times SO(3)$.*

Normally, a T-space point is specified in homogeneous coordinates [62]:

$$M = \begin{bmatrix} Rot_{3\times3} & Tr_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}.$$

M is the T-space point that define the end-effector pose, where Rot is a 3×3 rotation matrix that stands for the end-effector orientation and Tr is a 3×1 translation vector that stands for the end-effector position.

Since description of the end-effector orientation with a matrix is not intuitive for a human, often a shorter representation with Euler angles is used [28]. Euler angles depict a sequence of rotations about the axes of coordinate system. For example, to describe any rotation with Euler angles in 3D space, three angles are required, i.e., one angle denotes rotation about one axis.

The T-space is used for describing the pose of a robot end-effector. In order to describe the pose of the whole robot, i.e., the value of each angle of robot joints, the C-space is used[1].

Definition 2. *C-space—configuration space $\mathcal{C}$—is a set of possible robot configurations (e.g., joint angles for revolute joints), i.e., $\mathcal{C} = \mathbb{R}^n$, where n is the number of the robot degrees of freedom (DOF).*

The relation between the T-space and the C-space is described with two mappings called Forward and Inverse Kinematics [28].

Definition 3. *Forward Kinematics (FK) takes the robot joint angles and calculates the corresponding end-effector position and orientation, i.e., $FK : \mathcal{C} \to SE(3)$.*

Definition 4. *Inverse Kinematics (IK) is a multivalued function that takes the end-effector position and orientation and calculates the set of possible robot configurations, i.e., $IK : SE(3) \to Y$, where $Y \subset \mathcal{C}$.*

Note that a T-space description is not unique, as often a robot can reach this T-space point with several configurations. Often, up to 8 inverse kinematics solutions exist for a standard 6-DOF PUMA-like industrial robot. An example of reaching one T-space point with two robot configurations is shown in the Figure 2.1.

2.1.2 Robot Trajectory Definition

In order to make a movement, a robot has to follow a certain trajectory. A robot trajectory is specified with a geometrical path and a motion law by which the path must be tracked [18]. A general structure of robot trajectory is presented in Figure 2.2.

A path can be specified either for a whole robot, i.e., the robot joints, or for the robot end-effector. A path for an end-effector is defined as follows:

Definition 5. *An end-effector path $Path_{EE}(k)$ is a function that maps a value $k \in [0, 1]$ to a point in T-space, i.e., $Path_{EE} : [0, 1] \to SE(3)$.*

By an end-effector path, we imply a path either (i) for a tool center point (TCP) for the case when a robot is equipped with a tool or (ii) for an end of arm point (EOA) when no tool is being mounted.

Another way to define a path for a robot to follow is to describe it in the C-space. We call it a robot joint path or simply a robot path. In contrast to the end-effector path, mapping is done to the C-space, i.e., robot joints angles. It is defined as follows:

[1] Standard industrial robots often have 6 degrees of freedom, therefore, $\mathcal{C} = \mathbb{R}^6$

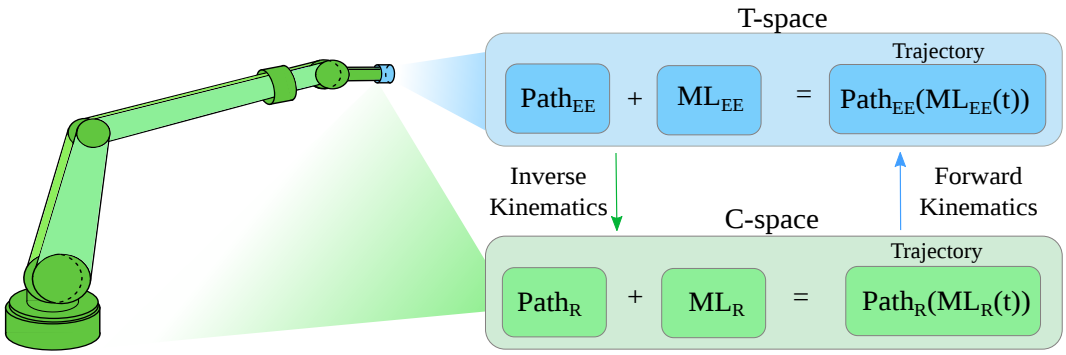

Figure 2.2: Overview of the robot trajectory calculation.

Definition 6. *Robot path $Path_R(k)$ is a function that maps a value $k \in [0,1]$ to a point in C-space, i.e., $Path_R : [0,1] \to \mathcal{C}$.*

Knowing only a path is not enough to describe a motion. One has to specify a motion law that connects time and domain of the path. A motion law is defined as follows:

Definition 7. *A motion law is a function that maps a time value from $[0,T]$ to a value from $[0,1]$, i.e., $ML : [0,T] \to [0,1]$, where T is a desired motion duration.*

A motion law can be applied to an end-effector path or to a robot joint path, then we refer to it as ML_{EE} or ML_R, respectively. As a rule, retrograde motion is not needed, therefore, a motion law is often a strictly monotonically increasing function.

After an end-effector path and a motion law are known, the end-effector trajectory can be obtained as follows:

Definition 8. *An end-effector trajectory is a composition of an end-effector path and a motion law, i.e., $Traj_{EE}(t) = Path_{EE}(ML_{EE}(t))$, where $t \in [0,T]$.*

An output of trajectory planning is a robot trajectory, as it uniquely describes the motion of the robot. A robot trajectory is a tuple of trajectories for every robot joint. It is defined as follows:

Definition 9. *A robot trajectory is a composition of a robot path and a robot motion law, i.e., $Traj_R(t) = Path_R(ML_R(t))$, where $t \in [0,T]$.*

In practice, both end-effector and robot paths are often defined as a set of via-points and then interpolated in the domain $[0,1]$ to reach continuity.

2.1.3 Task Definition

A robot movement should produce some useful outcome by performing certain tasks. An effective task is a task where a robot performs domain-related work, e.g., welding, drilling, cutting or surface painting. Often we need to perform several effective tasks, e.g., welding a set of seams rather than a single long seam. This requires a supporting task, i.e., a movement that a robot has to perform to move from one effective task to another. In the literature, supporting tasks are sometimes also called "intertasks" [57].

Effective and supporting tasks have different restrictions. In order to process a certain effective task, a robot has to follow a predefined end-effector path with its tool. In contrast, during performing a supporting task, a robot makes a point-to-point movement or follows any other collision-free path. On the one hand, industrial processes often have no special requirements for a certain motion law of a path for supporting tasks. On the other hand, some effective tasks require a particular motion law. For example, too fast movement of a tool in a milling task might damage the tool or the object.

The geometry of effective tasks can be simple (e.g., welding a linear seam or drilling a hole) or complex (e.g., painting a surface or milling an object). The geometry of a simple task is already a path for a robot to follow. For complex tasks, e.g., surface painting, an infinite number of end-effector paths exist. Often, computationally expensive techniques are required to obtain a low-cost path for an effective task with complex geometry, e.g., to solve a coverage path planning problem for painting [41].

In this thesis, we assume that an end-effector path for an effective task geometry is already calculated with a domain-dependent algorithm. Thus, from the robot's perspective, an end-effector path defines an effective task, i.e., $Task_{EF} = Path_{EE}$. Similarly, a robot joint path defines a supporting task, i.e., $Task_S = Path_R$.

2.1.4 Effective Task Relaxation

In contrast to supporting tasks, where a selected path is not critical for an application process, a distinctive feature of effective tasks is that a robot has to follow a predefined end-effector path with a predefined motion law. For example, welding a line with constant velocity produces an even influence on a surface and any deviations from the line or fluctuating velocity would violate the production process. The end-effector path and the motion law depend on industrial process requirements. They are typically provided without considering robot kinematics and, hence, often cause high jerks in the robot's joints. However, effective tasks often allow certain spatial freedom of execution that can be used to find an end-effector path such that it leads to smooth C-space trajectory. Typically this optimization criterion is defined as the cost of the corresponding robot trajectory. In this thesis, we introduce the notion of a relaxed effective task that describes such effective tasks where an end-effector path can have a certain deviation.

First, let us define a relaxation function that specifies the neighborhood area around a certain point in T-space:

Definition 10. *A relaxation function Relaxation(P) maps a T-space point P to a T-space subset, i.e., Relaxation : $SE(3) \to Y$, where $Y \subset SE(3)$.*

A relaxation function can map to any $Y \subset SE(3)$, e.g., to a sphere, a cube, or any other neighborhood up to 6D.

After defining a relaxation function, one can apply the relaxation to an effective task. Based on this function, we define a relaxed effective task as follows:

Definition 11. *A relaxed effective task $Task_{EF}^{Rel}(k)$ maps a value $k \in [0,1]$ to a certain neighborhood of the T-space point $Task_{EF}(k)$. It is defined as a function composition $Task_{EF}^{Rel} = Relaxation \circ Task_{EF} : [0,1] \to Y$, where $Y \subset SE(3)$.*

Similar to effective tasks, relaxed effective tasks can be specified with a finite number of via-volumes. These via-volumes are interpolated to reach continuous geometry.

For a relaxed effective task, there are an infinite number of admissible end-effector paths, which are equivalent in performing the task. However, such paths are not equivalent in robot kinematics and lead to different robot trajectories with different costs.

Further, we would need to know whether a point or a path belongs to a relaxed effective task or not. It is derived according to the following two definitions:

Definition 12. *We say that a T-space point P belongs to a $Task_{EF}^{Rel}$, when $\exists k \in [0,1]$, such that $P \in Task_{EF}^{Rel}(k)$.*

Definition 13. *We say that an end-effector path $Path_{EE}$ belongs to a relaxed effective task $Task_{EF}^{Rel}$, when for $\forall k \in [0,1]$ the obtained T-space point $Path_{EE}(k)$ is within the neighborhood $Task_{EF}^{Rel}(k)$, i.e., $Path_{EE}(k) \in Task_{EF}^{Rel}(k)$.*

We assume that a path that belongs to a relaxed effective task is valid. It means that the path can handle the relaxed effective task and does not harm the production process.

There are numerous ways of relaxing an effective task. One possible example is depicted in Figure 2.3. In the observed case, the neighborhood is defined as a circle for task points position and a box for an orientation, i.e., $([a_l, a_u], [b_l, b_u], [c_l, c_u])$ in the coordinate system X_w, Y_w, Z_w. Figure 2.3 depicts the orientation of a point x_{np}, y_{np}, z_{np}. The dotted blue end-effector path $Path_{EE}$ is valid, as for all $k \in [0,1]$ a T-space point $Path_{EE}(k)$ of this path is within a neighborhood $Task_{EF}^{Rel}(k)$.

The approaches proposed in this thesis do not depend on a way of relaxing effective tasks. However, for simplicity of explanation we specify the orientation neighborhood as a box $([a_l, a_u], [b_l, b_u], [c_l, c_u])$, where:

- $[a_l, a_u]$ represents the lower and upper bounds for the polar angle in spherical coordinates; therefore, the maximum limit is $[0°, 180°]$.

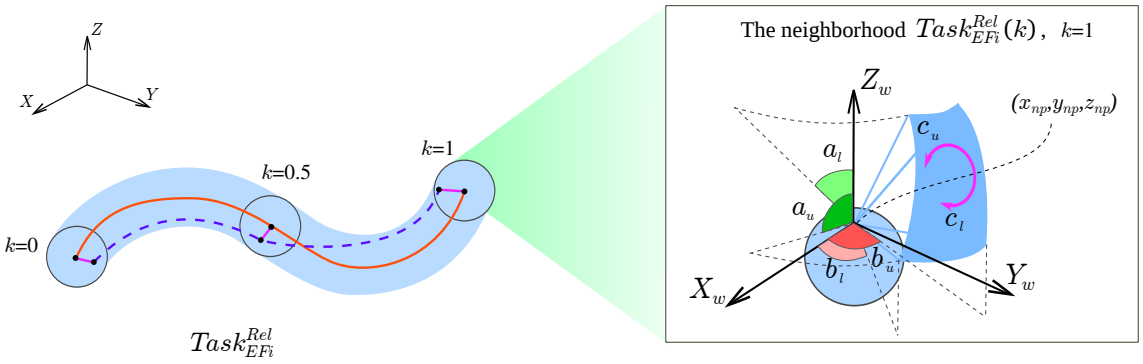

Figure 2.3: Effective task $Task_{EF}$ is depicted in red. Relaxed effective task $Task^{Rel}_{EF}$ defines the neighborhood depicted in blue. $Path_{EE}$ is one of many possible end-effector paths; it is depicted with a dotted blue curve. For simplicity, neighborhoods are depicted only for k=0, 0.5, 1 and a possible relaxation of orientation is depicted for k=1.

- $[b_l, b_u]$ represents the lower and upper bounds for the azimuthal angles in spherical coordinates; therefore, the maximum limit is $[0°, 360°]$.
- $[c_l, c_u]$ represents the lower and upper bound for a desired orientation along the tool axis; therefore, the maximum limits is $[0°, 360°]$.

Note that orientation can be expressed not only in the world coordinates X, Y, Z but also in the coordinates of a task. For this, a rotation matrix M_w should be specified:

$$M_w = \begin{bmatrix} X_{wx} & Y_{wx} & Z_{wx} \\ X_{wy} & Y_{wy} & Z_{wy} \\ X_{wz} & Y_{wz} & Z_{wz} \end{bmatrix}.$$

M_w is a rotation matrix that corresponds to orientation X_w, Y_w, Z_w of a neighborhood in relation to the world coordinate system X, Y, Z.

2.2 Objectives

A scenario from plastic manufacturing is shown in Figure 2.4. This case study is inspired by an industrial scenario based on a commercially available product[2]. It is easier for the manufacturing process to produce a plastic detail as one piece using a molding press machine, and then to cut out the holes, instead of producing a complex piece at once. The job for a robot is to cut a number of holes out of a big plastic board and deburr the outer border of the detail, i.e., delete the flashes. To do this, the robot is equipped with a cutting knife.

The left part of Figure 2.4 shows an initial plastic detail. No parts are cut out and the frame has a rugged shape. Such molding flashes appear due to the manufacturing

[2]KUKA Roboter application scenario: http://www.kuka-robotics.com/en/solutions/solutions_search/L_R148_Deburring_of_plastic_engine_covers.htm

process. When melted plastic is pressed into a form, it often leaves some pressing brackets. This results in uneven, rough external contours. The right part of Figure 2.4 shows the workpiece after processing where all cuts have been made.

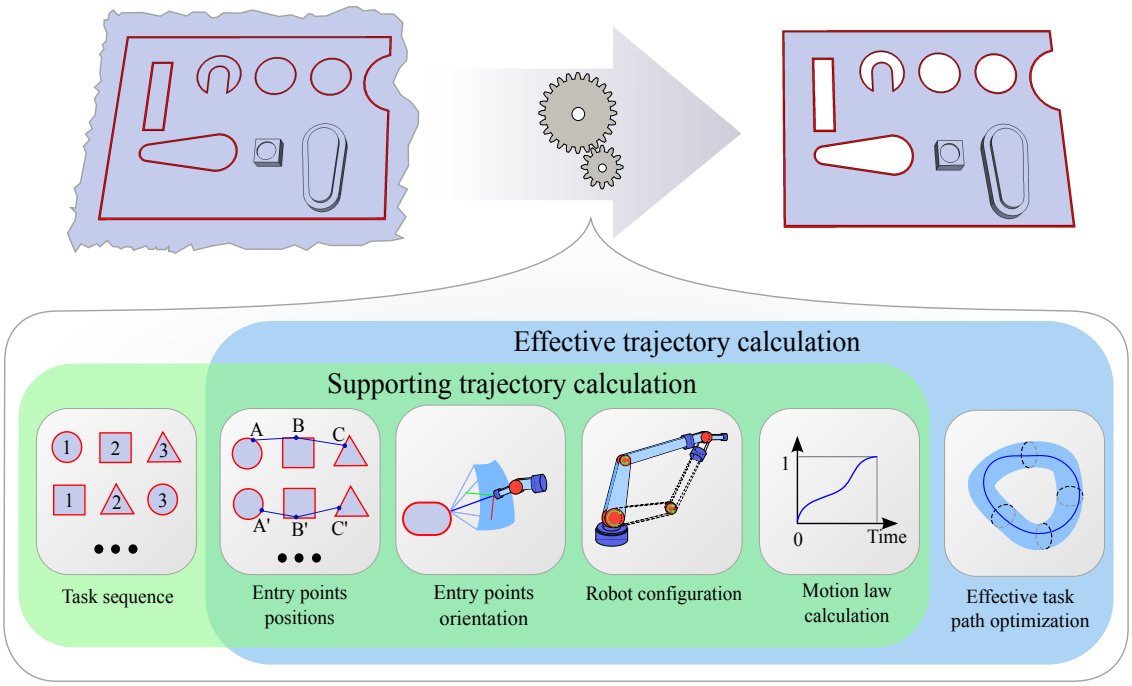

Figure 2.4: Problem overview

The problem is to find trajectories for effective tasks to perform the cuttings and for supporting tasks to move the robot from one effective task to another. Even this simple scenario brings multiple sub-problems that have to be solved:

Task sequence: Many industrial tasks do not impose constraints on sequence of execution, as tasks are independent from each other. For example, in our case study, there are no requirements on performing cuttings in a certain order; therefore, a sequence of effective tasks could be optimized.

Entry point position: Often an effective task itself provides some freedom. For example, a line segment provides two variants of execution: from one end to another and the other way around. In our scenario, effective tasks provide an infinite number of possible entry points, as closed-contours can be started from any point. For other types of tasks, an entry point can be within a 3D area, e.g., for a scenario where a robot has to take a picture with a camera mounted on its end-effector.

Entry point orientation: The right orientation of a tool is critical for many effective tasks. In our scenario, an incautiously chosen orientation can lead to damage of the knife and the plastic detail, e.g., when more pressure would come to the knife's edge rather than its blade. In our case, an admissible orientation of effective tasks is

specified with neighborhoods of a box-like shape. This is an important degree of freedom in robotics tasks as it allows a certain deviation, for example, in a plane perpendicular or parallel to the direction of cutting.

Robot configuration: In most situations, there is no requirement to use certain solutions of inverse kinematics to perform cutting/drilling of contours/holes. In general, a standard industrial robot can reach every 6D position with eight different configurations (i.e., "elbow-up vs. elbow-down", "in-front vs. overhead" and "forehand vs. backhand"). However, a choice of configurations for each effective task significantly affects the cost of supporting movements.

Motion law calculation: Motion law is applied for calculating trajectories of both supporting and effective tasks. These task types require different approaches for calculating a motion law. For supporting tasks, a motion law shape is domain-independent and a motion law that leads to a synchronous trapezoid velocity profile can be used. Effective tasks are sensitive to the end-effector path; therefore, a motion law of an end-effector path should be optimized with respect to the end-effector path geometry.

Effective task path optimization: Some of effective tasks require a strict definition of motion law. These are mainly the tasks that interact with a working object, e.g., welding or deburring, as a movement with a high velocity might not provide a desired effect on the object or damage the tool or the object. Such strict motion law dictates the performing time for an effective task. Nevertheless, by making use of the effective task path freedom, it is possible to reduce other costs, for example trajectory jerk, and make the movement smoother.

2.3 Thesis Problem

Summarizing the objectives stated above, the robot trajectory optimization problem considered in this thesis is formulated as follows:

> Find minimal-time supporting C-space trajectories and minimal-jerk effective C-space trajectories by considering the spatial relaxation of effective tasks and multiple figures of robot kinematics.

A trajectory for supporting movements can be calculated in two steps: (1) computing a path by optimizing the task sequence, position and orientation of the entry points, and the corresponding robot kinematics solution, and (2) applying a motion law to the obtained path. A trajectory of an effective task can also be constructed in two steps: (1) find an end-effector path that belongs to the task and starts from the given position and orientation of the entry point as well as the predefined robot kinematics solution, (2) apply a motion law to the obtained path.

2.4 Solution Concept

Every sub-problem stated in Section 2.2, is computationally expensive to solve, even apart from other sub-problems. The overall problem, i.e., robot trajectory optimization problem, is even more complicated, as the sub-problems have to be considered in synergy because they influence each other. For example, choice of a task sequence depends on chosen entry points and vice-versa. Choosing an inverse kinematics solution for a certain entry point position and orientation depends on the kinematics solutions chosen for other tasks. A sequence of entry points influences a trajectory for the effective tasks. All of this adds up to a large search space that is computationally expensive to explore. We suggest decomposing the problem into simpler components.

Decomposition

We propose a solution concept that decomposes the problem into two stages: first, optimize supporting trajectories and then effective trajectories, see Figure 2.5.

The supporting task trajectory optimization consists of five sub-problems that have to be resolved: find a task sequence, a position of each entry point, an orientation of each entry point, robot configurations and a motion law. In this thesis, we do not explicitly aim at optimizing a motion law, but rather apply a law that leads to a trapezoidal velocity profile for the supporting trajectory and an arbitrarily shaped law for effective tasks. The list of sub-problems is not complete, as there are other problems, e.g., collision-free path calculation, path-smoothing. In this thesis, we concentrate only on the described sub-problems.

We suggest decomposing this problem into two separate components. Component1 searches for a sequence and Cartesian entry points. It uses Euclidean distance as a metric. Component2 takes the output from the Component1, i.e., a sequence of relaxed tasks with entry points, and minimizes a robot trajectory cost metric by optimizing a position, an orientation and a robot configuration for each entry point. A robot trajectory cost metric can be time or distance traveled in C-space. A given motion law is then applied to the calculated path in order to obtain a C-space trajectory between the relaxed tasks. The output of Component2 is a C-space robot trajectory. This decomposition of the supporting task trajectory optimization into Component1 and Component2 allows us to compute a C-space trajectory more efficiently by sacrificing some search space.

The effective task optimization is realized in Component3. Component3 aims at optimizing a C-space trajectory cost for a given relaxed effective task with a C-space entry point. At this stage, motion law can be given or optimized. In this thesis, we are not focusing on optimizing of motion law, but rather assume it to be constant during optimization. An optimization goal can be trajectory jerk or time. An output is a C-space robot trajectory.

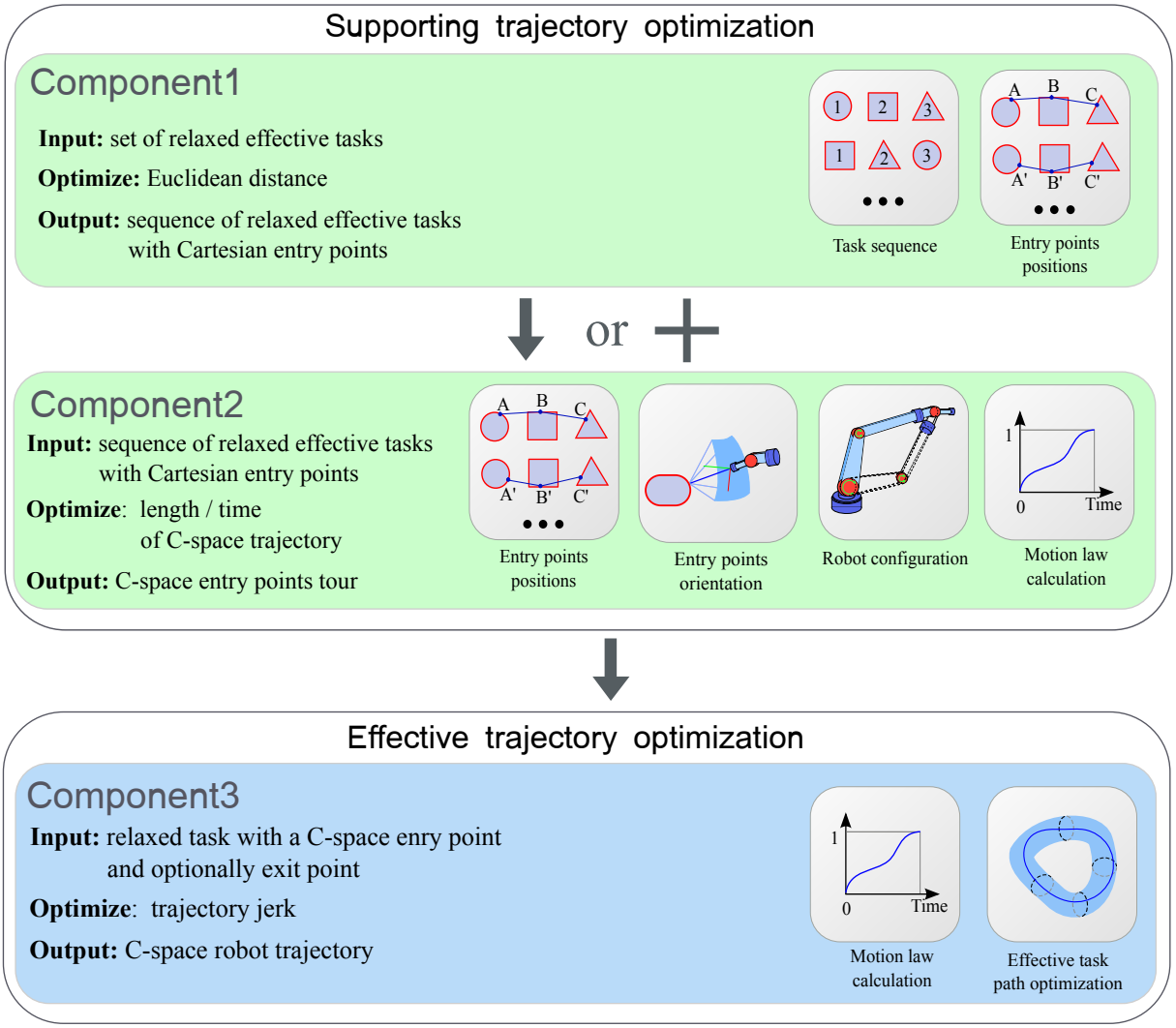

Figure 2.5: Solution concept overview

Possible Solution Strategies

We envision several strategies that can be used to obtain a solution for the overall problem by combining solutions for the sub-problem components. In this thesis, we evaluate the following two strategies:

- Component1 → Component2 → Component3: all three stages are applied sequentially, one after another. At first, a sequence of effective tasks is found, then a supporting trajectory is calculated and finally, effective tasks are optimized.
- (Component1 + Component2) → Component3: a sequence search algorithm in Component1 calls an optimization algorithm at Component2 as a cost function instead of Euclidean distance. Component2 returns a value of a robot-oriented metric: a time or a C-space distance. After a supporting trajectory is found, effective tasks are optimized.

Note that other variants of combining the components are also possible. For example, one can first perform the effective trajectory optimization (Component3) and then the supporting trajectory planning (Component1 and Component2). However, since the result of Component3 optimization is a C-space robot trajectory, which uniquely describes an effective task, applying Component3 first provides almost no freedom for optimization of supporting tasks — only a sequence of effective tasks can be optimized. In that case, a sequence can be found with any known Traveling Salesman Problem approach. In this thesis, we are focusing on utilizing the effective task freedom and, therefore, we do not consider a strategy where effective tasks are optimized first.

The mentioned components can also be used separately or in combination with other approaches. For example, it is possible to apply another task sequencing algorithm from [34] [85] [100], and then adapt a solution to a robot using Component2. It is possible to optimize every effective task trajectory with Component3 and then find a sequence with methods proposed in [102][16].

Below, we provide a mathematical definition of the problems solved in the aforementioned components.

2.4.1 Component1: Relaxed Effective Task Sequencing

In this component, we optimize a sequence of relaxed effective tasks and entry points positions for each task. Optimizing only a sequence will ignore the shape of the tasks in the sequence. Use of a robot space metric is computationally expensive due to multiple inverse kinematics calls. It would also lead to a bigger search space. Therefore, in this package, information about a robot is omitted and Euclidean distance is used as a cost metric. The problem is formulated as follows:

> Given a set of n relaxed effective tasks $Job = \{Task^{Rel}_{EF_1}, \ldots, Task^{Rel}_{EFn}\}$, find a minimal-cost cyclic tour $T = (p_1, \ldots, p_{n+1})$ such that it visits $Task^{Rel}_{EFi}$ in point p_i, i.e., $\exists k \in [0, 1]$ so that $p_i \in Task^{Rel}_{EFi}(k)$.

2.4.2 Component2: Supporting Trajectory Optimization for a Relaxed Effective Task Sequence

We assume that a given task sequence is already optimized and, therefore, the order of the tasks in the sequence is not changed. The goal of Component2 is to find a tuple of robot configurations such that they belong to the entry areas of the relaxed effective tasks and have minimal-cost supporting trajectories between them. The cost is based on C-space, e.g., distance traveled or time and, thus, this component is robot-dependent. The problem is formulated as follows:

> Given a sequence of n relaxed effective tasks $Job = (Task^{Rel}_{EF1}, \ldots, Task^{Rel}_{EFn})$, find a minimal-cost cyclic C-space tour $CT = (p_1, \ldots, p_{n+1})$ such that it visits $Task^{Rel}_{EFi}$ in a point $FK(p_i)$, i.e., $\exists k \in [0, 1]$ so that $FK(p_i) \in Task^{Rel}_{EFi}(k)$.

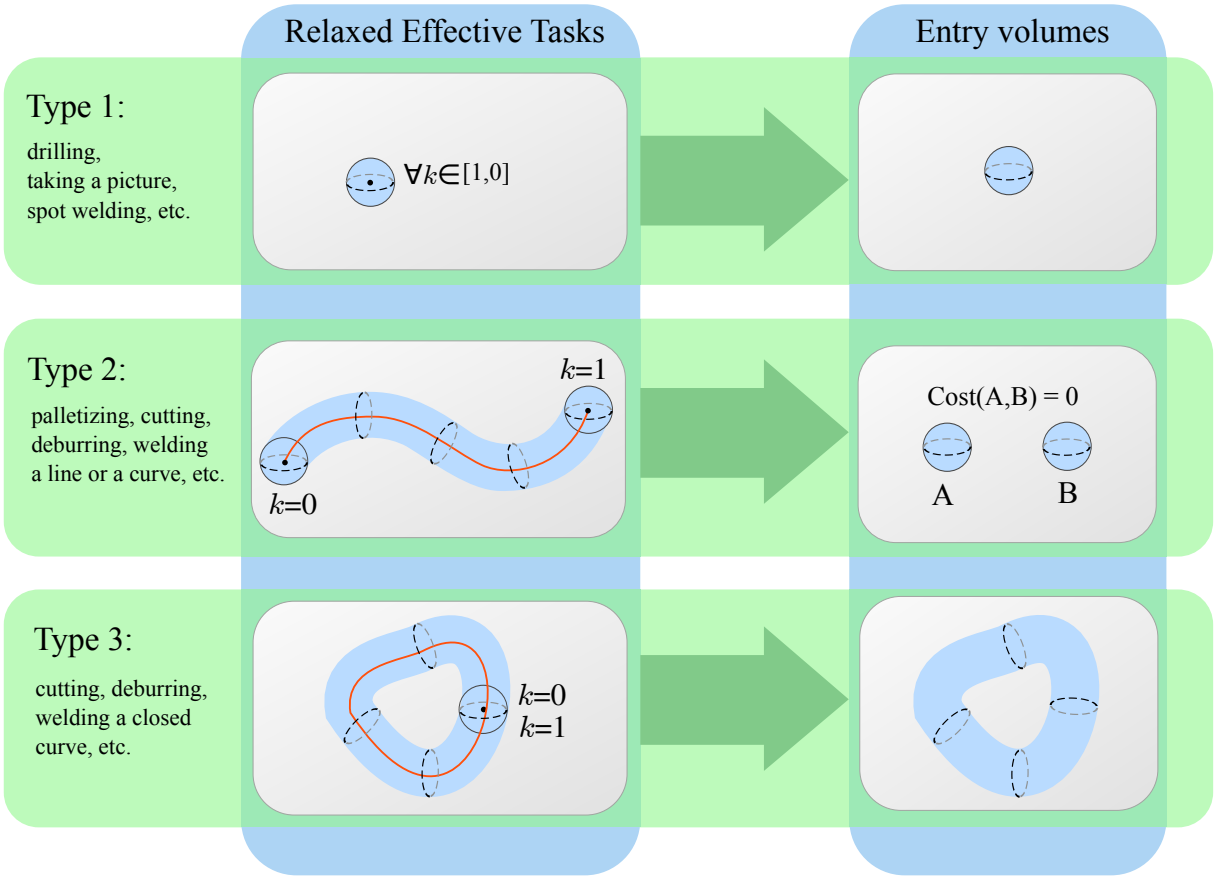

Figure 2.6: Task types and entrance volumes.

2.4.3 Component3: Robot Trajectory Optimization for a Relaxed Effective Task

After supporting movements are found, the trajectory of every effective task is optimized. This step is robot-dependent, as it involves a cost from a robot C-space. An optimization cost can be: energy, jerk or any domain-specific parameter. The problem is formulated as follows:

> Given a relaxed effective task $Task_{EF}^{Rel}$, an end-effector motion law ML_{EE} and a trajectory duration T, find such a path $Path_{EE}$ belonging to the $Task_{EF}^{Rel}$ that it leads to a minimal-cost robot C-space trajectory $Traj_R(t)$, where trajectory $IK(Path_{EE}(ML_{EE}(t))) \rightarrow Traj_R(t)$, with $t \in [0, T]$.

During optimization, it is important to verify that a C-space trajectory is feasible, i.e., maximal joint velocities, acceleration bounds and angular joint limits are not violated.

2.5 Assumptions

The proposed solution conception works under the assumption that the freedom of task execution is defined depending on industrial process limitations and, therefore, does not harm the quality of the performed work.

In the following, we assume that an effective task has an entry volume that is the same as an exit volume. It is valid, for example, for closed-contour welding or camera taking pictures scenarios (see Type 1 and 3 in Figure 2.6). However, in some domains or applications, an entry volume can differ from an exit volume. For example, in a palletizing scenario, an entry volume is a volume where a robot grasps an object, whereas an exit volume is a goal location where it releases the object. Another example is a scenario of welding an open-end curve which has two ways of execution: starting from one end or another. Although it might seem that entry volumes differ from exit volumes, a task from such a scenario can be decomposed into two separate tasks with a cost for traveling between each other equal to zero. That would reduce the task to a case with one entry-exit volume for each end. The search for entry points is then done inside these entry volumes. It should be mentioned, that Component3 makes sense to perform only for Type 2 and Type 3 as there is no effective movement in Type 1.

We assume that the working environment is not cluttered. Thus, explicit collision-free planning can be omitted during optimization and applied afterwards.

In this thesis, the motion law and trajectory duration time for Component3 is given and only the via-points are optimized. However, moving via-points leads to shorter or longer final path, therefore, with a constant motion law the final velocity would be lower or higher, respectively. We assume that this change of velocity is insignificant for production process.

Note that we do not intend to solve tasks like painting, as this requires complex path planning and optimization to meet an industrial process. We already assume that a certain curve for performing a task is calculated with known approaches, and we provide a certain relaxation to the task and use it to decrease the cost.

3. Related Work

> Always listen to experts. They'll tell you what can't be done and why. Then do it.
>
> Robert A. Heinlein

The problem solved in this thesis is closely related to (i) industrial robotics with its degrees of freedom and limitations, (ii) tour-searching combinatorial problems and (iii) trajectory optimization approaches. In this chapter, work related to this thesis is presented.

This chapter is organized as follows. State-of-the-art approaches for industrial robot programming and task sequencing features are presented in Section 3.1. In Section 3.2, the difference between this thesis problem and related planning problems in robotics are discussed. Task sequencing is usually modeled with tour-searching combinatorial problems that are presented in Section 3.3. Simple application of existing solvers for tour-searching problems is not enough, as robot features are ignored. The research approaches which adapt these solvers to robotics are discussed in Section 3.4. Robot trajectory optimization is presented in Section 3.5. We conclude and summarize the differences of this thesis to the discussed existing problems in Section 3.6.

3.1 Industrial Robot Programming

A common workflow of industrial robots is to perform a sequence of tasks. Performing a task sequence consists of two alternating stages: effective and supporting movements/tasks. An effective movement is a stage where a task is performed, e.g., moving along a welding seam with a switched-on welding torch. A way how an effective movement is executed highly depends on production process requirements. State-of-the-art methods of how to program movements of a robot can be split into offline and online approaches [19].

In online programming, a robot is directly taught a movement, which it has to replicate later in production mode, e.g., with the lead-through method [89]. Accordingly, a complete movement must be taught, e.g., movements along welding seams as well as movements between seams. This is a time-consuming process, which requires a lot of intuition and experience, if high-quality movements are desired. The quality of a path depends only on the experience of a programmer and almost no optimization approaches are involved [19, 79].

An offline approach is typically based on a simulation. A trajectory that has to be executed by a real robot later in production is calculated or taught in a simulated environment. With modern offline programming approaches, nowadays trajectories are automatically generated from CAD data. Supporting movements often do not have constraints on the execution except obstacle avoidance and are programmed manually or calculated with collision-free path planners [60]. These approaches has some major benefits, e.g., a possibility of testing a calculated path for potential collisions before real execution – and, therefore, preventing an expensive hardware from being damaged.

Regardless of what kind of approach is used—offline or online—programming itself is typically done in proprietary programming languages like Epson Robotics' SPEL+, ABB's Rapid, Stäubli's VAL3 or KUKA's KRL [89]. Despite the fact that each of the mentioned languages enables us to describe a very broad set of robot activities, they have obvious disadvantages like a time consuming programming in the imperative style. A movement is described with commands like: "move to point A linearly" or "make a point-to-point movement to point B". A complex task can include a large number of via-points and, thus, can be challenging to describe. Commercial systems try to ease the programming process by creating interfaces which eliminate the need of programmers to write the code for every point directly. They provide an opportunity to choose a needed action from a list and generate a required code. However, these systems are application-specific [19]. The quality and optimality of the obtained movements highly depends on the programmer's skills.

Usually, industrial robots have to repeat the same task sequence multiple times to produce a large batch of similar products. Therefore, it is important to minimize an execution cost, as it greatly influences the production efficiency. A task sequence optimization is either fully omitted in offline simulation environments (e.g., RobotWorks[1])

[1] Compucraft Ltd, RobotWorks www.compucraftltd.com

or presented with limited functionality (e.g., customization for DELMIA[2] adapted for drilling applications[3]). The customization for DELMIA allows automatic sequencing of drilling tasks but applies only a simple greedy algorithm, i.e., the next drilling task to be added to a tour is the nearest task. This trivial algorithm does not provide good results, as it often converges to a local minimum. In addition, tasks are specified with one entry robot configuration.

3.2 Related Planning Problems in Robotics

In this section, we briefly outline related planning problems in the field of robotics. We highlight the differences between them and the problem considered in the thesis. The work presented in this thesis is not competing with related problems but it rather can be easily integrated into existing architectures by extending the planning with an additional degree of freedom.

Production Scheduling

The problem discussed in this thesis differs from the scheduling cell optimization problem that is often mentioned in the economics domain, as they have different goals. The scheduling cell optimization problem addresses optimization of a production facility, i.e., how many robots should be used, how many parts comprise a detail to produce and how to spread work evenly between robots in order to increase the overall cell throughput. Due to the high computational complexity of the problem, collision factors and kinematics are ignored. As a consequence, the goal is to compute such a solution that minimizes the total Cartesian distance instead of a cost based on robot's joints. This kind of research is mostly concentrated on how to organize production processes, instead of focusing on how to construct optimal movements for a single machine or a robot [30]. This thesis concentrates on the motion of a robot, without involving the information about a production process.

Multi-Robot Task Planning

In scenarios with multiple robots, tasks have to be distributed among the robots. Most common methods are to divide tasks into clusters, one for each robot, and solve a single-robot problem for each cluster, e.g., by a graph-based approach [68] or by using Stochastic Clustering Auction technique [104]. More details on multi-robot planning can be found in a survey by Portugal and Rocha [82]. In the remainder of this thesis, we assume that clustering of tasks for multiple robots has been already performed. We present an optimization approach for a single robot with a given set of tasks.

[2]Dassault Systemes http://www.3ds.com

[3]DELMIA V5 Robotic Drilling Application http://www.delfoi.com/web/products/delfoi_products/en_GB/drilling-app/

Task-Level Planning

In task-level planning, robot actions are specified by interactions between the robot and objects. The goal of the task-level planning is to find a sequence of actions which a robot has to perform in order to modify an environment from an initial state to a goal state [68]. For example, a goal can be to construct a sequence of actions in order to put an object from a box on a table. A solution for this example will consist of the following actions: 1) "open the box", 2) "grasp the object", 3) "move the object onto the table", 4) "release the object". Every single action is then planned with domain-dependent planner. The data representation is often defined as an object-oriented model [25][26]. This model includes relations between objects, categories and physical laws. In this thesis, tasks have no logic or physic-based relations. Therefore, there are no constraints on the order of tasks.

Online Control-Based Planning

A environment where a task sequence is executed can dynamically change. A sensor will detect these changes and a control-based approach should quickly replan execution of the tasks. One way to implement it is to organize a task sequence as a stack and try to maintain its consistency online [69] For example, due to environment or robot restrictions, a controller removes or adds a new task to a stack. In this thesis, offline approaches for the known environment are presented. However, during trajectory execution, online planners can be used for a quick replanning.

Manipulation Planning

Another well-known related planning problem in robotics is a manipulation planning problem. It occurs when a robot has to move an object to a specific location in an environment. The goal of the manipulation planning is to find a sequence of actions the robot has to perform in order to accomplish this task [62]. Further details on the manipulation planning can be found in the following surveys [53] and [91]. In this thesis, we assume that the manipulation planning problem has already been solved and its solution is a description of an effective task. As soon as a path for an effective task is constructed, it can be optimized with the approach proposed in Chapter 6.

Collision-Free Path Planning

A significant amount of research has been focused on collision-free planning. The goal of the collision-free planning is to calculate an optimal path between two robot configurations. Optimality is expressed with multiple criteria that has to be minimized: distance traveled, time and energy. Normally, the planning is done in C-space and a path is obtained directly for the whole robot. For this problem, there are many heuristics, graph-based algorithms and, in particular, tree-based approaches: Probabilistic Roadmap (PRM) [55], Rapidly-growing Random Trees (RRT) [60] or Bidirectional RRT (BiRRT), Artificial potential fields [70] or fields with application of annealing [32].

Sample-based techniques, e.g., RRT/BiRRT or PRM, work well with high-dimensional spaces. Artificial potential fields are more popular for mobile applications. After a collision-free path is obtained, it is smoothed and a predefined velocity profile is applied in order to obtain a robot trajectory [62]. For more details and a comparison of collision-free planners see [39]. The approaches stated above allow us to calculate collision-free paths between two robot configurations automatically. However, they do not consider the optimization of a task sequence. Collision-free path planning is used for calculation of supporting task movements. In this thesis, we consider that an environment is not cluttered and, therefore, the collision-free planning can be applied afterwards.

3.3 Problems to Model Task Sequencing

Tour-searching combinatorial problems build a foundation for task sequencing in robotics. In this section, related combinatorial problems are described. The main parameters of the problems are illustrated in Figure 3.1.

Tasks can be represented as points or finite sets of points. We refer to this type of tasks as primitive tasks. Tasks can also be modeled as areas, in that case we refer to them as complex tasks. For primitive tasks, an output can be a sequence of the tasks or a path going through the tasks. Among other information, a path includes information about the order of the tasks. For complex tasks, knowledge about a sequence is not enough, as it is also required to know from which entry point each task has to be started. Therefore, a solution for the task sequencing problem with complex tasks is either a sequence of entry points or a path which includes information about the entry points.

3.3.1 Sequencing Primitive Tasks

The most well known problem of searching for an optimal sequence is the Traveling Salesman Problem (TSP) [10]. The goal of the TSP is to find a minimal-cost cyclic tour through a set of points such that every point is visited once. It is an NP-hard combinatorial optimization problem and has been well-known in industrial robotics for many years. The cost between two points in TSP is the same for both directions. If the distance between two points depends on the direction of traveling, then such problem is called Asymmetric TSP (ATSP) [45]. An important extension of the ATSP is inclusion of constraints on order, when a point “A” has to be visited before “B” in a tour. This problem is referred to as Sequential Ordering Problem (SOP) [75]. A Shortest Sequence Problem (SSP) is similar to the TSP, but with a difference that a salesman does not have to return to a starting point [98]. If points from the SSP are substituted with sets of points, then this problem is called SSP++. The goal of the SSP++ is to find a minimal-cost tour that contains a point from every set of points with no need to return to the starting point. A generalization similar to the SSP++ exist for the TSP. This problem is called Generalized TSP (GTSP) [94]. The goal of the GTSP is to find a minimal-cost cyclic tour that contains a point from every set. The GTSP is also known as TSP++, set TSP or One-of-a-Set TSP. A Multi-Goal Path

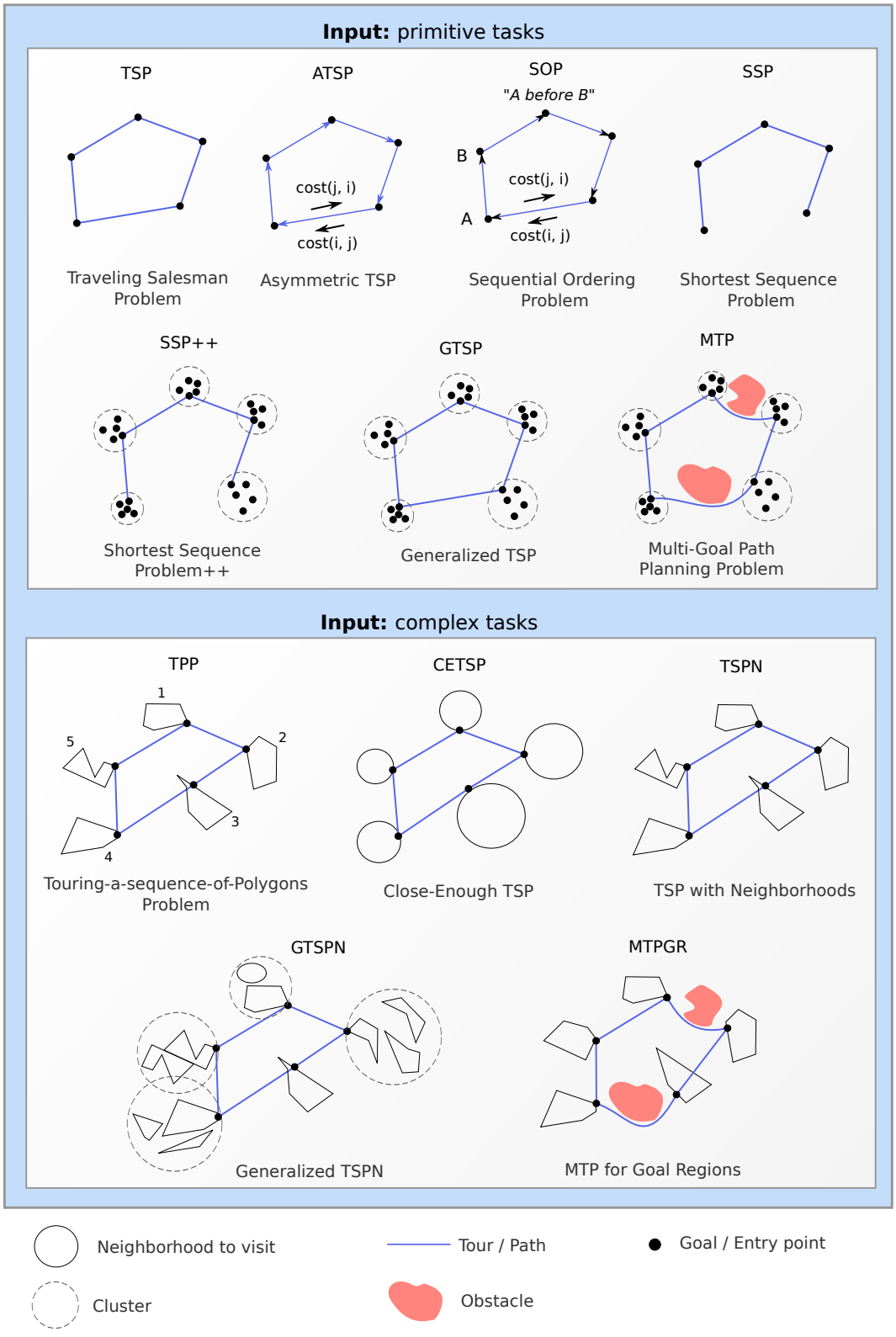

Figure 3.1: Problems to model task sequencing.

Planning Problem (MTP), a variant of the GTSP, was introduced by Wurll et al. [98] especially for robotics. Every set here is a set of the inverse kinematics solutions for a T-space goal point. The objective of the MTP is to calculate a minimal-cost cyclic collision-free path that visits one point from each set.

3.3.2 Sequencing Complex Tasks

In case if tasks are not points but polygons and their sequence is given, then this problem is known as a Touring-a-sequence-of-Polygons Problem (TPP) [33]. The goal of the TPP is to find a minimal-cost cyclic tour that visits a predefined sequence of regions. If points should be visited within a certain radius, then this TSP extension is known as Close-Enough TSP (CETSP) [71]. In this problem, every point in 2D space is represented as a disk. A more general problem is the TSP with Neighborhoods (TSPN) [11]. The goal of the TSPN is to find a minimal-cost cyclic tour through a set of regions (neighborhoods) such that every region is visited once. If neighborhoods are clustered, then this problem is called Generalized Traveling Salesman Problem with Neighborhoods (GTSPN) [96]. The goal of the GTSPN is to visit at least one neighborhood from each cluster. In case if obstacles are integrated into the TSPN and areas are convex polygons, then this problem is referred to as MTP for Goal Regions (MTPGR) [38]. This problem is the most general and computationally difficult to be solved among the aforementioned problems.

3.4 Robotic Task Sequencing Approaches

In this section, we discuss state-of-the-art approaches for sequencing of both primitive and complex tasks in robotics. Researchers referred to this problem differently. In general, there are two common names: task sequencing/scheduling [7, 57, 59] and multi-goal path planning [38, 97]. Multi-goal planning considers obstacles in the environment, therefore, a collision-free path is the output. Task sequencing/scheduling may consider obstacles [102] or may not [59, 103], therefore, the output could be either a sequence or a path. In the following, we refer to the issue of optimizing robot site in presence of multiple goals/tasks as the task sequencing problem. This problem is relevant to both service and industrial robotics. However, it has greater impact for industrial robotics, as in production environments the same tasks have to be repeated multiple time. The summary of approaches for task sequencing is presented in Table 3.1. The overall optimization process can involve large number of factors that influence the sequence. Further, these factors are described in detail.

Multiple Inverse Kinematics Solutions

Often the articulated robot could reach a T-space point (e.g., drilling point) with several C-space points (i.e., several robot configurations). Although this greatly increases the search space and makes the sequencing problem more difficult to solve, it brings a large potential for optimization. The freedom of choosing a robot configuration is used to avoid collisions and reduce the duration of the movement. The column labeled “Multiple IK” in Table 3.1 depicts the presence of this feature in the considered approaches.

Task Specification and Entry Points

Robot tasks vary by their complexity. Often they are modeled in a simple way as T-space or C-space points. At a first glance, a simple task such as a hole drilling could be represented with the appropriate robot configuration, i.e., C-space point. However, this definition is too explicit, as the task could be reached with multiple configurations. In that case a T-space point might be a sufficient representation. Nevertheless, the rotation along the drilling axis is not important for the manufacturing process, thus it could be also used for optimization, as it greatly influences the robot motion behavior. Therefore, even simple robotics tasks usually bring multiple possibilities for optimization. Thus, there is also a group of approaches that consider the task geometry and represent them as 2D-3D areas, closed-contours, etc. In case if a task is complex, it is then required to find the entry point. The columns labeled "Entry Point Optimization" and "Task Specification" in Table 3.1 depicts what type of specification is used and either entry points is required to be found.

Objective

The optimization objective in robotic task sequencing can be minimization of the time, length of the Cartesian path of the end-effector or the C-space path as well as specific industrial objectives. The motion time in the PTP movement is dictated by the slowest joint velocity and acceleration. In the approaches, where the kinematics is not considered, often the Cartesian path length has to be minimized. The objective could be also dictated by the specificity of the manufacturing process, e.g., minimization of the material distortion caused by welding [100] or minimization of the laser path length [59].

Problem Models

When taking several features into account the researchers might obtain a too complicated problem that is difficult or impossible to solve within a feasible amount of time even using the heuristic approaches. Therefore, often complicated problems are reduced to simpler problem models. That brings errors but makes the optimization feasible. For example, when solving MTPGR, first TSPN can be solved and then collision-free paths can be calculated [59]. When solving TSPN, at first the sequence could be found (i.e., TSP) and after that the position of the points (i.e., TPP) [71]. Therefore, the actual problem obtained by combining several features is often different to the sequencing problem that is solved at the end. See columns labeled "Actual Problem" and "Solved as" in Table 3.1 respectively.

3.4.1 Sequencing Primitive Robotic Tasks

One of the first research works that discovered an application of TSP-like problems in industrial robotics for point-to-point movements was done by Dubowsky et al. [34]. They modeled the problem of point-to-point movement as an ATSP instead of TSP,

Table 3.1: Overview of related state-of-the-art approaches. The "T-point" and "C-point" stand for points in T-space and C-space, respectively. "Cart. path" and "C-path" is the total distance of a path in Cartesian space and C-space, respectively.

Input	Approach	Entry Point Optimization	Multiple IK	Task specification	Objective	Actual problem	Solved as
Simple tasks	[34]	−	−	T-point	cycle time	TSP, SOP	ATSP, SOP
	[1]	−	+[a]	T-point	cycle time	GTSP	TSP
	[2]	−	+[a]	T-point	cycle time	GTSP	TSP
	[35]	−	+	T-point	cycle time	TSP	TSP
	[81]	−	+	T-point	cycle time	MTP	TSP
	[98]	−	+	T-point	C-path	MTP	GTSP, SSP++
	[93]	−	−	C-point	C-path	MTP	TSP
	[97]	−	+	T-point	C-path	MTP	GTSP, SSP++
	[86]	−	−	C-point	cycle time	MTP	TSP
	[103]	−	+	T-point	cycle time	GTSP	TSP
	[87]	−	+	T-point	cycle time	MTP	GTSP
	[85]	−	−	T-point	Cart. path	TSP	TSP
	[66]	−	+	T-point	C-path	GTSP	TSP
	[46]	−	+	T-point	cycle time	MTP	TSP
	[100]	−	−	T-point	complex[b]	TSP	TSP
	[22]	−	+	T-point	cycle time	MTP	several[c]
	[16]	−	+	T-point	cycle time	GTSP	TSP
	[99]	−	+	T-point	cycle time	MTP	TSP
	[102]	−	+	T-point	cycle time	MTP	TSP
	[52]	−	+	T-point	complex[d]	MTP	TSP
	[61]	−	+	T-point	cycle time	MTP	TSP
	[57]	−	+	T-point	complex[e]	GTSP	GTSP
Complex tasks	[43]	+	−	2D/3D areas	Cart. path	TSPN	TSPN
	[38]	+	−	Polygons	Cart. path	MTPGR	MTPGR
	[47]	+	+	T-point+2D[f]	cycle time	MTPGR	TSP
	[44]	+	−	7D curves	cycle time	MTPGR	TSPN
	[37]	+	−[g]	3D volumes	complex[h]	MTPGR	TSP
	[59]	+	−[g]	3D volumes	complex[h]	TSPN	TSP
	[96]	+	+	$\mathbb{R}^n$ areas	Cart. path	GTSPN	GTSPN
	THESIS	+	+	$\mathbb{R}^n$ areas	cycle time	TSPN	TSP + TPP

[a] two IK solutions are considered
[b] cycle time and welding distortion
[c] a fixed, an unfixed(i.e., TSP) or a mixed sequence (i.e., SOP)
[d] cycle time and cost of a candidate manipulator
[e] cycle time and painting quality metric
[f] certain freedom along Z axis and rotation around Z axis
[g] applied afterwards
[h] Lexicographical order of three criteria: minimal cycle time / minimal path length (scanner head) / minimal path length (laser end point)

as in robotics, a cost between two points depends on the direction of a movement due to gravity and kinematics. The general idea of the proposed method is to manipulate weights in a cost matrix in order to satisfy desired precedence constraints. Scenarios with no constraints or with partial constraints are also considered. Application of the precedence constraints was motivated by an industrial scenario where at first a robot has to pick up a detail from a conveyor and after that drill it. Cases where a robot has to change its tool during the sequence execution were integrated into the planning process. The branch-and-bound technique [64] was applied to obtain the exact ATSP solution. The method proposed by Dubowsky et al. is independent from a particular ATSP solving algorithm. The main limitation is that only a single inverse kinematics solutions is taken into account. Therefore, an obtained sequence is far from being optimal in real-life applications.

Abdel-Malek et al. [1] improved a manufacturing cell, where a TSP problem has to be solved for a 3-DOF robot, while considering multiple solutions of the inverse kinematics. The location of a robot's base was optimized so that the cycle time is minimized. For that, a grid search scheme was applied. It limits the problem to X and Y coordinates, which means that the robot's location cannot move vertically in a cell. Every cell of the grid is evaluated with regard to cycle time as if the robot was placed there. The position with the shortest cycle time is considered to be the optimal location for the robot's base in the given grid. However, there is a trade-off between an accuracy and computation time when choosing the granularity of the grid. Later Abdel-Malek et al. [2] extended the approach with support of several types of robots: cartesian, cylindrical, spherical and articulated.

Edan et al. [35] proposed an approach for a robot task sequencing. It was motivated by the fruit picking scenario. The main idea of the approach is straightforward: calculate a cost matrix and then apply the Nearest Neighbor algorithm to solve the TSP. The goal of the approach is to minimize the cycle time while considering both robot kinematics and dynamics. It was suggested that the approach can also be applied for a task-based robot choice. Since the algorithm requires a complete cost matrix, it makes integration of collision-free planning unfeasible for real-life scenarios, due to the need of extremely large computational time.

Zacharia et al. [103] presented a method based on the Genetic Algorithm (GA) and involved multiple inverse kinematics solutions in the optimization process. The method is capable of finding a task sequence for a 6-DOF robot in a feasible time. In order to take multiple inverse kinematics solutions into account, an innovative encoding for the GA was introduced. The GA chromosome consists of two parts. The first part encodes a sequence and the second part encodes robot configurations for the corresponding sequence.

Later Baizid et al. [16] extended the approach by Zacharia et al. [103] with a robot base layout optimization. It was done by adding a third component into the chromosome that denotes the position of a base. Although the heuristic approaches applied in [103] [16] allow an efficient search space exploration, it does not guarantee to

find an optimal solution. Such iterative search heuristics can be stopped after exceeding the maximal number of iterations. It does not reflect how far is the obtained solution from optimum.

Reinhart et al. [85] proposed an algorithm for solving the sequencing problem for open-end curves in remote-laser-welding applications. At first, every curve is represented with one point and a random tour through such points is constructed. Then random a swap of points in the tour is performed to improve the tour. Later the directions along the curves are randomly chosen. The algorithm randomly changes the directions to obtain a better tour. Although, a certain improvement of the initial solution can be obtained, random swaps efficiency should drop down with computation time.

Loredo-Flores et al. [66] developed a GUI that allows users to manually create a robot's path. Later, the constructed sequence is optimized by GA and Simulated Annealing (SA) approaches. The output is a sequence of robot configurations with the objective to minimize the robot joint displacement, i.e., C-space distance.

Yang et al. [100] considered the task sequencing problem for welding applications with an objective to minimize the time and the distortion of a welding process. The general idea is to split tasks into two categories by their influence on the product quality, i.e., welding deformation. A sequence of tasks from the category with minimal influence is optimized by an Elastic Net Method (ENM) [81] so that the cycle time is minimized. The category with high influence on deformation is optimized with a GA to minimize the product distortion. The main idea of the ENM is to apply two forces on the net: one is trying to keep the points closer to each other and another force attracts them to the goal positions. When two optimized sequences are obtained, they are merged with the nearest neighbor criterion. The proposed approach was evaluated in a 2D environment, as the calculation of the distortion in 3D is a much more complicated problem. However, the general ideas are domain-independent and, therefore, can be reused in other domains.

Kolakowska et al. [57] motivated the robot task sequencing with a painting scenario. The authors proposed an approach that aims at solving a task scheduling problem and is completely independent from the task and motion planning stages. The main contribution of the paper is the mathematical constraint model that can be further used in any constraint solver. The proposed approach is efficient for scenarios with up to 10 goals. The downside of applying an exact solver is the large computational time for larger scenarios. Due to the large search space, the solver was not able to finish calculation of solution for some scenarios within 40 hours.

Wurll et al. [98] introduced Multi-Goal Path Planning (MTP). A cost matrix for the MTP is almost impossible to pre-calculate because distances should be obtained by taking obstacles into account and collision-free planners are computationally expensive algorithms. Therefore, the costs are unknown and have to be obtained during solving. The solution process was split into three stages: controlling, path planning and shortest sequence planning. Controlling is a stage where a decision is made, which pair of start

and goal configurations should be sent to the path planning stage. They proposed four different strategies for the controlling stage. A collision-free path between two configurations is calculated in a C-space grid by using the best-first search method. In the third stage, the approach uses a GA to solve the TSP. The proposed approach loops in these three stages and guarantees that every newly obtained tour will be no worse than the previous one. Later Wurll et al. [97] extended this approach by parallelization of a bidirectional search to reduce computation time and used the A* search for collision-avoidance method.

Petiot et al. [81] solved the task scheduling problem for industrial robots with an ENM. The collision avoidance feature was integrated as another repulsive force into the ENM, i.e., a penalty term in the ENM function. The limitation of this approach is that obstacles must be convex and easily representable in the C-space. The algorithm performs well on robots with small number of DOFs (e.g., with 2 or 3). But it is difficult to generalize the approach for robots with high number of DOFs, as it becomes computationally expensive to map obstacles to the robot configuration space.

Spitz et al. [93] proposed a general framework suitable for a variety of robots. Domain specific knowledge could be integrated into the planner, as a local planner or an extension of the roadmap created with PRM. Based on this roadmap, another graph is created, which consists of only target points and all superfluous points are deleted. It is then used for tour optimization with known TSP solvers. The method assumes that task is reachable with one robot configuration that limits potential of optimization due to committing the multiple inverse kinematic solutions of the manipulator.

Saha et al. [86] observed the problem of multi-goal planning among the robot configurations. The use of C-space points instead of T-space points limited the potential freedom for optimization. Therefore, the approach was improved by specifying the input points in T-space [87]. Lower-bound approximation is used for computing the optimal path (normally the shortest distance) and the call of the collision-free path planner is delayed. A bidirectional tree-expansion PRM was used for collision-free planning. For TSP solving the Prim's algorithm was applied [27]. The main assumption of this approach is that calculation of a sequence is computationally less expensive than calculation of the collision-free movements. Therefore, collision-free paths are calculated after the sequence is obtained. If the collision-free path cost is significantly higher than the estimated cost, the sequence should be recalculated taking the newly obtained path cost into account.

Gueta et al. [46] proposed a method for solving the TSP problem for goals located on a revolvable table. The table and the robot are seen as one redundant system. The goals are put into clusters and a sequence of these clusters is calculated with the 2-Opt algorithm. Then for each cluster two goals are determined as connectors for the previous and following clusters in the sequence. After that the TSP is solved with Lin–Kernighan (LK) heuristic in each cluster. The distance between the goals is considered as a straight line in C-space. The system redundancy is used to select such configurations of the robot that allow to obtain a collision-free path.

Bu et al. [22] presented a method for the optimization of the robot base layout and robotic task sequence. The objective is the minimization of the cycle time. The input sequence could be either fixed (linear topology), unfixed (complete directed graph topology) or fixed and unfixed sequence (mixed topology). The first step of the solution process is to calculate the 5D space of the possible base locations, where 3D stands for position, 1D stands for orientation around the base axis and 1D shows if the base axis is upward or downward. Then this 5D space is divided into discrete grid cells. Optimization of the sequence is done with an Ant Colony Algorithm (ACA) for each cell. The next step is to find the local optimum for the robot base position and orientation with the pattern search for each cell in relation to the costs of the obtained sequences. Finally, the global optimum is the result of comparing all local optima. The approach does not provide automatic smart technique for collision avoidance. Obstacle-avoidance is organized in a way that when the collision occurs between the robot and a workpiece, an intermediate frame should be added between the two target frames.

Xidias et al. [99] investigated the problem of determining the optimal sequence of task points for an articulated robot with the obstacle avoidance feature in a 2D environment. The problem is divided into the optimal sequencing problem – TSP and the path planning problem. The objective is to obtain the shortest cycle time. The proposed concept is an adaptation of the GA, proposed by Zacharia et al. [103]. Furthermore, they introduce the Bump Surface concept for collision checking. The general idea of the Bump-surface is to represent the 2D working environment in a unified way by mapping it into a 3D space. It is done by discretizing the 2D map into a uniform grid of points and assigning a value from $[0, 1]$ to each point. If the point is outside the obstacle then the value is 0 and if it is inside then the value is from the range $(0, 1]$. The search space is represented as one mathematical entity, a B-spline. Therefore, the complexity of the problem does not depend on the complexity of the environment (shape and location of obstacles). The limitation of this approach is that the working environment is considered to be 2D.

Further, Zacharia et al. [102] extended two approaches: [103] by adding obstacle avoidance and [99] by adapting it to the real-world 3D environment. The optimization problem was solved with the GA. The chromosome was extended with the information about the intermediate configurations required for collision-free movement. The search space is represented with the Bump-surface as a single mathematical entity. The innovative characteristic of the proposed approach is that sequencing, motion planning and obstacle avoidance are incorporated into the objective function. It leads to the situation, when all paths (both collision and collision-free) are considered to be feasible but collision-free paths are preferred by the search algorithm.

Huang et al. [52] observed the task of selecting a specific manipulator for the particular multi-goal task planning in a system that consists of a robot arm and a position table. The input parameters for the algorithm are the list of points and the candidate manipulator specification (Denavit–Hartenberg parameters). They proposed to decompose the problem into the three nested stages. Less computationally expensive algorithms are chosen for the inner loops due to the fact that stages are nested in

each other and, therefore, the inner loops have to be repeated more often than outer loops. The output is the manipulator structural configuration as well as positions of the manipulator and rotational table.

Lattanzi et al. [61] proposed the extension to the LK heuristic by integrating collision-free planning into the cost function. As it is computationally expensive to obtain a collision-free path for the whole robot, the collision-free path was calculated only for the robot's end-effector. The robot end-effector path between two Cartesian points is the shortest (in terms of robot motion time) when the robot performs PTP movement. It was necessary to make sure that this PTP movement does not cause collision of the end-effector. PTP joint path is transferred into the end-effector path with FK and represented as a poly-line. In case of a collision, an intermediate path-point is inserted and PTP movement is recalculated. As LK is a tour improvement heuristic, it was evaluated with different tour construction heuristics: the shortest/farthest insertion and the greedy approach. The work was motivated by a visual inspection task in an industrial setup.

3.4.2 Sequencing Complex Tasks

Within the last five years, the use of task shape potential for optimization attracted a lot of attention. Further these approaches are discussed in detail.

Gentilini et al. [43] used TSPN for modeling the sequencing problem of camera inspection tasks. The robot has to take a picture with a camera mounted on its end-effector. The problem does not require a precise position but rather an area from which the pictures have to be taken. TSPN problem was formulated as Mixed-Integer Non-Linear Program (MINLP). It was shown that searching for an exact solution requires unreasonable amount of time. Therefore, a heuristic was introduced to a MINLP solver to speed up the calculation time. The method was evaluated on test instances with up to 16 areas. The obtained solutions are very close to the optimum, i.e., the average error is less than 0.6% on tests with up to 16 areas. The optimization process is based on the Euclidean distance metric, thus the robot kinematics flexibility is not considered.

Sequencing of tasks with extra freedom was investigated by Kovács [59]. This problem was motivated by a specific industrial application – robot remote laser welding. An efficient combination of the TSP solver (Farthest Insertion Heuristic, Tabu search and 2-Opt) and the path planning method was proposed. Collision-free path planning is not considered during the sequencing, but rather done after the sequence is calculated. The planning is done in Cartesian space and after that the solution is converted into the robot configuration space by using inverse kinematics transformation. The minimization objective is the lexicographical order of the three criteria: minimal cycle time / minimal path length (scanner head) / minimal path length (laser end point).

Vicencio et al [96] proposed GTSPN problem. It is the extension of the well-known TSPN problem, where the neighborhoods can be clustered and each cluster should be visited one. The Hybrid Random-Key Genetic Algorithm (HRKGA) was proposed

to solve the GTSPN. It was shown that HRKGA is capable of effective and efficient solving of scenarios up to 300 neighborhoods. The algorithm was tested on 3D and 7D instances.

Faigl et al. [38] presented the multi-goal path planning problem for goal regions (MTPGR) in the polygonal domain. The areas could be arbitrary shapes and are allowed to overlap. The proposed approach is based on self-organizing map (SOM) algorithm and was evaluated in a 2D environment. The presented method appears to be very efficient and is capable of solving test cases with up to 106 areas in less than 6 seconds.

Later Gueta et al. [47] extended the previously described approach [46] by representing the goals as T-space points with two parameters that provide additional freedom: 1) distance of the camera from the goal point along the approach vector and 2) rotation about the approach vector. In addition, the tool attachment was optimized using a Simulated Annealing (SA) method.

Gentilini et al. [44] used the TSPN to model the task sequencing problem of the camera inspection tasks for the case when the inspected object is located on the rotated table. The neighborhoods (sets of robot configurations for each goal placement) are represented as curves in the seven-dimensional configuration space (6D is the robot configuration space and 1D is added by the table rotation). The paper concentrates on exploring the redundancy of the system, rather than kinematics, therefore, only one configuration was chosen among the set of possible inverse kinematics solutions. Due to the search space explosion, the exact methods are limited to a small amount of goals. Therefore, the heuristic Hybrid Random-key Genetic Algorithm was applied. A single/multiple query BiRRT was used to construct the roadmap that is further used for collision-free planning.

Erdős et al. [37] proposed the improvement of the previous work of Kovács [59]. The objective was the same – minimization of the cycle time of the remote laser welding operations. During the planning stage only the scanner-workpiece was checked for collision, but not the whole robot. Due to the search space fast growth, collision-free path planners for the whole robot have to be applied only after the sequence is obtained.

3.5 Robot Trajectory Optimization

Trajectory optimization is one of the well-known problem in robotic domain. The comprehensive overview on state of the art in robot trajectory optimization can be found in the survey [14]. In general case, the problem is to generate a trajectory between starting and goal robot configurations that satisfies the objectives the constraints. The objective could be minimum time or jerk optimization. The constraints are often imposed by the robotic kinematics, e.g., robot joint angular or velocity limits or task, e.g., following a certain path.

In this thesis we solve the problem of following the relaxed effective task. The goal is to choose such an end-effector path from the relaxed effective task that in conjunction

with a given motion law leads to the minimal cost C-space trajectory. In the following, we discuss commonalities and differences to the state-of-the-art problems.

The problem proposed in this thesis is different from the trajectory tracking problem [15], as we are concerned in making the end-effector trajectory more suitable for a robot, rather than in a precise following of the given end-effector trajectory.

In general, smooth functions are used for the T-space path interpolation. For example, Liu et al. [65] proposed the optimization method for calculating time-optimal and jerk-continuous trajectories for robot manipulators. The T-space trajectory is represented with Cubic spline, i.e., twice continuous differentiable. C-space trajectory is multi-degree B-spline with bounded and continuous velocity, acceleration and jerks. However, in contrast to the problem proposed in this thesis, it is assumed that via-points should be visited strictly and no relaxation is allowed.

This thesis problem is different for trajectory optimization for the supporting tasks. Supporting task allows for a lot of freedom while motion, often the linear movement in C-space is used. It is the shortest movement of the robot between two configurations and called point-to-point (PTP) motion. In that case, often velocity profiles for the robot joints are synchronized [28].

The trajectory planning problem for the C-space constrained trajectory between two T-space points was addressed by Chettibi et al. [24]. The constraints represented as upper and lower bounds were set in C-space, i.e., for every joint. Therefore, this approach suits better trajectory optimization for the supporting tasks.

Another approach for industrial robot trajectory optimization was proposed by Gasparetto et al. [42]. Their method does not require to specify desired trajectory duration. Proposed method allows to minimize execution time and the jerk value. By using weight coefficients, a balance between fast and smooth movement can be obtained. Algorithm input is the T-space path, i.e., sequence of via-points. These via-points are then converted to C-space path. Then C-space points are interpolated with cubic or firth-order B-splines. Any suitable optimization technique could be applied to minimize the cost function. The output of the algorithm is the vector of time intervals between pairs of via-points.

Continuous end-effector path planning problem for the effective tasks instead of a point-to-point movements was observed by Olabi et al. [77]. A robot has to strictly follow the end-effector path and its motion-law was optimized. In our problem, we do exactly the opposite – the path geometry is optimized, however, the motion-law is followed strictly.

3.6 Conclusion

The majority of robotic task sequencing approaches considers tasks as primitive goals represented as points in T-space or in C-space. Approaches that allow sequencing

of tasks with complex shapes are the most relevant to this thesis. However, these approaches have several limitations.

Some approaches for complex task sequencing do not involve information about robot kinematics, e.g., [38][43][96]. Other approaches either observe only one solution out of a set of possible solutions, e.g., [44], or apply inverse kinematics after a sequence of tasks is calculated [59][37]. In this thesis, we propose two strategies: one that postpones selection of a robot's configuration (i.e., Component1 $\rightarrow$ Component2) and one that involves robot kinematics optimization into the planning stage (i.e., Component1 + Component2).

Relevant research proposed only one approach to deal with multiple inverse kinematics solutions [47]. It makes optimization only in one neighborhood, where a neighborhood is a T-point with a relaxation in a translation along approaching axis Z and in a rotation around axis Z. In the thesis, effective tasks are relaxed with an infinite number of neighborhoods.

The aforementioned approaches for the optimization of a robot's trajectory often ignore freedom of a path, i.e., freedom of the position and the orientation of each point. Those approaches that exploit path freedom are either domain-specific or not applicable for industrial applications. The idea proposed in this thesis extends their scope of applications. The proposed problem definition is domain-independent and formulated with no dependence on the solving approach. As a consequence, there are no special requirements for constraints or a cost function.

4. Component1: Relaxed Effective Tasks Sequencing

> Business leads the traveling salesman here and there, and there is not a good tour for all occurring cases; but through an expedient choice and division of the tour so much time can be won that we feel compelled to give guidelines about this.
>
> German handbook "Der Handlungsreisende", 1832.[1]

In this chapter, we propose approaches to optimize a sequence and entry points of relaxed effective tasks, see Figure 4.1. A formalization of this scenario leads to a Traveling Salesman Problem with Neighborhoods (TSPN). TSPN is a generalization of TSP where points are substituted with areas and the objective is to find a minimal-cost tour through a set of regions, visiting each of them once. In the following, we present two approaches for solving TSPN: Constricting Insertion Heuristic for tour construction and Constricting 3-Opt for tour improvement.

The remainder of this chapter is organized as follows. We present the industrial scenarios that motivate this chapter in Section 4.1. Section 4.2 outlines related work. Preliminaries on applied sub-problems and algorithms are described in Section 4.3. Proposed solution approaches are presented in Section 4.4.

[1]The translation from the German original was done by Linda Cook for the book "The Traveling Salesman Problem: A Computational Study" by David L. Applegate et al.

Figure 4.1: Overview of Component1.

4.1 Motivation

It is obvious that an efficient production process requires an optimized sequence of effective tasks. If for every effective task, the entry point is fixed, the task sequencing problem can be translated to the Traveling Salesman Problem (TSP) [10] which aims at computing a minimal-cost cyclic tour that visits all points once. However, in reality, many effective tasks do not require strong determinism and often allow certain degrees of freedom. For example, in the application scenario in Section 2.2, it is not important where individual cuttings should start and/or end. Therefore, the choice of an entry point for each effective task significantly influences the cost of a task sequence.

There are many other examples apart from the stated cutting scenario that provide freedom of execution: robotic manufacturing of a toboggan[1], laser-welding [59] or an object observing task with a camera installed on a robot end-effector [43]. Often an allowed deviation could be significant. For example, From et al [40] claimed that deviation up to 20% of an end-effector orientation in a painting scenario does not affect the quality of coating.

Every relaxed effective task is defined as an entry area. Therefore, in the following notation, a relaxed end-effector task $Task_{EF}^{Rel}$ can be understood as an polyhedron $A \in SE(3)$. Therefore, the problem of sequencing relaxed effective tasks stated in Section 2.4.1 can be formalized as the Traveling Salesman Problem with Neighborhoods as follows:

> Given a set of n polyhedra $A = \{A_1, ..., A_n\}$, find a minimal-cost cyclic tour $T = (p_1, ..., p_{n+1})$ such that it visits A_i in a point p_i and $p_1 = p_{n+1}$.

In the remainder of this chapter, notation from the TSPN domain will be used. In order to ease the explanation, we restrict ourselves to 2D space (i.e., $\mathbb{R}^2$) and use a Euclidean distance function $d(p, q)$ that defines the cost of moving between two points $p, q \in \mathbb{R}^2$. Of course, generalization to higher dimensional spaces as well as other distance functions is possible.

[1]Kuka Roboter: see http://www.kuka-robotics.com/en/solutions/solutions_search/L_R131_Deburring_of_plastic_toboggans.htm, accessed on August 26, 2015

4.2 Related Work

Heuristics for TSP-like problems can be split into two categories: construction and improvement heuristics [54]. Construction heuristics construct a feasible solution. Improvement heuristics start with a complete tour and try to reduce its cost by modifying it. The iterative improvement process stops when a stopping condition is met, e.g., no more improvement is possible or a maximal number of iteration is reached.

The TSPN was originally introduced by Arkin and Hassin [11]. Later, it received significant attention in the domain of approximation algorithms [12, 73]. Furthermore, it has already been applied in the field of robotics. For example, Gentilini et al. [43] applied this idea to a real-world use case where a robot with a hand-mounted camera takes pictures of an object from different positions. They formulated this problem as a TSPN and implemented a heuristic to speed up a Mixed-Integer Non-Linear Programming solver. This method shows good, close to optimum results on instances with up to 16 tasks. However, in some industrial domains[2], 50 or even more tasks are common.

Mennell [71] proposed an approach for solving Close-Enough Traveling Salesman Problem (CETSP) which is a special case of TSPN. Its goal is to find a minimal-cost tour such that a salesman visits every point exactly once by moving within a certain distance from it. Thus, an area describing the neighborhood of each point is a disk in 2D space. A generic three-stage approach was proposed: (1) find intersections between disks, (2) represent every intersection area with a point and calculate a TSP tour, (3) optimize the previously found point sequence with a TPP solving method. Multiple variations and combinations of different algorithms for these stages were investigated.

Elbassioni et al. [36] proposed a method for a Discrete TSPN, similar to Generalized TSP (GTSP)[3]), where areas are replaced with sets of points and the goal is to find a minimal-cost path visiting exactly one point from each set. First, areas are sorted by their diameter. Then, for every area, starting from the area with the smallest diameter, an inner area point is picked up. The algorithm selects a new point as close as possible to the already chosen points. When all areas are assigned with inner points, the algorithm optimizes the points allocation and computes a TSP tour. The authors claim that the proposed method can also be applied for continuous TSPN (we refer to this problem simply as TSPN) under an assumption that the new closest point can be efficiently found in an infinite set of points.

The TSP got much larger attention from researchers than the TSPN. As a result, a large number of effective algorithms exist for TSP. A naive idea would be to use algorithms from the TSP domain for the TSPN. This could be done in several ways.

One approach is to convert a TSPN problem to a GTSP by replacing areas with a sets of points and solve the obtained problem with a GTSP approach.. The objective

[2]Pan et al. [80] showed an example of welding applications with 500 goals for two robots (250 goals for each).

[3]It is also referred to as One-of-a-Set TSP, Group TSP, Multiple Choice TSP or Covering Salesman Problem.

is to find a minimum-cost path that passes through one point of each set. Oberlin et al.[76] proposed to solve a GTSP by converting it to a TSP, solving the obtained problem and transforming a TSP solution to a GTSP. Though this process is possible in theory, it immensely increases the search space and becomes practically infeasible [88]. Another drawback of representing a TSPN as a GTSP is the errors appearing due to the discretization of areas.

Another way to apply algorithms from the TSP domain to TSPN is to represent every area with one point. It transforms a TSPN into two subproblems: a TSP and a Touring-a-sequence-of-Polygons Problem (TPP). TPP is an NP-hard problem where the goal is to find the shortest path that passes through a given sequence of areas [33]. In this chapter, we present a method to solve TSPN based on this idea. In contrast to other approaches, our heuristic simultaneously solves task sequencing (TSP) and allocation of points inside areas (TPP).

4.3 Preliminaries

This section gives a brief introduction to formalization of relevant problems. It also explains general ideas of existing algorithms that are used as a foundation for our methods.

4.3.1 Involved Sub-Problems

The well-known Traveling Salesman Problem is formalized as follows:

> Given a weighted undirected graph $G = (V, E)$, where V is a set of n vertices and E is a set of edges. The objective is to find a minimal-cost cyclic tour $T = (v_1, ..., v_{n+1})$ that visits all vertices only once and $v_1 = v_{n+1}$.

Another related problem is the Touring-a-sequence-of-Polygons Problem. It is defined as follows:

> Given a sequence of n polygons $A = (A_1, ..., A_n)$, find a minimal-cost cyclic tour $T = (p_1, ..., p_{n+1})$, such that it visits A_i in a point p_i and $p_1 = p_{n+1}$.

There are two types of TPP: floating and fixed. In the fixed TPP, start and end points have to be defined and their positions are fixed. In the floating TPP, start and end points are no longer required. The floating TPP is relevant to this paper and simply referred to as TPP. Note that for TPP, an ordering of the polygons is required as an input data. This order is not changed during tour calculation. In contrast to TPP, an input of TSPN contains a set of polygons instead of their sequence.

4.3.2 Involved Sub-Algorithms

Before presenting algorithms for TSPN, we provide an explanation of the TSP and TPP existing heuristics which are involved in the TSPN methods.

4.3.2.1 Insertion Heuristic

One of the most well-known algorithms for tour-construction is Insertion Heuristic (IH) [48]. The general structure is represented in Algorithm 4.1.

The algorithm is based on three strategies $S1$, $S2$ and $S3$. $S1$ denotes a strategy to construct an initial tour. Often an initial tour is either a triangle tour (i.e., a tour consisting of three points) or a tour that follows the points that form the convex hull border of V. $S2$ is a strategy to choose a point v that is not yet in the tour T. $S3$ is a way to choose a position in T where point v should be inserted. The strategies $S2$ and $S3$ are repeated while T is a partial tour, i.e., not all the points of V are in the tour.

Algorithm 4.1: Insertion Heuristic

Input: Weighted graph $G = \{V, E\}$
Output: Tour $T = (v_1, ..., v_n)$

Initialize sub-tour T with strategy $S1$;
while T *is a partial tour* **do**
    Choose vertex $v \notin T$ with strategy $S2$;
    Add v to the tour T with strategy $S3$;
end
return T;

After a complete tour is obtained, it is possible to apply tour-improvement heuristics such as 2-Opt or 3-Opt [49]. The 2-Opt and 3-Opt are local search algorithms that are certain cases of the K-exchange algorithm [63].

4.3.2.2 3-Opt Heuristic

The K-exchange algorithm was developed by Lin [63]. The basic idea is to sequentially select K edges from a tour and reconnect them in all possible ways. If a reconnection causes a decrease of the tour cost, the algorithm starts from the beginning; otherwise, the next K edges are selected. The algorithm stops when there are no more exchanges that could improve the tour.

The evaluation in [54] shows that if K is greater than three, the effectiveness of the approach decreases, i.e., the spent computational time is no longer pays off for the gained cost improvements. Therefore, normally the value K=3 is used and the corresponding method is referred to as 3-Opt [21]. Later 3-Opt received a lot of attention and was successfully applied in different areas, e.g., the UAV routing problem [72] or the Sequentially Ordered Problem [50].

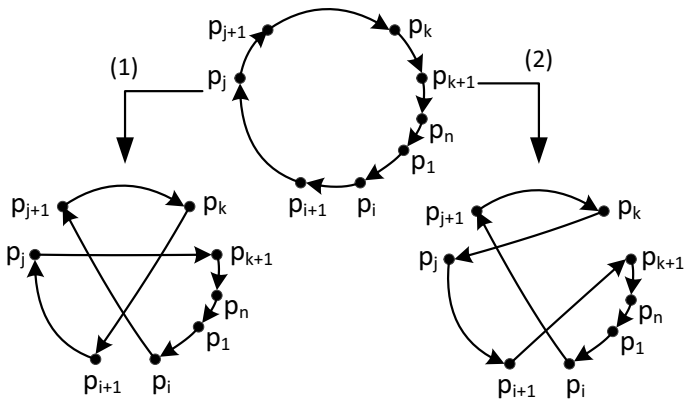

Figure 4.2: Two possible replacements of edges (p_i, p_{i+1}), (p_j, p_{j+1}) and (p_k, p_{k+1}) in 3-Opt.

There are seven possible ways of reconnecting three edges in 3-Opt. It was shown in [84] that only two ways of reconnecting should be considered to make an admissible tour, see Figure 4.2. The decrease of a cost in 3-Opt that can be achieved after reconnection is calculated as the difference between the costs of newly added and deleted edges.

4.3.2.3 Rubber-Band Algorithm

Rubber-Band Algorithm (RBA) is a method to solve the TPP was proposed by Pan et al. [78]. The general structure of the RBA is presented in Algorithm 4.2.

Algorithm 4.2: Rubber-band algorithm

Input: Sequence of areas $A = (A_1, ..., A_n)$, accuracy ε
Output: Tour $T = (p_1, ..., p_n)$

```
1 Construct a sequence T = (p_1, ..., p_n) such that p_i ∈ A_i;
2 while Desired accuracy ε is not reached do
3 |   foreach p_i ∈ T do
4 |   |   p_i ← argmin_{p_i ∈ ∂A_i} (d(p_{i-1}, p_i), d(p_i, p_{i+1}));
5 |   end
6 end
7 return T;
```

The RBA constructs a feasible tour $T = (p_1, ..., p_n)$ by allocating points inside the areas $p_i \in A_i$ and then iteratively improves it. Optimization is performed for one area at a time in the same order as areas appear in the tour. A position of a point p_i of area A_i is improved by selecting another position new_p_i on the border ∂A_i of the area with the minimal distance to its tour neighbors p_{i-1} and p_{i+1}, see Figure 4.3 and line 4 in Algorithm 4.2.

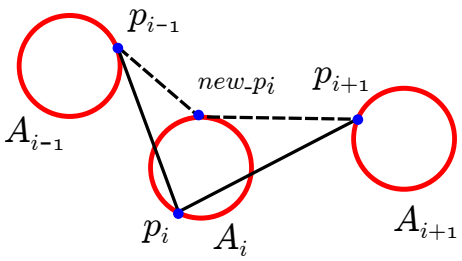

Figure 4.3: Constricting point p_i to its neighboring points p_{i-1} and p_{i+1}. Edges before the constriction are marked by solid lines. Edges after the constriction are marked by dash lines.

One iteration of the improvement cycle is finished when this procedure is performed for all areas. The RBA stops when a maximal number of iterations has been performed or a desired accuracy ε was reached, i.e., the difference between tour lengths on iteration j and $j+1$ is less than ε.

Following the TPP definition, the solution of the problem is a tour consisting of $n+1$ points. Note that Algorithm 4.2 returns a tour that consists of n points. It does not contradict the definition, as it is easy to modify the tour by extending it with a point p_{n+1} that is equal to p_1. Therefore, tours provided by the RBA are correct.

4.4 Solution Approaches

TSPN can be understood as a generalization of TSP and TPP, see Table 4.1. We rely on the idea, that by alternating TSP and TPP solving processes one can obtain near-optimal solution for TSPN. The alternating of solving processes here means that on every iteration of the TSP solving algorithm (i.e., IH or 3-Opt), points are constricted to each other with TPP solving algorithm (i.e., RBA) to optimize their location in the areas with respect to the obtained sequence.

Before starting with presenting the methods, an important note about the RBA modification has to be made. As previously described, RBA takes two parameters as an input: a sequence of areas A and an accuracy value ε. In the algorithm, an optimal sequence of points $T = (p_1, ..., p_n)$ is calculated by optimizing allocation of the points in the areas. We introduce mRBA which is a small modification of RBA where a sequence of points T is included as additional parameter in the input, i.e., line 1 in Algorithm 4.2 is omitted.

Searching for new_p_i is a geometrical task and could be performed with any approach. For some shapes, e.g., polygons, analytical solution is possible. We assume that in general case, any metric could be involved and tasks might be arbitrary dimensions and shape. In that case optimization techniques can be applied. One-dimensional Golden search or Bisection method [83] can be used for one dimensional tasks, e.g., closed contours. Gradient descent, Pattern Search [51] or any other multi-dimensional

Problem	Input	Optimize	Output
TSP	set of points	sequence	sequence of points
TPP	sequence of areas	location of points	sequence of points
TSPN	set of areas	location and sequence	sequence of points

Table 4.1: The differences between TSP, TPP and TSPN.

optimization approach can be used for multi-dimensional tasks. In the following realization, Bisection method is used for closed-contour tasks. It searches in the curve domain [0,1]. Optimization is done until accuracy μ is reached. Therefore, mRBA requires four input parameters: A, P, ε and μ.

In the remainder of this section, we present two approaches: Constricting Insertion Heuristic for tour construction and Constricting 3-Opt for tour improvement.

4.4.1 Constricting Insertion Heuristic

Constricting Insertion Heuristic is obtained from Insertion Heuristic by modifying the strategies in Algorithm 4.1. Specialized implementation of the insertion strategies makes CIH capable to solve TSPN efficiently. CIH is shown in detail in Algorithm 4.3.

Several functions are involved during calculation. Function $Count(R)$ returns the number of elements in the set R. Function $Insert(T, p, i)$ inserts point p to the tour T at the position after the element with index i. Function $Remove(T, p)$ removes point p from the tour T. Function $ConvexHullBorderTour(P)$ returns a tour consisting of points that form the *border* of the convex hull of the set P. One example of such output is illustrated on part 2 in Figure 4.4. Function $Length(T)$ takes a sequence of points T and returns the tour length.

The algorithm takes a set of areas A as an input. In the line 1 of Algorithm 4.3, set P is constructed in a way that point p_i belongs to area A_i[4]. In the lines 2 and 3 in Algorithm 4.3 a set of points P is split into two subsets: T and R. T is a tour which follows the points that form a convex hull border of P. R consists of all the remaining points from P that are not in the border of convex hull.

It could be the case that all points from P belong to the border of the convex hull. This check is performed in line 4 of Algorithm 4.3. In that case T is already the desired sequence of areas to visit and mRBA is applied to find the optimal point allocation within the obtained sequence.

In case if not all points from P belong to the border of the convex hull set, points from the remainder R are selected one by one and inserted in the tour T. Line 9 in

[4]In the following test instances in evaluation, p_i is a geometrical center of the ellipses.

Algorithm 4.3: Constricting Insertion Heuristic

Input: Set of areas $A = \{A_1, ..., A_n\}$, desired accuracies ε, μ
Output: Tour $T = (p_1, ..., p_n)$

1 Construct a set $P = \{p_1, ..., p_n\}$ so that $p_i \in A_i$;
2 $T \leftarrow ConvexHullBorderTour(P)$;
3 $R \leftarrow (P - ConvexHullBorderTour(P))$;
4 **if** $R = \emptyset$ **then**
5 $\quad T \leftarrow mRBA(A, T, \varepsilon, \mu)$;
6 $\quad$ **return** T;
7 **end**
8 **while** $R \neq \emptyset$ **do**
9 $\quad p_{temp} \leftarrow \underset{p_q}{\operatorname{argmin}}(d(p_q, t_j))$, where $p_q \in R$, $t_j \in T$, $q \in [1, Count(R)]$, $j \in [1, Count(T)]$;
10 $\quad L \leftarrow \infty$;
11 $\quad T_{temp} \leftarrow T$;
12 $\quad$ **for** *i=1 to* $Count(T)$ **do**
13 $\quad\quad Insert(T_{temp}, p_{temp}, i)$;
14 $\quad\quad T_{temp} \leftarrow mRBA(A, T_{temp}, \varepsilon, \mu)$;
15 $\quad\quad$ **if** $Length(T_{temp}) < L$ **then**
16 $\quad\quad\quad L \leftarrow Length(T_{temp})$;
17 $\quad\quad\quad insert\_index \leftarrow i$;
18 $\quad\quad$ **end**
19 $\quad\quad Remove(T_{temp}, p_{temp})$;
20 $\quad$ **end**
21 $\quad Insert(T, p_{temp}, insert\_index)$;
22 $\quad Remove(R, p_{temp})$;
23 $\quad T \leftarrow mRBA(A, T, \varepsilon, \mu)$;
24 **end**
25 **return** T;

Algorithm 4.3 reflects the strategy $S2$ where the point p_{temp} is chosen so that it is the closest point to one of the points from T.

Strategy $S3$ (lines 10-23) in Algorithm 4.3 is a sequential insertion of p_{temp} to all possible positions within tour T performing mRBA algorithm every time and measuring the tour distance. After all combinations are checked, p_{temp} is inserted to the position that gives a minimal increase of the tour length and mRBA is performed. Afterwards, p_{temp} is deleted from the remainder set R. The algorithm stops when all points from R are inserted to T. The obtained tour T is the desired tour.

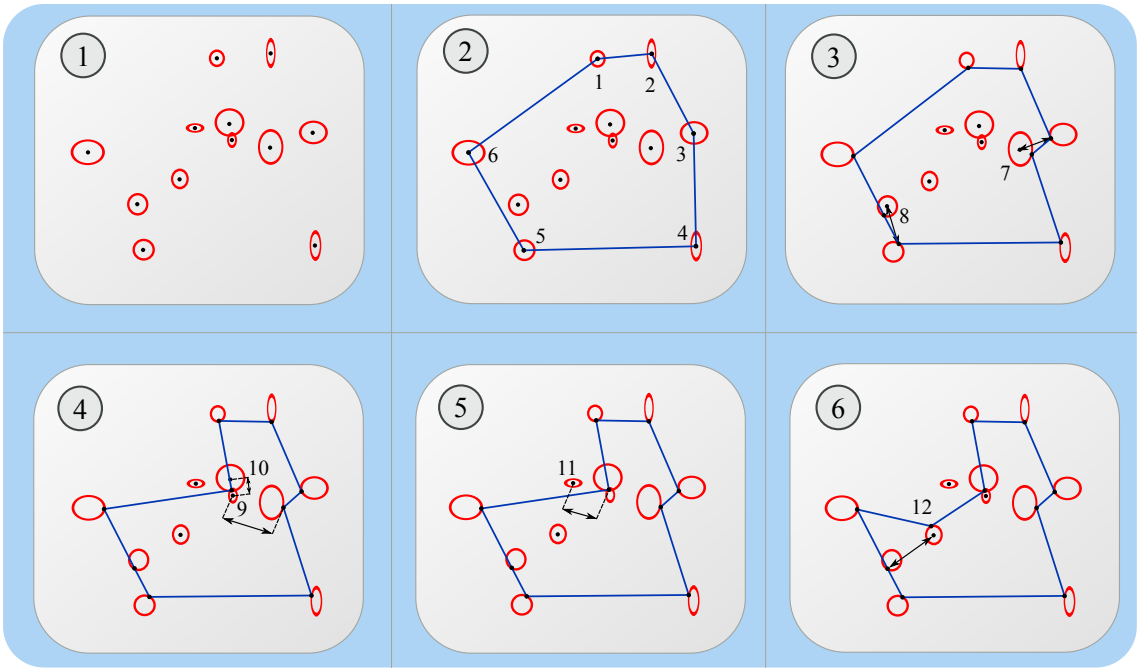

Figure 4.4: Workflow of CIH on test instance "tspn2DE12_2"

Example of CIH Workflow

In the following, a test instance "tspn2DE12_2" developed by Genitili et al. [43] for TSPN with 12 ellipses is solved by CIH. CIH is a tour construction heuristic and starts from any arbitrary sub-tour. However, it is more efficient to take points that form the border of the convex hull as an initial tour. Therefore, 6 points are added to the tour on part 2 of Figure 4.4. If all points are inserted, their optimal sequence is found and only their locations within the areas should be optimized by mRBA. However, in this example 6 points are left. Therefore, new points 7–12 are added one by one to the tour so that a point which is the nearest to the tour is picked up. The nearest distance is denoted with an arrow in Figure 4.4. For example, on part 3 point 7 is picked up as it is the nearest to the point 3 in the existing tour in part 2. Further, point 8 is added to the tour, because it is the nearest point to the point 5, which is already in the tour. The parts 2–6 in Figure 4.4 show how the algorithm adds new points to the tour one after another.

Note that some iterations are combined at one part of Figure 4.4 (e.g, points 7 and 8 in part 3) as the picture was not changed visually.

CIH stops when there is no point left outside of the tour. In this example, an optimal solution was obtained as it is illustrated on part 6 of Figure 4.4. An animated process of CIH is presented online[5].

Note that TSP tour-improvement heuristics could improve CIH solution for TSPN by changing the sequence. However, this improvement may cause such a case that the

[5]CIH visualization: see https://cse.cs.ovgu.de/cse/robotics/tspn/, accessed on August 26, 2015

Algorithm 4.4: Constricting 3-Opt

Input: Set of areas $A = \{A_1, ..., A_n\}$, initial tour $T' = (p'_1, ..., p'_n)$ (so that $p'_i \in A_i$), accuracies ε, μ
Output: Tour $T = (p_1, ..., p_n)$

```
1  T ← T';
2  Lgth_T ← Length(T);
3  candT ← null;
4  for i = 1; i ≤ n − 2; i++ do
5      for j = i + 1; j ≤ n − 1; j++ do
6          for k = j + 1; k ≤ n; k++ do
7              candT←NewTour1(T, Lgth_T, μ, i, j, k, n);
8              if candT ≠ null then
9                  if Length(candT) < Lgth_T then
10                     T ← mRBA(A, candT, ε, μ);
11                     GoTo(line 2);
12                 end
13             end
14             candT←NewTour2(T, Lgth_T, μ, i, j, k, n);
15             if candT ≠ null then
16                 if Length(candT) < Lgth_T then
17                     T ← mRBA(A, candT, ε, μ);
18                     GoTo(line 2);
19                 end
20             end
21         end
22     end
23 end
24 return T;
```

location of the points in the areas becomes not optimal in regard to the new obtained sequence. Therefore, mRBA is applied afterwards.

4.4.2 Constricting 3-Opt Heuristic

In this section we describe the tour-improvement heuristic Constricting 3-Opt (C3-Opt) for solving TSPN. It is an extension of the classic 3-Opt that is successfully applied in the TSP domain. The main difference of C3-Opt to 3-Opt is that it also involves a TPP solver (in our case RBA), i.e., it is able to calculate where the points should be located inside the areas. In the original 3-Opt these points are fixed.

In general, tour-improvement heuristics are slower than tour-construction methods. The adaptation to TSPN also brings extra expenses of computational time. Therefore, we provide a description of several techniques that allow speeding up calculation time and minimizing risks of getting into local optima fast.

General structure of the C3-Opt is presented in Algorithm 4.4. The input of the C3-Opt is a set of areas A, an initial tour T' and desired accuracies ε, μ. Three main loops are started in lines 4–6. Indices i, j, k are used to select the edges that have to be exchanged. C3-Opt uses the same two ways to reconnect the edges as in 3-Opt. Therefore, the overall logic could be split into two segments denoting each reconnection variant: the first is in lines 7–13 and the second is in lines 14–20. A candidate tour $candT$ is constructed with the function *NewTour1* by applying the first variant of reconnections to the original tour. If the candidate tour was successfully constructed, i.e., $candT$ is not empty (line 8), and it is shorter than the current tour T (line 9), then the tour is optimized by the RBA algorithm. RBA does not change the sequence but only optimizes locations of the points in the areas. The optimized tour becomes the current tour T (line 10). The length of the reduced tour is saved in $Lgth_T$.

After that C3-Opt continues the optimization process with the first set of edges as the original 3-Opt. Otherwise, it can miss some possible improvements occurred after reconnection. Therefore, main loops are restarted in lines 11 and 18. The second variant of reconnection (lines 14–20) has a similar logic. In contrast to the first variant, it applies another tour construction function, *NewTour2*, due to the alternative way of reconnection of the tour (see Figure 4.2). In the next subsection, we will show how the candidate tour is constructed. The algorithm stops when there are no more possible reconnections. The output is the optimized tour T.

Construction of the Candidate Tour

3-Opt compares sums of the costs of the new and deleted edges to check whether the exchange improved the solution or not. It is straightforward, as the costs of all other edges are constant. However, in TSPN, points could change their locations within the areas. Therefore, a new exchange could bring the improvement not only by changing the sequence, but also by optimizing locations of the points inside the areas. A trivial solution could be to apply RBA algorithm on the newly constructed tour. However, in practice it leads fast to the local minimum. We propose another method: after adding a new point T_{last} to the tour, the location for the previously added point T_{last-1} should be recalculated, i.e., function $PointConstrict(T_{last-2}, A_{last-1}, T_{last}, \mu)$ should be applied. This method of tour constricting does not return the optimum point positions within the areas, but only slightly improves the tour. In practice, this improvement prevents the method to get to a local optimum fast.

The candidate tour for the first reconnection case (see Figure 4.2) is constructed by the Algorithm 4.5. It copies the points from the current tour T one by one to the new tour $candT$.

Points are added with the method *AddPointAndConstrict*. It adds point T_h to the end of the tour $candT$ and then constricts the previously added point $candT_{last-1}$ by using $PointConstrict(candT_{last-2}, A_{last-1}, candT_{last}, \mu)$. When all the points are copied, the algorithm returns the tour.

Algorithm 4.5: NewTour1 function

Input: T, $Lgth_T$, μ, i, j, k, n
Output: Tour $candT$

```
1  candT ← null;
2  for h = 1; h ≤ i; h++ do
3  |   candT ← AddPointAndConstrict(candT, T_h, μ);
4  end
5  for h = j + 1; h ≤ k; h++ do
6  |   candT ← AddPointAndConstrict(candT, T_h, μ);
7  end
8  for h = i + 1; h ≤ j; h++ do
9  |   candT ← AddPointAndConstrict(candT, T_h, μ);
10 end
11 if Length(candT) > Lgth_T then
12 |   candT ← null;
13 |   return candT;
14 end
15 for h = k + 1; h ≤ n; h++ do
16 |   candT ← AddPointAndConstrict(candT, T_h, μ);
17 end
18 PointConstrict(candT_{last−1}, A_last, candT_1, μ);
19 return candT;
```

Note that at some point in time the length of the candidate tour could exceed the length of the current tour. In that case it makes no sense to perform further copying-constricting actions and, therefore, the algorithm returns a null tour. This check could be performed multiple times at any stage of creating a candidate tour. However, in case of earlier or multiple appliances, the expenses of the check could exceed the benefit from it. We propose to locate it after the third loop in the lines 11–14 of the Algorithm 4.5.

The algorithm *NewTour2* for the second variant of reconnection is constructed in the same way as *NewTour1*; the only difference is that points from p_{i+1} to p_j should be copied in reverse order.

4.5 Conclusion

In this chapter, solution methods for optimizing a sequence of relaxed effective tasks are proposed. In order to find a solution, this problem is represented as a Traveling Salesman Problem with Neighborhoods (TSPN). The developed solution approaches, Constricting Insertion Heuristic and Constricting 3-Opt, are based on heuristics from the TSP domain. The CIH is capable of constructing high-quality solutions even for large TSPN instances. The obtained solutions can be improved further by using C3-Opt. Although the proposed planning approaches are illustrated by their robotic application,

they are more general, as no domain-specific knowledge is involved, e.g., kinematics or metrics in C-space. Therefore, these general approaches could be applied to any domain that can be modeled as the TSPN, e.g., Unmanned Aerial Vehicle routing or integrated circuit production planning. Another advantage of the proposed approaches is their simplicity in terms of understanding and implementing.

5. Component2: Entry Points Optimization for a Relaxed Effective Task Sequence

Each of a set of brightly painted hollow wooden dolls of varying sizes, designed to fit inside each other.

A definition of "matryoshka" in online Oxford Dictionaries

In this chapter, we consider a situation where a sequence of relaxed effective tasks has already been calculated with any known algorithm or has been strictly defined by an industrial process. In this case, the effective tasks can be relaxed and the robot-related cost can be minimized. The proposed approach does not depend on the production domain and could be combined with any algorithm for constructing an initial task sequence. We propose an approach that improves a given relaxed effective task sequence by optimizing entry points so that it leads to minimal-cost supporting trajectories between them. The problem overview is presented in Figure 5.1.

The remainder of this chapter is organized as follows. The motivation and introduction to the solution concept are presented in Section 5.1. Related problems and state-of-the-art approaches are discussed in Section 5.2. Nested RBA and problem decomposition are presented in Section 5.3. We conclude and summarize in Section 5.4.

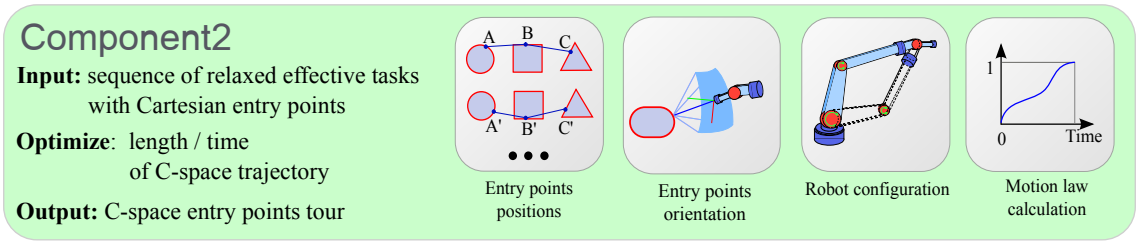

Figure 5.1: Component2 overview.

5.1 Motivation

An industrial robot's workflow typically consists of a set of tasks that have to be repeated multiple times. The efficiency with which the robot performs the sequence of tasks is an important factor in most production domains. In most practical scenarios, the majority of tasks have a certain freedom of execution, i.e., they are relaxed. For example, a closed-contour welding task can often be started and finished at any point of the curve. The sequence of the relaxed effective tasks can be found with the approaches presented in Component1 in Chapter 4. These approaches use Euclidean distance; therefore, the obtained tour is far from being optimal with respect to robot costs, e.g., distance traveled in C-space or time.

In this chapter we propose a method that is able to automatically improve the given sequence of relaxed effective tasks, by optimizing entry point position, orientation and robot configuration. This problem requires a search to be performed in different domains, i.e., find a neighborhood that contains an entry point, find a new position and orientation of the entry point, as well as a robot configuration. The problem can be modeled as the Touring-a-sequence-of-Polygons Problem (TPP) where a tour has to be found through the sequence of polygons. In contrast, we do not have polygons, but rather nested search domains. Therefore, the existing methods cannot be applied directly, but have to be modified.

We propose using the main idea of the RBA algorithm and applying it to optimize the points' locations in nested domains. We refer to such modification of RBA as Nested RBA. In contrast to classic RBA, where optimization is done sequentially for every area, we have several optimization domains instead of a single area, i.e., entry point position, entry point orientation and robot configuration. The choice of the entry point position depends on the results of optimization of corresponding end-effector orientation. The cost of end-effector orientation depends on the optimization results of the inverse kinematics. Therefore, the optimization domains are nested in each other.

5.2 Related Work

A nested domain decomposition splits a problem into several nested domains (subproblems) and then solves each subproblem with domain-specific approaches. Subproblems are interconnected and, therefore, during the solving process, solutions of

subproblems are passed up and down between the different levels of nesting. Such solving strategy excludes the need to apply a metaheuristic for a whole search space and, as it was mentioned, enables us to apply simpler planning methods specialized for the domains.

Nested Domain Decomposition

The nested domain decomposition has already been applied in the field of robotics. Smith et al. [92] used it for a multi-resolution planning for a hexapod robot ATHLETE, developed for moon exploration by NASA. This problem has a large search space and, therefore, the authors suggested to split the problem into nested domains. In the outer domain, more global problems like route planning were solved. The inner domain contained more local problems like moving a robot's leg.

Another application of the nested domain technique in robotics was proposed by Huang et al. [52]. They consider a problem of selecting a manipulator for a system consisting of a robot arm and a position table to perform a given set of tasks. As it is not possible to explore the whole search space in an admissible time, the authors decomposed the problem into three nested domains: 1) manipulator selection (solved by Multiple-Objective Particle Swarm Optimization), 2) layout design (solved by Particle Swarm Optimization) and 3) motion planning (solved by Nearest Neighborhood Algorithm). Less computationally expensive algorithms are chosen for the inner loops (i.e., domains 2 and 3) because the domains are nested in each other and, therefore, the inner loops have to be repeated more often than the outer loops.

Path Planning with End-effector Pose Constraints

Often an end-effector pose (i.e., position and orientation) is strictly defined by a programmer. Only few applications define not a single pose but rather a set of possible poses. In the path planning domain, there exist several approaches to define possible end-effector poses along a path and/or for a goal by using different constraints.

Stilman [95] represented a constrained task as a combination of a task frame, a coordinate system and a motion constrained vector. The last one is a vector of binary values that corresponds to each of the coordinate axis. A value of one means that an end-effector movement should not change the corresponding coordinate. Later Berenson et al. [17] proposed the concept of a Task Space Region (TSR). The idea is to represent an end-effector goal in the T-space as a continuous region — a box that limits possible translations and rotations. TSRs are also used to describe constraints on an end-effector along a path.

Yao et al. [101] addressed a path planning problem with an objective to find a goal configuration and a collision-free path while satisfying pose constraints. A starting robot configuration and a goal pose are given. Constraints for the end-effector are described with system of equations. For example, in oder to constrain the movement of the end-effector to a plane, the equation of a plane is used. End-effector orientation are described with fixed angle representation.

Cefalo et al. [23] proposed a collision-free planner that exploits a freedom of robot kinematics redundancy to obtain cyclic motions under repetitive task constraints, i.e., a robot starts and finishes a task sequence in the same configuration. The proposed algorithm can be reused for calculating a motion cost along closed contour tasks.

From et al. [40] proposed a fast method for minimizing displacements of a paint gun. A freedom is given as a subset of paint gun orientations. Since a real-time behavior was required, a simple T-space cost was used. As a consequence, robot kinematics was not involved on the solving stage but was applied afterwards to an obtained solution.

We propose a way to describe a relaxed task that is inspired by Berenson's TSR [17] and Stilman's approach [95] with several critical changes that allow us to describe curve-like tasks efficiently. In [17], in order to describe a curve, one has to define many TSRs. In our approach, a curve is described by a set of via-volumes that are then interpolated in order to achieve continuity.

5.3 Solution Approach

In this section, we present a solution approach for the problem of entry point optimization for a relaxed effective task sequence. In the following, we introduce a data container which is further used to ease an explanation of the problem decomposition concept and optimization approach.

5.3.1 Entry Point Container

A relaxed effective task provides a freedom in choice of a position and an orientation that a robot's tool can have when starting/following a task. Even though every task has this freedom defined as described in Section 2.1.4, it is assigned with a particular entry point during the solving process. Let us define an entry point structure that uniquely specifies a position and orientation of a tool. This entry point container will be used further in optimization. It is specified as $EP_i = (Task^{Rel}_{EFi}, k, x, y, z, a, b, c, conf_{best})$, where:

- $k \in [0, 1]$ is a parameter that is used to obtain a neighborhood of a task $Task^{Rel}_{EFi}$,
- x, y, z is a position in the k-th neighborhood $Task^{Rel}_{EFi}(k)$,
- a, b, c is an orientation in the k-th neighborhood $Task^{Rel}_{EFi}(k)$,
- $conf_{best}$ is the minimal-cost robot configuration to reach an end-effector pose.

An entry point EP_i allows us to obtain the information about a point that is effected by a tool (i.e., x, y, z) from which a robot's tool has to start-finish processing the task $Task^{Rel}_{EF}$. Furthermore, EP_i specifies an end-effector orientation (i.e., a, b, c) and a robot configuration (i.e., $conf_{best}$). Entry point EP_i of the task $Task^{Rel}_{EF}$ is visualized in Figure 5.2.A.

Note that an entry point and a robot end-effector pose are defined in different coordinate systems. An entry point is defined in X_w, Y_w, Z_w coordinate system, i.e., in relation to a task. However, a robot end-effector pose, i.e., an end of arm point, for this entry point has to be defined in global coordinates. Further, we provide details on calculation of an end-effector pose $Goal_i$ in global coordinates X, Y, Z for a particular entry point EP_i. The position of the $Goal_i$ is defined as point (x_{ef}, y_{ef}, z_{ef}) and orientation X_{ef}, Y_{ef}, Z_{ef}, see Figure 5.2.B. Angles a and b are used to compute an approaching vector of the end-effector, i.e., vector Z_{ef}. Angle c denotes a desired orientation along the tool axis and allows us to calculate normal and sliding vectors of the end-effector, i.e., vectors X_{ef}, Y_{ef}.

The first step to get the end-effector $Goal_i$ point is to calculate the corresponding neighborhood $Task^{Rel}_{EFi}(k)$ for a certain k. Then calculate a rotation matrix for the angles a, b, c of entry point EP_i:

$$M_{rot} = RotZ(b) \cdot RotY(a) \cdot RotZ(c),$$

where $a, b, c \in EP_i$. $RotY$ and $RotZ$ are the rotation matrices about the corresponding axis in a three-dimensional space.

Next, calculate a robot end-effector position $EF = (x_{ef}, y_{ef}, z_{ef})^T$:

$$\begin{aligned} x_{ef} &= x + R \cdot sin(a) \cdot cos(b) \\ y_{ef} &= y + R \cdot sin(a) \cdot sin(b) \\ z_{ef} &= z + R \cdot cos(a), \end{aligned}$$

where R is the length of a robot's tool. For some tasks, it can be constant, e.g., cutting or drilling, see Chapter 5, or it can vary, e.g., taking a picture, see Chapter 6.

Finally, calculate a T-space point $Goal_i$ for a robot to reach:

$$Goal_i = \begin{bmatrix} M_w & 0 \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \cdot \begin{bmatrix} M_{rot} & EF \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \cdot \begin{bmatrix} RotX(180°) & 0 \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}, \tag{5.1}$$

where M_w defines a coordinate system in the neighborhood $Task^{Rel}_{EFi}(k)$.

$RotX$ is a rotation matrix about X axis in three-dimensional space. Rotation on 180° is needed to direct the Z_{ef} from the end of arm to the task entry point. A function that returns the end-effector pose $Goal_i$ for a particular entry point EP_i is denoted as: $GetGoal(EP_i) \rightarrow Goal_i$.

The described concept of entry point has redundant information, as it is possible to save only a robot configuration $conf_{best}$. End-effector position (x, y, z) and orientation (a, b, c) can be obtained with forward kinematics.

The previously described transformations show how to calculate a robot end-effector goal for a particular entry point container.

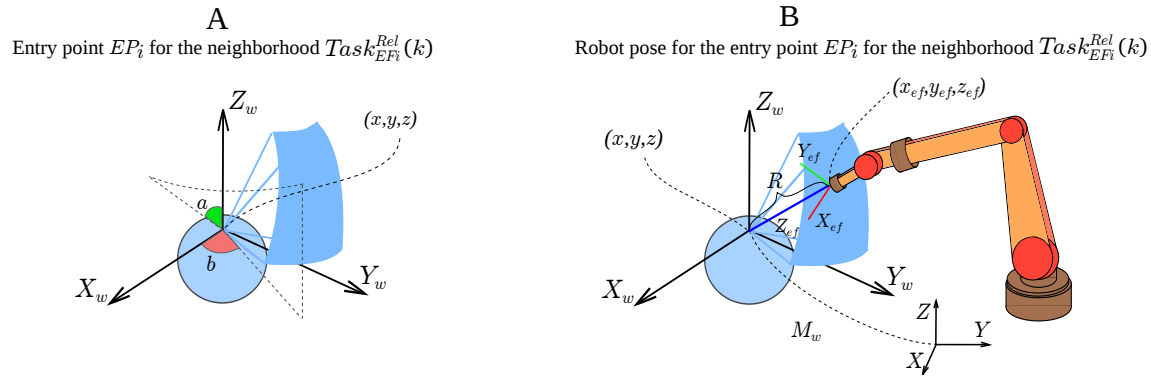

Figure 5.2: Robot pose is visualized for the entry point $EP_i = (Task^{Rel}_{EFi}(k), k, x, y, z, a, b, c, conf_{best})$ of the task neighborhood $Task^{Rel}_{EFi}(k)$. Angle c is omitted for simplicity.

5.3.2 Problem Decomposition

The problem of optimizing entry points for the given relaxed effective task sequence is computationally difficult to solve, as it leads to a large and multidimensional search space. We propose to apply a hierarchical optimization approach. It performs a fast local search on every nested stage instead of applying time-consuming global optimization techniques to the whole problem.

We split the optimization process into the following nested stages:

Stage 1: Optimize parameter $k \in [0, 1]$ for a relaxed effective task $Task^{Rel}_{EFi}$ such that a neighborhood $Task^{Rel}_{EFi}(k)$ contains an entry point with near-optimal position and orientation.

Stage 2: Optimize a position (x, y, z) and an orientation (a, b, c) of an entry point EP_i in the task neighborhood $Task^{Rel}_{EFi}(k)$ obtained in Stage 1.

Stage 3: Optimize a robot configuration $conf_{best}$ for the position (x, y, z) and orientation (a, b, c) (obtained in Stage 2) that belongs to the neighborhood $Task^{Rel}_{EFi}(k)$ obtained in Stage 1.

Each outer stage depends on results of optimization on the inner stage, see Figure 5.3. In other words, in order to optimize the entry point of an area with respect to preceding and succeeding areas, one has to find a neighborhood (parameter k) that contains a near-optimal entry point. The cost of a certain neighborhood of the relaxed effective task depends on the chosen position (x, y, z) and orientation (a, b, c) of an end-effector. The cost of a position and an orientation depends on an inverse kinematics solution for this entry point. The cost of a kinematics solution in a particular entry point depends on possible kinematics solutions in entry points of preceding and succeeding tasks.

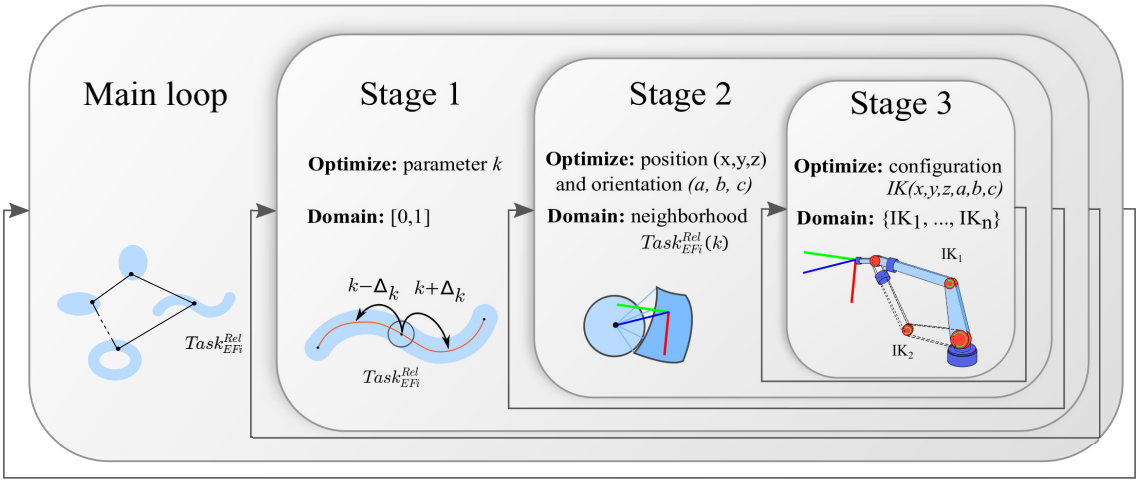

Figure 5.3: Three nested stages for optimizing a entry points for the sequence of relaxed effective task. The figure depicts a particular case where the approach calculates the cost for a neighborhood $Task^{Rel}_{EFi}(k + \Delta_k)$, a position (x, y, z) and orientation Euler angles a, b, $c + \Delta_c$ in this position.

During the optimization process, an outer stage executes an inner stage multiple times. For example, while searching for the near-optimal position and orientation of the end-effector (Stage 2) in a certain neighborhood, the costs of multiple position and orientation tuples have to be compared. For each of them, inverse kinematics optimization (Stage 3) has to be performed.

We applied the Pattern search [51] for optimization on the Stages 1 and 2. The Dijkstra algorithm or simple exhaustive search can be applied for the Stage 3.

5.3.3 Optimization Approach

The underlaying idea of our approach is based on the Rubber-band algorithm (RBA) [78]. The RBA iterates over all areas and improves entry points, one at a time. We propose the modification of RBA that instead of searching in one search domain, does the search in several nested domains. We refer to this approach as Nested RBA. Our modification of the RBA that takes a sequence of entry point EPS and outputs an optimized EPS. As input our approach takes a sequence of relaxed effective tasks $Job = (Task^{Rel}_{EF1}, ..., Task^{Rel}_{EFn})$ defined as described in Chapter 2. Its output is an entry point sequence $EPS = (EP_1, ..., EP_n)$. A sequence EPS is initialized by selecting an entry point $EP_i = (Task^{Rel}_{EFi}, k, x, y, z, a, b, c, conf_{best})$ for every task $Task^{Rel}_{EFi}$. Parameters of EP_i can be chosen randomly so that x, y, z, a, b, c are within a neighborhood $Task^{Rel}_{EF}(k)$. Another way is to apply the algorithms from Chapter 4.

Afterwards, one has to calculate the best robot configuration $conf_{best}$ for every EP_i. It can be done by obtaining a list of possible robot configurations for each entry point and constructing a graph, where every configuration from entry point EP_i is connected with every configuration from entry point EP_{i+1}. The weights in the graph

can be time or C-space distances. Then a graph search algorithm obtains the shortest path that visits only one configuration in every entry point. For this purpose, the Djikstra algorithm is used. Although, this approach is capable of fixing configurations in several entry points at once, it is time consuming. Since the Stage 3 is executed more often than other stages, having there a slow algorithm dramatically increases the computational time. Therefore, the Djikstra algorithm is applied only once to obtain the cost of an initial solution.

Note that an entry point sequence EPS is transferred between all stages and represents a container that accumulates all information about the end-effector poses and robot configurations for a particular task sequence.

The overall workflow of the Nested RBA is shown in Algorithm 5.1. We iterate over all entry points and improve the sequence EPS by optimizing the i-th entry point in every iteration with a function *OptimizeNeighborhood* (lines 2–4, Algorithm 5.1). This function receives a sequence EPS and an index i as an input and returns an improved EPS and its cost. This approach is described in Section 5.3.2 (Stage 1). The whole entry point sequence instead of a single point EP_i is passed to this function because information about other entry points is also required to calculate costs at the Stages 1 and 3. A stopping condition for the man loop can be an elapsed computational time, a desired precision of a solution or a maximum number of iterations.

Algorithm 5.1: Main loop of the Nested RBA

Input: Initial T-space tour $T = (t_1, ..., t_n)$
Output: C-space tour CT

```
1 while stopping condition is not satisfied do
2 |   for i ← 1 to n do
3 |   |   EPS, cost ← OptimizeNeighborhood(EPS, i);
4 |   end
5 end
6 return EPS;
```

In the following, the realization of the Nested RBA stages is described in detail.

5.3.3.1 Stage 1: Optimization of an Neighborhood

In this section, the *OptimizeNeighborhood* algorithm is described. Its goal is to choose the best value of the parameter k for an entry point EP_i. Its workflow is presented in Algorithm 5.2.

At first, Stage 1 algorithm calculates a current cost $cost_{cur}$, initializes a step Δ_k as a value from $(0, 1]$ and saves the initial EP_i (lines 1–3, Algorithm 5.2). *GetSequenceCost* is a user-defined cost function that represents the optimization criterion. It returns the cost of the supporting movement in regard to a desired metric, e.g., time or traveled distance in C-space. If the $conf_{best}$ of an entry point is not feasible for a robot to reach

and, therefore, has a value *null*, then the function *GetSequenceCost* returns ∞. It prevents from obtaining invalid tours with tasks that are not reachable by the robot. The algorithm calculates the cost for the values: $k + \Delta_k$ and $k - \Delta_k$, see Stage 1 in Figure 5.3 and lines 9 and 16 in Algorithm 5.2.

At each step, we call the *OptimizePose* algorithm (described in Stage 2 in Section 5.3.2) to obtain the costs $cost_{k-\Delta}$ and $cost_{k+\Delta}$ as well as new entry point sequences $EPS_{k-\Delta}$ and $EPS_{k+\Delta}$ (lines 10 and 17, Algorithm 5.2). Note that *OptimizePose* is called with different $EPSs$, where the parameter k of EP_i has different values: $k + \Delta_k$ and $k - \Delta_k$. If one of these steps brings an improvement, then stop the algorithm and return the improved entry point sequence and the new cost (lines 21 and 23, Algorithm 5.2). If these actions did not improve the cost then decrease the step Δ_k (line 26, Algorithm 5.2) and repeat the process.

The algorithm returns an entry point sequence when the first improvement was found. However, an improvement might not be found even if the step Δ_k was decreased many times. This happens when local search strategies on the inner Stages 2 and 3 come to a local minimum and cannot improve the current cost any more. One way to determine this is to compare $cost_{k-\Delta}$ and $cost_{k+\Delta}$ values between two iterations. When there was no improvement, then a rollback for EP_i should be made (line 29, Algorithm 5.2). This approach guarantees that during optimization the overall sequence cost will decrease or stay the same.

5.3.3.2 Stage 2: Optimization of an End-effector Pose

This section covers details of the *OptimizePose* algorithm. This algorithm optimizes position (x, y, z) and orientation angles (a, b, c) of an entry point EP_i of a value of the parameter k defined by the outer Stage 1. An output of the algorithm is an optimized entry point sequence EPS and a new cost. In contrast to the Stage 1, where optimization is done until the first improvement is found, the optimization of orientation is performed until a stopping condition is met. A stopping condition can be reaching a desired solution precision.

At first, *OptimizePose* initializes its main variables in lines 1–3 in Algorithm 5.3. Initial values should belong to the corresponding neighborhood. For example, initial values of orientation angles a, b, c can be computed as the middle angles of an orientation window (see Section 5.3.1), e.g., $a = a_l + (a_u - a_l)/2$. An initial cost is calculated with the algorithm $OptimizeConfiguration$ described in Section 5.3.2. Finally, steps Δ_x, Δ_y, Δ_z, Δ_a, Δ_b, Δ_c are initialized. The size of a step depends on the allowed interval of each angle of a current node point. For example, in our implementation, initially Δ_a is a half of the interval $[a_l, a_u]$. The steps Δ_b and Δ_c are obtained analogously.

The main idea of Stage 2 is to iteratively improve the position values and orientation angles. Within one iteration (lines 4–18, Algorithm 5.3) six angle combinations are checked, i.e., $(a - \Delta_a, b, c)$, $(a + \Delta_a, b, c)$, $(a, b - \Delta_b, c)$, $(a, b + \Delta_b, c)$, $(a, b, c - \Delta_c)$, $(a, b, c + \Delta_c)$. Position values are checked analogically. In pseudocode, we show only the case $(a - \Delta_a, b, c)$, as the steps for other combinations are similar.

Algorithm 5.2: *OptimizeNeighborhood*

Input: Entry point sequence EPS, current index i
Output: Improved entry point sequence EPS, cost of the improved EPS

```
1  cost_cur ← GetSequenceCost(EPS);
2  Δ_k ← InitializeStep;
3  EP_initial ← EP_i;
4  while true do
5      k_temp ← k;
6      if k − Δ_k < 0 then
7          cost_{k−Δ} ← ∞;
8      else
9          k ← k − Δ_k, where k ∈ EP_i, EP_i ∈ EPS;
10         EPS_{k−Δ}, cost_{k−Δ} ← OptimizePose(EPS, i);
11     end
12     k ← k_temp;
13     if k + Δ_k > 1 then
14         cost_{k−Δ} ← ∞;
15     else
16         k ← k + Δ_k, where k ∈ EP_i, EP_i ∈ EPS;
17         EPS_{k+Δ}, cost_{k+Δ} ← OptimizePose(EPS, i);
18     end
19     if (cost_{k−Δ} < cost_cur) or (cost_{k+Δ} < cost_cur) then
20         if cost_{k−Δ} < cost_{k+Δ} then
21             return EPS_{k−Δ}, cost_{k−Δ};
22         else
23             return EPS_{k+Δ}, cost_{k+Δ};
24         end
25     else
26         Δ_k ← Δ_k / 2;
27     end
28     if no changes in cost_{k−Δ}, cost_{k+Δ} between iterations then
29         EP_i ← EP_initial;
30         return EPS, cost_cur;
31     end
32 end
```

The angle a takes a value $a - \Delta_a$, whereas the other orientation angles b and c keep their initial values. If the value $a - \Delta_a$ is not within the range $[a_l, a_u]$, then the step Δ_a is decreased (lines 14, Algorithm 5.3). Otherwise, an improved $EPS_{a-\Delta}$ and its $cost_{a-\Delta}$ are obtained by optimizing a robot configuration with a function

Algorithm 5.3: *OptimizePose*

Input: Entry point sequence EPS, current index i
Output: Improved entry point sequence EPS, cost of the improved EPS

1 $x, y, z, a, b, c \leftarrow InitializeStartingValues$;
2 $EPS_{best}, cost_{cur} \leftarrow OptimizeConfiguration(EPS, i)$;
3 Δ_x,Δ_y,Δ_z,Δ_a,Δ_b,$\Delta_c \leftarrow InitializeSteps$;
4 **while** *stopping condition is not satisfied* **do**
5 $a_{temp} \leftarrow a$;
6 $a \leftarrow a - \Delta_a$, where $a \in EP_i$, $EP_i \in EPS$;
7 **if** *(*$a_l < a$*) and (*$a < a_u$*)* **then**
8 $EPS_{a-\Delta}$, $cost_{a-\Delta} \leftarrow OptimizeConfiguration(EPS, i)$;
9 **if** $cost_{a-\Delta} < cost_{curr}$ **then**
10 $EPS_{best} \leftarrow EPS_{a-\Delta}$;
11 $cost_{curr} \leftarrow cost_{a-\Delta}$;
12 **end**
13 **else**
14 $\Delta_a \leftarrow \Delta_a$ / 2;
15 **end**
16 $a \leftarrow a_{temp}$;
 ⋮
17 Check $(x + \Delta_x, y, z)$, $(x - \Delta_x, y, z)$ $(x, y - \Delta_y, z)$, $(x, y + \Delta_y, z)$, $(x, y, z - \Delta_z)$, $(x, y, z + \Delta_z)$, $(a + \Delta_a, b, c)$, $(a, b - \Delta_b, c)$, $(a, b + \Delta_b, c)$, $(a, b, c - \Delta_c)$, $(a, b, c + \Delta_c)$ by analogy;
 ⋮
18 **if** *no cost improvement* **then**
19 Δ_x,Δ_y,Δ_z,Δ_a,Δ_b,$\Delta_c \leftarrow ReduceSteps$;
20 **end**
21 **end**
22 **return** EPS_{best},$cost_{cur}$;

OptimizeConfiguration that is described in Section 5.3.3.3. If a newly obtained cost is less than the initial cost, then save the new sequence and its cost (lines 9–11, Algorithm 5.3). Then the angle a takes the initial value and the other five combinations are checked. If none of the six combinations lead to an improvement, the steps Δ_a, Δ_d, Δ_c, are decreased (lines 19, Algorithm 5.3).

5.3.3.3 Stage 3: Optimization of a Robot Configuration

The goal of the Stage 3 is to choose a near-optimal robot configuration to reach a position (x, y, z) and orientation angles a, b, c (given by the Stage 2) that belongs to a neighborhood $Task_{EF}^{Rel}(k)$ (given by the Stage 1). Choosing the best configuration is

Algorithm 5.4: *OptimizeConfiguration*

Input: Entry point sequence EPS, current index i
Output: Improved entry point sequence EPS, cost of the improved EPS

```
1  Goal_i ← GetGoal(EP_i);
2  IKS ← GetIK(Goal_i);
3  if IKS = ∅ then
4  |  return EPS, ∞;
5  end
6  costIK_min ← ∞;
7  foreach IK_j ∈ IKS do
8  |  costIK_temp ← GetCost(IK_j, conf_best ∈ EP_{i−1}) +
   |  GetCost(IK_j, conf_best ∈ EP_{i+1});
9  |  if costIK_temp < costIK_min then
10 |  |  costIK_min ← costIK_temp;
11 |  |  conf_best ← IK_j, where conf_best ∈ EP_i, EP_i ∈ EPS;
12 |  end
13 end
14 cost ← GetSequenceCost(EPS);
15 return EPS, cost;
```

nontrivial. Any change of a single entry point configuration affects the global cost of the whole entry point sequence. Therefore, ideally configurations of all entry points should be optimized at once, for example, with a Dijkstra algorithm.

Since the Stage 3 is executed more often than the other stages, having there a global algorithm dramatically increases computational time. Therefore, a local strategy that optimizes only $conf_{best}$ in the current entry point EP_i is preferred.

The approach of the Stage 3 is implemented as a function $OptimizeConfiguration$. It takes an entry point sequence EPS and a current index i as an input and returns an improved sequence EPS with a robot configuration $conf_{best}$, optimized only for EP_i, and its cost. The main steps of the approach are presented in Algorithm 5.4. At first, a function *GetGoal* calculates a T-space point $Goal_i$ that is affected by an end-effector in an entry point EP_i according to Equation 5.1. A function $GetIK$ solves an inverse kinematics problem and obtains a set of possible robot configurations IKS for a robot to reach the point $Goal_i$. If no inverse kinematics solution was found, i.e., the $Goal_i$ is not reachable by the robot, then the algorithm returns the initial EPS and the infinite cost. The core goal of the algorithm is to choose such an IK solution from the IKS that minimizes the cost of a trajectories between the point EP_i and its neighboring entry points EP_{i-1} and EP_{i+1}. It is realized in lines 6–10 where a function *GetCost* returns the cost between two robot configurations. Finally, when the best configuration is chosen, the sequence cost is recalculated (line 11, Algorithm 5.4). Then an improved

sequence EPS with an optimized robot configuration $conf_{best}$ and its cost $cost$ are returned to the Stage 2.

5.4 Conclusion

This chapter considers the problem of improving a given relaxed effective task sequence with respect to a robot-oriented cost. The goal is to optimize the entry point of each relaxed effective task so that the cost of a robot supporting movement is minimized. Entry points here denote the starting position, orientation and a robot configuration for every task. We modeled this problem as the Touring-a-sequence-of-Polygons Problem, where a tour has to be found through the sequence of polygons. In contrast to the classic TPP, here we have to search in several nested domains, i.e., find a neighborhood that contains the entry point, find the position and orientation of the entry point and finally find the robot configuration. The main idea of the proposed solving method is to decompose the problem into three nested stages and solve each of them with local search techniques. Similar conceptions of problem decomposition were previously successfully applied to the planning of a hexapod robot for NASA moon exploration [92] and for industrial robot task sequencing [52]. We modify the Rubberband algorithm to be able to search in nested domains. We refer to the proposed domain independent approach as Nested RBA.

6. Component3: Robot Trajectory Optimization for a Relaxed Effective Task

I will not follow where the path may lead, but I will go where there is no path, and I will leave a trail.

"Wind-Wafted Wild Flowers"
by Muriel Strode

After finding a sequence of Cartesian entry points using the methods presented in Component1 and improving entry points using the method in Component2 with respect to robot-related cost function, we obtain a supporting trajectory and starting robot configurations for every relaxed effective task. At this point, the optimization of supporting movements is finished. The next step is to optimize the robot trajectory for every relaxed effective task. In this chapter, we present an RBA-based approach that constricts the neighborhoods of the relaxed effective task with a smooth curve, i.e., an end-effector path for a robot. The objective is to minimize robot trajectory cost.

The remainder of this chapter is organized as follows. The introduction and motivation for the problem are presented in Section 6.1. Related problems are discussed in Section 6.2. Our solution approach is presented in Section 6.3. The method to calculate the trajectory from the given motion law is described in Section 6.3.2.

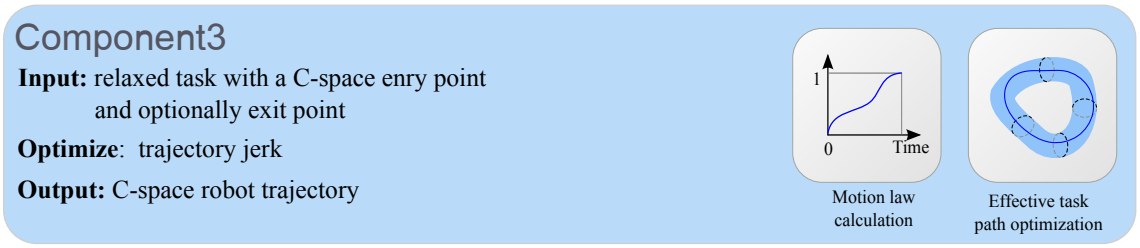

Figure 6.1: Component3 overview.

6.1 Motivation

The way to define a robot's path for an effective task depends on the application domain and task geometry. Therefore, a robot's end-effector path and its motion law are often defined in a strict way to meet the requirements of an industrial process. For example, a robot has to weld a line and to maintain a certain velocity in order to have a desired effect on a surface. As a consequence, the obtained robot trajectory is dictated by the application scenario; however, the robot's execution can be "awkward", e.g., it can cause high jerks in robot joints.

At the same time, effective tasks often allow some freedom of execution. For example, laser-welding can be performed with one of several possible tool orientations [59] and cutting can be performed with a set of possible tool positions and orientations [4].

We present an approach to find such an end-effector path for a relaxed effective task with a given C-space entry point (see Figure 6.1) that leads to a minimal cost robot trajectory. A motion law is assumed to be given, e.g., it was imposed by an industrial application or has already been computed and optimized. In order to solve this problem, it is modeled as the TPP, see Figure 6.2. This is done by discretizing a given relaxed effective task into a sequence of neighborhoods which have to be visited. We propose a modification of the Rubber-band Algorithm to solve the TPP. The proposed heuristic approach does not depend on a cost function and a way to define the relaxation of an effective task.

In this chapter, we are focusing on continuous trajectories instead of point-to-point trajectories, i.e., predefined motion law must be maintained throughout the whole path and not only in its via-points. A trajectory cost is domain-dependent and can be, for example, jerk, energy or material influence metric.

6.2 Related Work

There are several methods that also exploit a freedom of executing an effective task in order to optimize a corresponding trajectory. In this section, we discuss them in detail.

Generation of a C-space trajectory is computationally expensive when an end-effector path consists of a large number of via-points. Aspragathos [13] described a

technique to relax an end-effector path by using position deviation. He considered an end-effector path optimization problem which utilizes an execution freedom to minimize the number of via-points and, as a consequence, the number of IK calls. In contrast, our problem formulation does not aim at minimizing a number of via-points but rather at minimizing a cost of robot's C-space trajectory by letting the via-points float in their neighborhoods.

Another similar problem was proposed by Kolter et al. [58] who used cubic splines to construct T-space smooth trajectories. All constraints are convex and are applied to the via-points of a T-space path. The problem was solved with a general purpose convex solver. The presented approach is powerful and can incorporate numerous objective functions from the T-space, except minimization of the trajectory duration time. Cost functions from the C-space can also be used but in this case Jacobian approximation should be performed along the trajectory. As a consequence, only one of many IK solutions is considered. The main limitation is that this technique is not suitable for the scenarios where a path must go through non-convex narrow corridors in a robot C-space. Such scenarios often appear in industrial robotics when effective tasks are involved.

From et al. [40] described a freedom for the orientation of a paint gun by using convex constraints. The proposed a real-time approach which calculates an optimal paint gun orientation for each time step and a given constant velocity value. The minimal cost here means that the displacements of a paint gun are minimized. The cost function must be convex. The problem considered in this chapter is a generalization of their problem, as the freedom is provided both for the end-effector orientation and position. We do not require constraints or an objective function to be convex. There is also no requirement that a velocity has to be a constant value, it can be an arbitrary function.

6.3 Solution Approach

Exhaustive search strategies are impractical due to the large search space of the robot trajectory optimization for the relaxed effective task. Convex solvers cannot be applied, as we do not restrict problem constraints and a cost function to be convex. A way to solve the problem is to apply a heuristic approach. Heuristics do not guarantee finding the optimum, however, they can provide near-optimal solution to real-life scenarios in a reasonable time. We propose a heuristic search that is based on the RBA [78] and on the Pattern Search (PS) [51]. We refer to this approach as Smoothed RBA. In contrast to standard RBA, proposed modification of RBA constricts the areas not with a polyline, but with a smooth curve. In addition, the cost is calculated for the C-space trajectory but not for the spline in T-space. Therefore, during optimization, moving one via-point of an end-effector trajectory requires not only to measure distance to the neighboring points, but to recalculate full C-space trajectory instead. Further a detailed explanation of the algorithm is given.

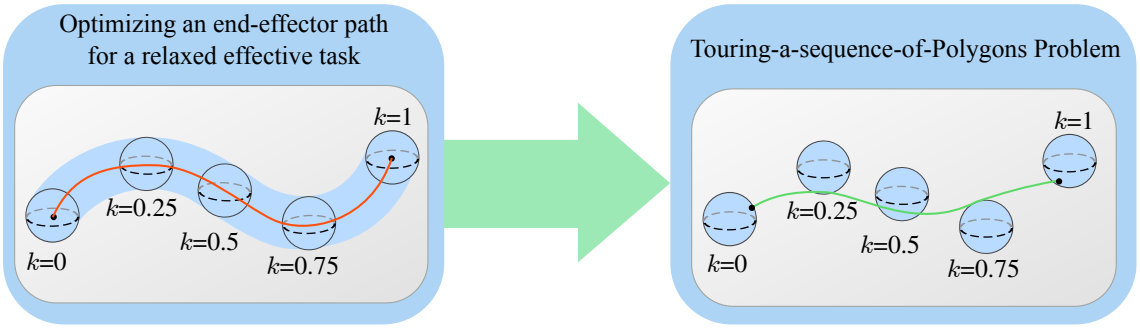

Figure 6.2: A problem of optimizing an end-effector trajectory is represented as the Touring-a-sequence-of-Polygons Problem by using discretization.

6.3.1 Smoothed RBA

First, we introduce a notion of a discretized relaxed effective task. A relaxed effective task $Task_{EF}^{Rel}$ is a continuous function. Therefore, for optimization, the task $Task_{EF}^{Rel}$ is discretized, i.e., only the key via-neighborhoods should be chosen. It can be done by discretizing the domain of an effective task. For example, the domain parameter k can take values from the discrete set $\{0, 0.1, ..., 0.9, 1\}$ instead of the continuous interval $[0, 1]$. We denote to a discretized effective task as $Task_{EF}^{RelD}$.

Our algorithm requires the following parameters as an input: a discretized relaxed effective task $Task_{EF}^{RelD}$, an end-effector motion law ML_{EF}, a motion duration T and a $Frequency$ of the output trajectory. Note that motion duration time T depends on the industrial domain and particular task. If it is too small the algorithm will not find a solution because the limits of a robot joint velocity will be violated. The Smoothed RBA output is an optimized continuous C-space trajectory $Traj_R$. The main steps of the optimization process are presented in Algorithm 6.1.

Initially, the algorithm gets a feasible discrete path $Path_{EF}^{D}$, so that it belongs to the given relaxed effective task $Task_{EF}^{RelD}$. Then a C-space robot trajectory is calculated with the algorithm $GetTrajectory$ which is discussed further in Section 6.3.2. A robot trajectory cost is calculated by the function $GetCost$ according to Equation 7.1.

Smoothed RBA is an iterative approach. In each iteration, it runs through all the points from the $Path_{EF}^{D}$ and optimizes the position and the orientation of each point one by one. The iterative process terminates when a stopping condition, e.g., a number of iterations or elapsed calculation time, is satisfied.

Optimization for a single point can be done in a number of ways. In the current implementation, PS is applied (lines 6–15 in Algorithm 6.1). At first, PS modifies the T-space point P_i (line 7) by changing one of the point's coordinates by a certain small value. The PS loop terminates when no further modification is possible (line 6), i.e., all coordinates have already been modified and improvement was found. For every new modification, a new path $Path_{EFnew}$ is obtained and a trajectory is recalculated with the further described method $GetTrajectory$ (line 8) and its cost is obtained (line

Algorithm 6.1: Smoothed RBA

Input: $Task_{EF}^{RelD}$, ML_{EF}, T, $Frequency$
Output: $Traj_R$

```
1  Get a feasible initial path Path_EF^D ∈ Task_EF^RelD;
2  Traj_R ← GetTrajectory(Path_EF^D, ML_EF, T, Frequency);
3  cost ← GetCost(Traj_R) ;
4  while stopping condition is not satisfied do
5      foreach P_i ∈ Path_EF^D do
6          while Modifications are possible do
7              Path_EFnew^D ← Modify(Path_EF^D, P_i);
8              Traj_Rnew ← GetTrajectory(Path_EFnew^D, ML_EF, T);
9              cost_new ← GetCost(Traj_Rnew) ;
10             if cost_new < cost then
11                 Path_EF^D ← Path_EFnew^D;
12                 cost ← cost_new;
13                 Traj_R ← Traj_Rnew;
14             end
15         end
16     end
17 end
18 return Traj_R;
```

9). If the modification leads to the cost decrease (line 10), then save the $Path_{EFnew}$, $cost_{new}$, $Traj_{Rnew}$ (lines 11–13). The algorithm guarantees that the path worse than the initial one will not be returned. Note, that the algorithm only varies the path of the end-effector but keeps the motion law unchanged.

6.3.2 C-space Trajectory Calculation

In this section the way to calculate C-space trajectory is discussed. The C-space trajectory $Traj_R$ is calculated with the method $GetTrajectory$. The straightforward way to obtain a C-space trajectory is to map every point of the end-effector trajectory to the robot configuration by calculating inverse kinematics. However, it requires a large number of inverse kinematics calls that are normally computationally expensive.

At first, a discrete robotic path $Path_R^D$ is obtained by calculating inverse kinematics for a discrete end-effector path $Path_{EF}^D$ via-points, see line 1 in Algorithm 6.2. Note, that only one of several possible solutions of inverse kinematics is chosen. For example, "elbow-up" configuration for all the points from the $Path_{EF}^D$. Then $Path_R^D$ is interpolated into a continuous smooth robot path $Path_R$ with a smooth function with a $Parameter_{init}$, an array consisting of values from the interval $[0, 1]$ starting with 0 and increasing by a small step (line 3 in Algorithm 6.2).

In this work, cubic splines were applied for interpolation of the path and motion law, as they are twice continuous differentiable and provide constant jerk. Higher order

Algorithm 6.2: *GetTrajectoryectory*—C-space trajectory calculation

Input: $Path^{D}_{EF}$, ML_{EF}, T, $Frequency$
Output: $Traj_R$

1 $Path^{D}_{R} \leftarrow IK(Path^{D}_{EF})$;
2 $Parameter_{init} \leftarrow GenerateParameterfrom[0,1]$;
3 $Path^{temp}_{R} \leftarrow Interpolate(Path^{D}_{R}, Parameter_{init})$;
4 **for** $i \leftarrow 0$ **to** 1 ***with step*** $1/(Frequency \times T)$ **do**
5 **if** $i = 0$ **then**
6 $Conf_1 \leftarrow Path^{temp}_{R}(i)$;
7 $Pose_1 \leftarrow FK(Conf_1)$;
8 $Configurations.Append(Conf_1)$;
9 $Parameter.Append(0)$;
10 **else**
11 $Conf_2 \leftarrow Path^{temp}_{R}(i)$;
12 $Pose_2 \leftarrow FK(Conf_2)$;
13 $Configurations.Append(Conf_2)$;
14 $Parameter.Append(d(Pose_1, Pose_2))$;
15 $Conf_1 \leftarrow Conf_2$;
16 **end**
17 **end**
18 $Parameter \leftarrow Normalize(Parameter)$;
19 $Path_R \leftarrow Interpolate(Configurations, Parameter)$;
20 $Traj_R \leftarrow Path_R(ML_{EF})$;
21 **return** $Traj_R$;

splines generally suffer from unwanted high osculation and might lead to a retrograde motion [67]. This allows to obtain smooth $Path^{temp}_{R}$.

A loop in lines 4–17 of Algorithm 6.2 calculates a new interpolation parameter *Parameter* in order to scale the $Path_R$ for computing a robot trajectory. The loop iterates through the spline domain $[0, 1]$ with a step size 1/(Frequency * T). In the first iteration, it initiates variables for the last robot configuration $Conf_1$, the last robot's pose $Pose_1$, an array of considered configurations *Configurations* and a new parameter *Parameter*. The robot's pose is obtained by applying forward kinematics to the configuration. An array *Parameter* consists of the distances between the consequent positions of the end-effector. The sum of all values from *Parameter* equals the length of the T-space path. In the remaining iterations, the algorithm computes the current configuration $Conf_2$, the corresponding pose $Pose_2$ and extends the arrays *Configurations* and *Parameter* with new values.

The final steps are as following. After obtaining the new Parameter array, its elements are normalized so that their sum equals 1, i.e., each element is divided by the sum of all elements. Then the algorithm constructs a new continuous robot path by using interpolation with this normalized Parameter and the sequence of robot's Config-

urations. This allows us to obtain a smooth robot path which leads to a monotonous motion of the end-effector. Finally, a robot trajectory $Traj_R$ is calculated by applying the motion law ML_{EF} to the path $Path_R$.

The described algorithm reduces the number of IK calls. In case if more control on precision is desired, the number of via-points (size of an array Parameter) can be increased.

6.4 Conclusion

In this chapter, we observed the problem of optimizing robot trajectory for a given relaxed effective task. The motion law can be either dictated by a certain industrial application, e.g., maintain certain velocity while welding, or it can be optimized with known approaches. To solve the problem, we represent a continuous relaxed effective task as a sequence of neighborhoods. In this way, the problem can be represented as a Touring-a-sequence-of-Polygons Problem. Despite this discretization, we are still focusing on continuous trajectory, and during optimization we make sure that a predefined motion law is maintained through the whole continuous end-effector path. We propose the algorithm Smoothed RBA which is based on an RBA, but instead of using a polyline to constrict areas, it uses a smooth curve for an end-effector path. The goal of optimization is to reduce the C-space trajectory cost. The described solution method is application-independent that does not impose any requirements on types of constraints and a cost function, e.g., they can be non-convex.

7. Evaluation

> Climate is what you expect,
> weather is what you get.
>
> Robert A. Heinlein

In this thesis, we presented approaches for (i) calculating a task sequence and the corresponding entry points of the tasks, (ii) improving a task sequence with regard to a robot cost metric, and (iii) optimizing a robot trajectory for a relaxed effective task. In this chapter, an evaluation of the proposed approaches is presented.

7.1 Evaluation of the Component1 Approaches

This section presents an evaluation of the proposed task sequencing algorithms on three sets of instances. We compare Constricting Insertion Heuristic (CIH), Constricting 3-Opt (C3-Opt) and their variants with state-of-the-art approaches on three set of instances: (i) 24 instances with known optimum, (ii) 18 instances with "stretched" ellipses and (iii) 20 instances for the Close-Enough TSP (CETSP).

All the methods from Component1 proposed in this thesis were run on the following hardware: Intel Core 2 Quad CPU, 2.83 GHz with 8 GB of RAM, running Microsoft Windows Vista. A state-of-the art approach and a solver for optimal values were ran by Gentilini et al. [43] using Intel Xeon, 3.33 GHz CPU with 12 GB of RAM, running Fedora. Since the provided computational time was obtained in different conditions, it cannot be compared directly.

7.1.1 Evaluated Algorithms

CIH and C3-Opt variants

A problem of task sequencing and entry point selection is modeled as a TSPN. We split this problem into a TSP and a TPP that are then solved simultaneously. In the following we present evaluation results for CIH and C3-Opt heuristics.

The CIH is a tour-construction heuristic which iteratively builds a solution that can be improved further. We apply the 2-Opt and the 3-Opt algorithms as tour-improvement methods for the TSP and mRBA for the TPP. We denote a combination of 2-Opt and mRBA by 2-*Impr.* and a combination of 3-Opt and mRBA by 3-*Impr.*, respectively. Starting points for the listed tour-construction heuristics were set to the centers of ellipses.

The C3-Opt is a tour-improvement heuristic, and its efficiency depends on an input tour. We evaluate three variants of the C3-Opt with different input tours:

- NN→C3-Opt, where the initial tour is generated by the Nearest Neighbor algorithm (NN) which iteratively extends a solution with a point nearest to the last added one.
- Rand→C3-Opt, where an initial tour is generated pseudo-randomly by inserting areas in the order in which they are listed in the instance file.
- CIH→C3-Opt, where an initial tour is generated by CIH.

State-of-the-art algorithms

We compare the proposed algorithms with two state-of-the-art algorithms. The first one was proposed by Gentilini et al.[43] who improved a Mixed-Integer NonLinear Program (MINLP) solver by introducing a heuristic. In the following, we will refer to this approach as Heuristic in Solver (HIS).

Another concept to compare with is an idea proposed by Mennell et al. [71] where, first of all, every area is represented with a point and then sequentially TSP and TPP are solved. The original approach called LK-SOCP uses the Lin-–Kernighan (LK) heuristic for solving the TSP and a second-order cone program (SOCP) for the TPP. In the following, we refer to this method as TSP→TPP. Our aim is not to replicate all nuances of the original approach but rather to compare the concept of applying methods for the TSP and TPP sequentially or in parallel. To make an unbiased comparison between the CIH and the TSP→TPP, the same TPP solver is used, i.e., RBA. As a TSP solver, the Nearest Neighbor (NN) algorithm improved with the 3-Opt method is applied.

7.1.2 Evaluation on Instances with Known Optimum

Since heuristic approaches do not guarantee to find the optimal solution, one should estimate how far from optimum the solutions found by a heuristic are. Therefore, in this section, the proposed approaches are evaluated on test instances with known optimal costs. This instances were developed by Gentilini et al. [43] and are available on-line[1] with a precise description. The set contains 24 instances with maximum number of 16 areas. An example of an instance name is "$tspn2DE7\_N$" that stands for a $2D$ instance with 7 ellipses. Number N reflects a box size circumscribing an ellipse and equals "1" or "2". Ellipses with the box size "1" are larger than with box size "2".

The evaluated approaches have the following values of the parameters. The accuracy μ for the CIH and the accuracy ε for the mRBA are set to 0.01. The C3-Opt was executed with the following parameters: ε=20 and μ=20.

The results of the evaluation are presented in Table 7.1. For every instance, the table contains an optimal cost as well as a deviation from the optimum and a run time for each heuristic. The optimum value was calculated by Gentilini et al. [43].

Since both CIH and TSP→TPP are as tour construction heuristics, for both of them, the tour-improvement heuristic 3-$Impr.$ is applied during this evaluation. However, on these test instances, the improvement algorithm has no effect for the TSP→TPP. Therefore, we omit this column from the results table.

The average time for all instances is 650.42 ms for the HIS and only 11.97 ms for the CIH. Even though these computational times cannot be compared directly, as the methods were executed on different hardware, the difference is significant—the CIH outperforms the HIS in 54 times on average. Although this number should not be understood as an unbiased comparison, it shows that our algorithm achieved a significant improvement in computational time.

The TSP→TPP found optimal solutions for 13 instances out of 24. The deviations from optimum produced by the TSP→TPP are not caused by the TPP solver (except "$tspn2DE9\_2$") but rather by a "bad" representation of the areas with initial points, as the tours produced by the TSP solver are optimal. The HIS solved 15 instances out of 24 to optimality with an average deviation of 0.15%. Though the underlaying principal of the CIH is greediness, this method provides good results in practice. The CIH obtained optimal costs on 20 tests out of 24. Application of the 3-$Impr.$ decreased the average deviation from 0.32% to 0.29%.

The use of the greedy NN heuristic to calculate an input tour for the C3-Opt led to results worse than Rand→C3-Opt and CIH→C3-Opt. The average and the maximum deviations are 0.19% and 2.39%, respectively. It could be explained by the fact that the C3-Opt is a local search algorithm, therefore, it has no techniques to escape from a local minimum established by the NN. The average computational time is 38.21 ms.

[1] STSPN Instances: http://wpweb2.tepper.cmu.edu/fmargot/ampl.html

Table 7.1: Performance of the Component1 and state-of-the-art approaches on the instances with known optima.

Instance	Optimal	HIS		TSP→TPP		CIH		CIH (3-*Impr.*)		NN→C3-Opt		Rand→C3-Opt		CIH→C3-Opt	
	value	deviation(%)	$t(ms)$	deviation(%)	$t(ms)$	deviation(%)	$t(ms)$	deviation(%)	$t(ms)$	deviation(%)	$t(ms)$	deviation(%)	$t(ms)$	deviation(%)	$t(ms)$
tspn2DE5_1	191.255	0.00	140	0.00	0.24	0.00	0.87	0.00	1.23	**1.46**	0.69	0.00	1.19	0.00	1.38
tspn2DE5_2	219.307	0.00	130	0.00	0.23	0.00	0.57	0.00	0.67	0.00	0.56	0.00	1.29	0.00	0.84
tspn2DE6_1	202.995	0.00	240	0.00	0.41	0.00	0.93	0.00	1.08	0.00	1.09	0.00	1.83	0.00	1.47
tspn2DE6_2	248.860	0.00	180	0.00	0.38	0.00	0.82	0.00	0.98	0.00	0.69	0.00	1.62	0.00	1.32
tspn2DE7_1	201.492	0.00	300	0.00	0.63	**0.02**	3.46	**0.02**	3.94	0.00	2.95	0.00	2.58	**0.02**	4.50
tspn2DE7_2	239.788	0.00	250	**0.98**	0.61	0.00	1.78	0.00	2.04	0.00	2.27	0.00	5.86	0.00	2.87
tspn2DE8_1	190.243	0.00	370	**0.07**	0.57	0.00	0.42	0.00	0.74	0.00	2.92	0.00	7.90	0.00	2.67
tspn2DE8_2	229.150	**0.01**	400	0.00	0.87	0.00	3.49	0.00	4.11	0.00	3.43	0.00	4.69	0.00	5.36
tspn2DE9_1	259.290	0.00	400	**4.23**	1.34	0.00	5.78	0.00	6.81	0.00	8.00	0.00	11.47	0.00	8.79
tspn2DE9_2	262.815	0.00	410	**2.05**	1.16	0.00	4.58	0.00	5.30	0.00	6.12	**0.01**	13.62	0.00	7.78
tspn2DE10_1	225.126	0.00	410	**0.15**	1.30	0.00	5.84	0.00	6.83	0.00	9.01	0.00	10.89	0.00	11.31
tspn2DE10_2	273.192	**0.21**	350	**0.21**	1.55	0.00	5.08	0.00	6.18	0.00	16.66	0.00	17.48	0.00	10.97
tspn2DE11_1	247.886	**0.75**	630	**0.69**	2.19	0.00	8.02	0.00	9.82	**0.69**	18.79	0.00	24.94	0.00	16.12
tspn2DE11_2	258.003	0.00	390	0.00	2.20	0.00	7.37	0.00	9.04	0.00	25.65	0.00	24.08	0.00	15.46
tspn2DE12_1	265.858	0.00	550	0.00	2.38	0.00	9.54	0.00	11.52	0.00	25.40	0.00	63.28	0.00	21.14
tspn2DE12_2	312.493	**0.50**	860	**2.62**	2.21	0.00	11.89	0.00	13.75	0.00	66.01	0.00	72.10	0.00	23.96
tspn2DE13_1	278.876	0.00	1150	0.00	4.70	0.00	15.24	0.00	18.49	0.00	27.05	0.00	66.96	0.00	32.58
tspn2DE13_2	324.271	**0.20**	490	**0.20**	4.67	0.00	15.33	0.00	18.07	0.00	60.90	0.00	60.43	0.00	34.01
tspn2DE14_1	310.794	0.00	950	0.00	12.45	0.00	22.82	0.00	26.75	0.00	85.96	0.00	99.12	0.00	45.93
tspn2DE14_2	270.638	**0.56**	690	**0.26**	12.38	0.00	18.68	0.00	21.99	0.00	111.71	**0.07**	212.62	0.00	43.78
tspn2DE15_1	289.716	**0.22**	1080	0.00	4.69	0.00	28.28	0.00	34.44	0.00	38.49	0.00	196.19	0.00	60.44
tspn2DE15_2	293.357	**0.01**	1200	**0.02**	7.80	**1.36**	28.06	**1.36**	32.99	**0.01**	64.65	0.00	91.77	0.00	78.50
tspn2DE16_1	369.945	**1.09**	2840	0.00	26.90	**6.26**	26.92	**5.44**	36.35	**2.39**	192.27	0.00	172.19	0.00	152.94
tspn2DE16_2	295.130	0.00	1200	0.00	10.73	**0.01**	61.58	**0.01**	71.31	0.00	145.68	0.00	339.39	0.00	105.34
Average:		0.148	650.42	0.48	4.27	0.319	11.97	0.285	14.35	0.190	38.21	0.003	62.64	0.001	28.73
Max:		1.09	2840.00	4.23	26.90	6.26	61.58	5.44	71.31	2.39	192.27	0.07	339.39	0.02	152.94

The Rand→C3-Opt and the CIH→C3-Opt showed very close solution quality results: an average deviation of 0.003% and 0.001%, respectively. The time with CIH input tour is shorter than with Random tour (28.73 ms versus 62.64 ms). Both of these variants outperformed the other algorithms in the solution quality.

7.1.3 Evaluation on Instances with "Stretched" Ellipses

In this test, we evaluate the efficiency of the CIH and the C3-Opt on larger scenarios.

Since instances proposed by Gentilini et al.[43] have up to 16 areas, we generated a set of instances with up to 60 "stretched" ellipses. The ellipses have different ratio between their axis radii, i.e., stretched along one of the axis. The set contains 18 instances and is available on-line[2].

The test instances are generated according to the following principle. At first, coordinates of the centers of ellipses (x_i, y_i) are selected as a random integer number that lies in the interval $[0, 100]$. A coefficient of elongation CE is a float number chosen from the interval $[A, B]$, where A and B are positive real numbers, with equal probability for each value. To generate instances for this section, intervals [1,1], [1,5] and [1,10] were used. A radius along X-axis is calculated as $R_x = 100/N \times 2 \times Rand$, where N is a desired number of areas in the test and $Rand$ is a random real number that lies in $[0.1, 1]$. A radius along the Y-axis is calculated as $R_y = R_x \times CE$. With a probability of 0.5, R_x and R_y are exchanged. This method allows us to generate test instances with ellipses of different elongations along the axis. The name of an instance "60_1_5" should be understood as a scenario with 60 ellipses which have one of the axis radii stretched from 1 to 5 times in comparison to another axis radius.

Due to the sizes of the instances, it is infeasible to obtain the optimal cost. Therefore, we evaluate the algorithms by comparing them with the best obtained value for each single instance. The best obtained value is the minimal value among results of the seven analyzed approaches.

The evaluated methods were executed with the following parameters: μ=0.01, ε=0.01 for the CIH and ε=20 and μ=20 for the C3-Opt. The results of the evaluation are presented in Table 7.2. The CIH (3-$Impr.$) found best values for 4 out of 18 instances. The average deviation from the best obtained value is 4.67%. The TSP→TPP (3-$Impr.$) reached the best found cost once. Although both the CIH (3-$Impr.$) and the TSP→TPP call 3-Opt once, the CIH (3-$Impr.$) finishes its computations on average faster. The reason is that in the TSP→TPP, the major time is spent on 3-Opt execution, as the initial tour is far from being optimal. In contrast, in the CIH (3-$Impr.$), an initial tour obtained by the CIH is close to optimum. Therefore, the 3-Opt makes less exchanges and requires less time.

[2]TSPN Test Instances: see https://cse.cs.ovgu.de/cse/robotics/tspn/, accessed on August 26, 2015

Table 7.2: Performance of the Component1 and state-of-the-art approaches on test instances with "stretched" ellipses.

Instance	Best obtained value	TSP→TPP		TSP→TPP (3-*Impr.*)		CIH		CIH (3-*Impr.*)		NN→C3-Opt		Rand→C3-Opt		CIH→C3-Opt	
		deviation(%)	$t(s)$	deviation(%)	$t(s)$	deviation(%)	$t(s)$	deviation(%)	$t(s)$	deviation(%)	$t(s)$	deviation(%)	$t(s)$	deviation(%)	$t(s)$
20_1_1	318.904	**0.56**	0.02	**0.56**	0.03	**2.39**	0.09	**1.82**	0.13	**0.62**	0.22	0.00	0.74	0.00	0.37
20_1_5	312.915	**0.18**	0.02	**0.18**	0.03	**3.30**	0.08	**3.30**	0.09	0.00	0.43	0.00	1.44	0.00	0.39
20_1_10	252.350	**17.92**	0.06	**15.21**	0.07	0.00	0.14	0.00	0.14	**1.40**	1.29	**1.10**	0.70	0.00	0.22
30_1_1	383.578	**0.06**	0.28	**0.06**	0.38	**1.44**	0.22	**1.37**	0.27	**0.77**	1.96	0.00	3.94	0.00	2.13
30_1_5	316.854	**10.71**	0.55	**8.49**	0.76	0.00	0.31	0.00	0.36	**1.08**	6.90	**0.11**	12.15	0.00	0.96
30_1_10	306.338	**11.66**	0.49	**8.66**	0.59	0.00	0.44	0.00	0.49	**0.12**	3.57	**1.50**	6.08	0.00	1.05
40_1_1	416.556	**1.99**	1.87	**1.99**	2.04	**3.58**	0.42	**2.29**	0.80	0.00	28.93	0.00	51.89	**0.46**	8.33
40_1_5	366.637	**4.59**	1.57	**0.59**	1.96	**0.53**	0.75	**0.53**	0.91	**3.77**	12.42	0.00	45.46	**0.53**	2.94
40_1_10	311.714	**15.53**	1.11	**15.45**	1.48	0.00	0.91	0.00	1.08	**4.55**	38.34	**0.31**	52.75	0.00	3.11
50_1_1	438.215	**0.24**	1.95	0.00	2.39	**3.10**	0.82	**1.51**	2.39	**0.01**	54.17	0.00	158.94	**0.03**	64.49
50_1_5	435.158	**5.97**	2.84	**5.04**	3.85	**6.97**	1.34	**6.32**	2.02	**0.23**	103.14	**0.65**	247.36	0.00	51.63
50_1_10	391.303	**14.26**	4.00	**10.09**	4.76	**2.44**	1.66	**1.70**	2.82	0.00	135.43	**0.85**	208.36	**0.26**	45.37
60_1_1	559.042	**0.81**	9.65	**0.81**	12.42	**8.87**	1.23	**6.48**	7.00	**0.41**	154.66	0.00	265.01	**1.00**	143.74
60_1_5	550.121	**6.67**	10.97	**5.94**	14.02	**2.93**	1.69	**2.71**	4.23	**2.27**	151.88	0.00	473.45	**0.41**	101.05
60_1_10	482.289	**11.55**	10.23	**9.02**	12.08	**7.85**	1.86	**6.73**	3.39	**2.20**	179.61	0.00	467.24	**0.21**	126.04
70_1_1	599.819	**5.54**	26.47	**5.54**	29.20	**5.74**	2.19	**4.95**	9.07	**4.82**	419.90	**0.20**	544.83	0.00	270.63
70_1_5	564.303	**4.79**	30.65	**4.79**	37.05	**7.60**	2.59	**3.92**	13.27	**0.52**	717.22	0.00	959.91	**0.78**	200.96
70_1_10	447.452	**16.07**	24.63	**14.09**	29.74	**9.28**	3.30	**9.03**	11.79	**5.17**	634.06	0.00	906.24	**2.08**	217.23
Average:		7.17	7.08	5.92	8.49	3.67	1.11	2.93	3.35	1.55	146.90	0.26	244.81	0.32	68.92
Max:		17.92	30.65	15.45	37.05	9.28	3.30	9.03	13.27	5.17	717.22	1.50	959.91	2.08	270.63

The method of sequential solving of the TSP and then TPP (TSP→TPP) gives worse results than the method of solving both at the same time (CIH or C3-Opt). The reason is that during solving of a TSP, calculation the areas are represented as points (e.g., geometrical centers of ellipses for the instances used in this evaluation) and information about the overall shape is ignored. The obtained tour could be optimal, but only with regard to the chosen points. Applying a TPP method afterwards will improve the obtained solution by optimizing entry points inside the areas. But such strategy does not consider how shapes of the areas affect a sequence in which the areas are visited. Application of both methods at the same time, in contrast, means that during the calculation of a TSP tour we also coherently optimize the point locations inside the areas. Therefore, it enables consideration of certain information about shapes of the areas during the process of TSP solving. Thus, a final sequence obtained by the CIH is better than a sequence calculated by the TSP→TPP, even if the same algorithms for a TSP and a TPP are applied in both strategies.

The NN→C3-Opt produced the worst results among all C3-Opt variants. Nevertheless, these results are better than the ones obtained by the CIH and the CIH (3-*Impr.*). The best results were achieved by the Rand→C3-Opt, however, it required much more time than other approaches. A compromise between solution quality and run time was demonstrated by the CIH→C3-Opt. It produces results only slightly worse than the Rand→C3-Opt (deviation 0.32% versus 0.26%), but significantly outperforms it in computational time (68.92 s versus 244.81 s).

The C3-Opt required more time for computation than the CIH but produces better solutions. The CIH→C3-Opt was able to produce solutions for instances with 30 ellipses within 3 s, for 40 ellipses within 9 s and for 50 ellipses within 65 s. Note that four new best known values were obtained by incorporating the Bisection method into the CIH. The main advantage of the C3-Opt over the CIH is that it has no high "jumps" of deviation and constantly produces good results.

7.1.4 Evaluation on Instances for CETSP

Even though our algorithms were developed for a more general problem (TSPN), we also evaluate them on test instances for the CETSP developed by Mennell et al. [71] and accessible online[3]. This test set contains 20 instances with up to 595 areas. Due to large sizes of the instances, computation time of the NN→C3-Opt, Rand→C3-Opt and CIH→C3-Opt increases dramatically and, therefore, these methods are not evaluated on these instances.

The CIH heuristic was applied with and without a tour-improvement heuristic. During this test, the 2-*Impr.* is applied for the improvement phase. The TSP tour-improvement heuristic 3-Opt used for the previous evaluations within 3-*Impr.* was substituted with the 2-Opt, as the 2-Opt requires significantly shorter computational time. For example, the instance "bubble5" was solved by CIH in 167.5 s and the obtained solution was 19.72% worse than the best known solution, whereas solutions found by the

[3] http://www.minlp.org/library/problem/index.php?i=65&lib=MINLP

Table 7.3: Performance of the CIH and CIH (2-*Impr.*) approaches on CETSP instances.

Instance	N of areas	Best known value	CIH dev.(%)	CIH $t(s)$	CIH (2-*Impr.*) dev.(%)	CIH (2-*Impr.*) $t(s)$
concentricCircles1	17	53.158	0.00	0.01	0.00	0.02
concentricCircles2	37	153.132	5.32	0.38	5.32	0.40
concentricCircles3	61	271.076	4.15	1.88	4.15	1.92
concentricCircles4	105	454.457	4.71	9.23	3.13	9.56
concentricCircles5	149	645.381	6.26	28.02	5.35	29.13
Average			4.09	7.90	3.59	8.21
bubbles1	37	349.135	0.05	0.56	0.05	0.57
bubbles2	77	428.279	0.85	5.20	0.84	5.34
bubbles3	127	530.733	0.33	22.41	0.33	23.50
bubbles4	185	829.888	10.87	67.50	10.51	188.40
bubbles5	251	1062.335	19.72	167.45	18.82	179.25
bubbles6	325	1383.139	13.87	355.69	10.46	409.91
bubbles7	408	1720.214	19.52	693.41	15.64	819.52
bubbles8	497	2101.373	20.47	1263.88	17.74	1826.49
bubbles9	595	2426.274	27.18	2147.98	23.04	2803.94
Average			12.54	524.90	10.83	695.21
team1_100	100	307.337	2.61	10.62	2.61	10.71
team2_200	200	246.683	1.22	93.39	1.22	86.39
team3_300	300	466.241	12.20	258.50	12.20	836.56
team4_400	400	680.211	8.02	606.50	7.77	635.70
team5_499	499	702.823	11.50	1068.40	10.60	1418.90
team6_500	500	225.216	0.19	1168.12	0.17	1230.18
Average			5.96	534.25	5.76	703.07

2-*Impr.* and 3-*Impr.* were 18.82% worse (179.25 s) and only 16.78% worse (2975.2 s), respectively. Such large difference in run time is caused by a larger number of possible edge exchange combinations in 3-Opt and, thus, more possibilities for improvement.

The CIH is executed with accuracy μ=0.01. Since the larger test instances contain up to 595 areas, two speed up techniques were introduced: (1) p_{temp} is selected randomly in line 9 in Algorithm 4.3, and (2) accuracy ε is equal to 0.5 in line 14 in Algorithm 4.3 and 0.01 in all other mRBA calls.

Mennel et al. [71] evaluated 11 heuristics for the CETSP using the same instances. These heuristics were developed specifically for the CETSP and, therefore, are based on the fact that areas are represented as disks. This information about the shape allows for developing very efficient methods that are, however, restricted to this problem. In contrast, the CIH was developed for the TSPN and is capable of solving scenarios with arbitrary shapes. Therefore, the comparison of the CIH and the heuristics of Mennel et al. is a stress test for our heuristic and is highly biased in favor of the other heuristics.

The performance of the CIH in comparison with the best known solutions is shown in Table 7.3.

Even in the unfavorable conditions, the CIH (2-*Impr.*) achieved the third best result for the "bubble" instances 1–3 comparing to the heuristics of Mennel et al. Since our algorithm is not tuned for the problem, its solution quality deviates more and more from the other 11 heuristics with the number of areas larger than 100. Thus, the overall performance of our heuristic on the "bubble" instances is on the 11th place. For the "concentricCircle" instances, the CIH (2-*Impr.*) took the 7th place and the 8th place for the "team" instances.

This evaluation shows that even though the CIH is a heuristic for the TSPN, a more general problem, it is still capable of solving CETSP instances even better than some specialized CETSP heuristics.

7.1.5 Evaluating the Influence of the Precision Parameters

The CIH and the C3-Opt efficiency depend on the two precision parameters ε and μ. In order to evaluate it, we select the following values: 0.01, 1, 10, 20 and 30, and assign them to ε and μ in all possible combinations. The solutions obtained by the CIH and C3-Opt with these values are then optimized by the RBA. An initial solution for the C3-Opt is constructed randomly. The evaluation is conducted on the benchmarks from Section 7.1.2.

The results of the CIH are presented in Figure 7.1 and of the C3-Opt in Figure 7.2. Obviously, the time to solve an instance increases with smaller values of precision parameters ε and μ. For the CIH, the average solving time varied from 4 ms when ε=30, μ=30 up to 18 ms when ε=0.01, μ=0.01. For the C3-Opt, the average solving time ranged from 44.94 ms when ε=30, μ=30 to 167.25 ms when ε=0.01, μ=0.01.

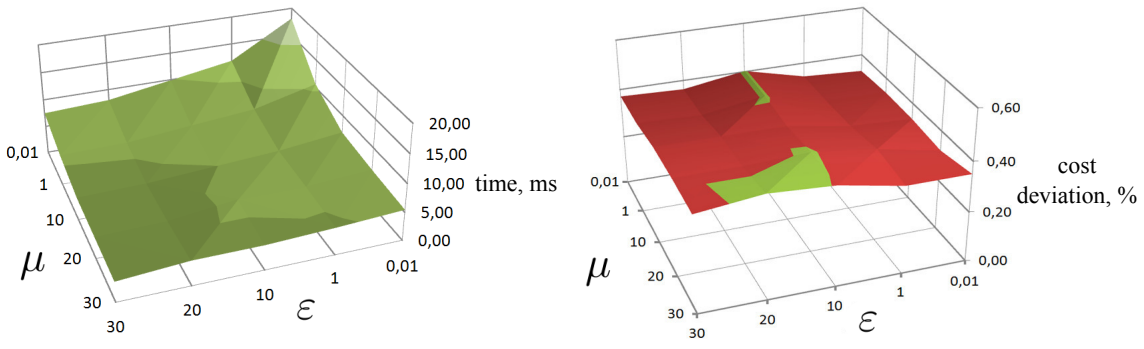

Figure 7.1: CIH dependence of the time (left) and the cost given as a deviation from the best solution (right) on precision parameters ε and μ

According to the results, change of parameter values causes a very insignificant cost deviation. The reason is the fact that we applied the mRBA algorithm after the CIH and the C3-Opt to minimize the error accumulation of the point location on the

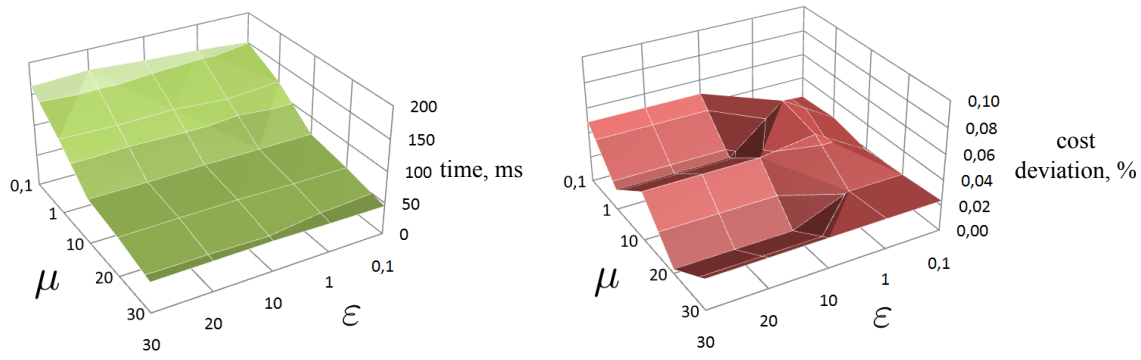

Figure 7.2: C3-Opt dependence of the time (left) and the cost given as a deviation from the best solution (right) on precision parameters ε and μ

ellipses borders, which appears with large precision values. For the CIH, the average cost deviation grows slowly from 0.32% when ε=0.01, μ=0.01 up to 0.4% when ε=30 and μ=30. For the C3-Opt the average cost deviation ranges from 0.003% when ε=20, μ=20 up to 0.05% when μ=0.1 and ε=10, 20 or 30.

The reported results show that both the CIH and the C3-Opt are very stable even if the precision values are very inaccurate.

7.2 Evaluation of the Component2 Approaches

This section describes an evaluation of the Nested RBA, proposed for optimizing the entry points for the relaxed effective task sequence. The objective is to obtain minimal cost supporting trajectories between given effective tasks. The evaluation was conducted on two test cases inspired by real-world applications: cutting holes in plastic cover[4] and dashboard casing[5]. A demo video is available on-line[6]. The robot layout is depicted in Figure 7.4.

For the evaluation, we used a system with an Intel Core 2 Quad CPU, 2.83 GHz with 8 GB RAM, running Ubuntu 12.04. The Nested RBA is implemented using Python in OpenRAVE [31]. Inverse kinematics is calculated analytically with the IKfast generator for the KUKA KR30L16 robot. All robot configurations are checked for being not in a collision with environment.

The two test scenarios have the following parameters. In a plastic cover case study, a robot has to cut out 5 holes and to deburr the border of a plastic cover detail. This results in 6 closed-contour effective tasks. The second case study is a dashboard casing that consists of 15 closed-contour effective tasks. In both scenarios, it is possible to perform cutting starting from any point within a closed-contour. A starting point has also an orientation freedom for an end-effector. An orientation freedom is defined

[4]Plastic Cover cutting: http://www.youtube.com/watch?v=bS2ESlSvdEo
[5]Dashboard Casing: http://www.youtube.com/watch?v=p0CZnmmaspk
[6]Demo video: https://cse.cs.ovgu.de/cse/robotics/

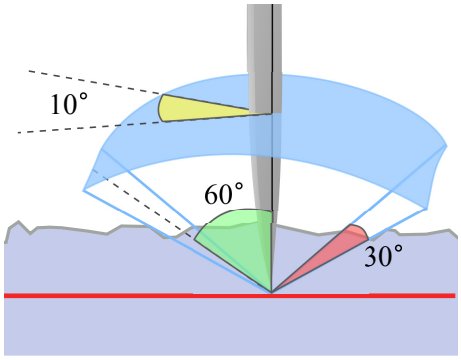

Figure 7.3: Defined orientation window for a knife.

for the both case studies with following parameters: a_l=30, a_u=150, b_l=−15, b_u=15, c_l=−5, c_u=5, and it is the same for all possible entry points of all tasks. This orientation window is depicted in Figure 7.3. This definition of the orientation freedom represents to the sphere segment areas for an end-effector to reach.

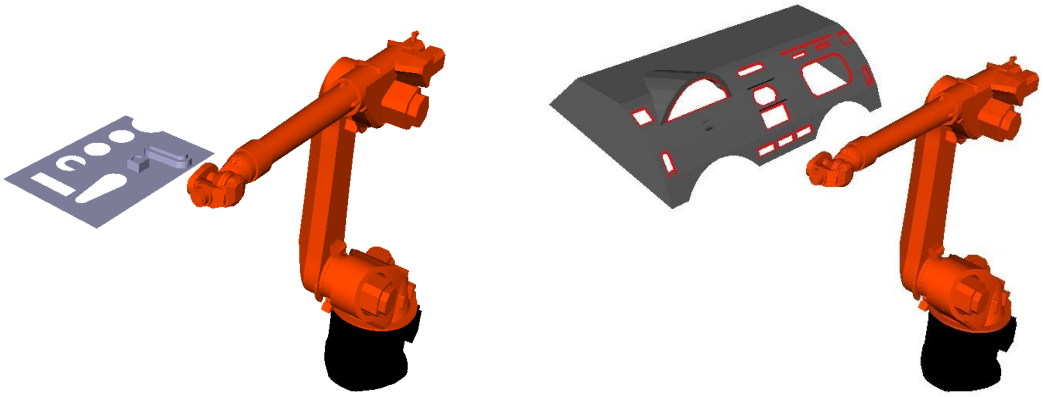

Figure 7.4: Evaluation was performed on two examples: plastic cover (left) and car dashboard casing (right).

We evaluate a sequence with three metrics: Euclidean distance of an end-effector path, Time and C-space distance. A Euclidean distance for an end-effector is measured in meters and represents a Euclidean length of a tour consisting of end-effector positions (i.e., point x_{ef}, y_{ef}, z_{ef} in Figure 5.2). For given Cartesian entry points, end-effector orientation angles a, b, c are chosen as the middle values of the corresponding allowed orientation window, i.e., intervals $[a_l, a_u]$, $[b_l, b_u]$ and $[c_l, c_u]$, respectively. The Time metric for a tour is the time in seconds required to perform all supporting movements. It is obtained by the build-in OpenRAVE function *RetimeActiveDOFTrajectory*() using the Parabolic Trajectory Retimer. The C-space distance is calculated as a Euclidean distance between robot configurations, i.e., a distance between two 6D points. For simplicity, it is calculated without joint weight coefficients [56]. However, as we show further, using the Euclidean distance formula for C-space can effectively substitute

the Time metric in calculation. C-space distance for a tour is the sum of all C-space distances for all supporting movements.

In the following, we evaluate three approaches:

- Component1: A sequence of tasks calculated by the CIH and improved by the 2-Opt [29] using the Euclidean distance for an end-effector.
- Component1 → Component2: A sequence and Cartesian entry points are calculated with the CIH and the obtained solution is improved with the 2-Opt. A solution for the Component2 is calculated with the nested approach proposed in Chapter 5.
- Component1+Component2: The CIH constructs a task sequence, but as a constricting mechanism applies Component2 approach instead of the mRBA.

Since the components solve different problems, they are evaluated with regard to different metrics. Solution quality of the approach Component1 is measured as the Euclidean distance for an end-effector. Both Component1→Component2 and Component1+Component2 are evaluated on the Time and C-space cost functions.

The achieved results are presented in Figure 7.5 and Table 7.4. Figure 7.5 depicts the obtained robot's paths

The approach Component1 outperforms Component1→Component2 and Component1+Component2 with regard to the Cartesian distance metric. However, since it internally uses only this metric, the approach achieves significantly worse results with regard to the robot-based metrics (Time and C-space distance). The approach Component1+Component2 outperformed Component1→Component2 on 0.47% (plastic cover, C-space distance cost), 8.78% (dashboard, C-space distance cost), 3.96% (dashboard, Time cost). However, it was worse on 3.69% on the plastic cover scenario with regard to the Time metric.

A convergence rate is evaluated for the Component2. We report results in Figure 7.6 for up to 30 and 100 iterations for the plastic cover and dashboard casing scenarios, respectively. The figure shows the decrease of a cost (C-space distance and Time) with respect to the number of iterations. The first iterations often bring large improvements. The algorithm only returns the solution that is better than current one on every iteration. However, the improvement can be insignificantly small for multiple iterations in a row. For example, iterations 35–60 in Dashboard Casing scenario while Time optimization. Afterwards, the algorithm can significantly reduce the cost. Therefore, the use of the difference between costs from two consequent iterations as a stopping condition is undesirable. We recommend to use a maximum computational time or a number of iterations for stopping condition.

The approach CIH from Component1 found solutions for the plastic cover scenario in only 2 s and for the dashboard casing scenario in 28 s. It is explained by the algorithm's simplicity and the computationally cheap Cartesian cost function.

Table 7.4: Evaluation results on the plastic cover and dashboard casing scenarios.

Plastic cover, minimizing C-space distance:

	Component1	Component1→Component2		Component1+Component2	
	value	value	improvement(%)	value	improvement(%)
Cartesian dist. (m)	1.899	2.036	-7.23	2.329	-22.65
Time (s)	4.063	2.224	45.26	2.350	42.16
C-space dist. (rad)	8.450	1.354	**83.98**	1.313	**84.45**
Calculation time (s)	2	20		787	

Plastic cover, minimizing Time:

	Component1	Component1→Component2		Component1+Component2	
	value	value	improvement(%)	value	improvement(%)
Cartesian dist. (m)	1.899	1.934	-1.85	2.069	-8.96
Time (s)	4.063	2.087	**48.62**	2.237	**44.93**
C-space dist. (rad)	8.450	1.761	79.15	3.075	63.60
Calculation time (s)	2	103		2158	

Dashboard casing, minimizing C-space distance:

	Component1	Component1→Component2		Component1+Component2	
	value	value	improvement(%)	value	improvement(%)
Cartesian dist. (m)	5.758	6.332	-9.96	7.130	-23.83
Time (s)	10.973	6.829	37.75	6.441	41.29
C-space dist. (rad)	24.459	6.401	**73.82**	4.255	**82.60**
Calculation time (s)	28	68		6907	

Dashboard casing, minimizing Time:

	Component1	Component1→Component2		Component1+Component2	
	value	value	improvement(%)	value	improvement(%)
Cartesian dist. (m)	5.758	5.885	-2.20	5.607	2.61
Time (s)	10.973	6.021	**45.12**	5.587	**49.08**
C-space dist. (rad)	24.459	8.063	67.03	7.285	70.21
Calculation time (s)	28	482		25622	

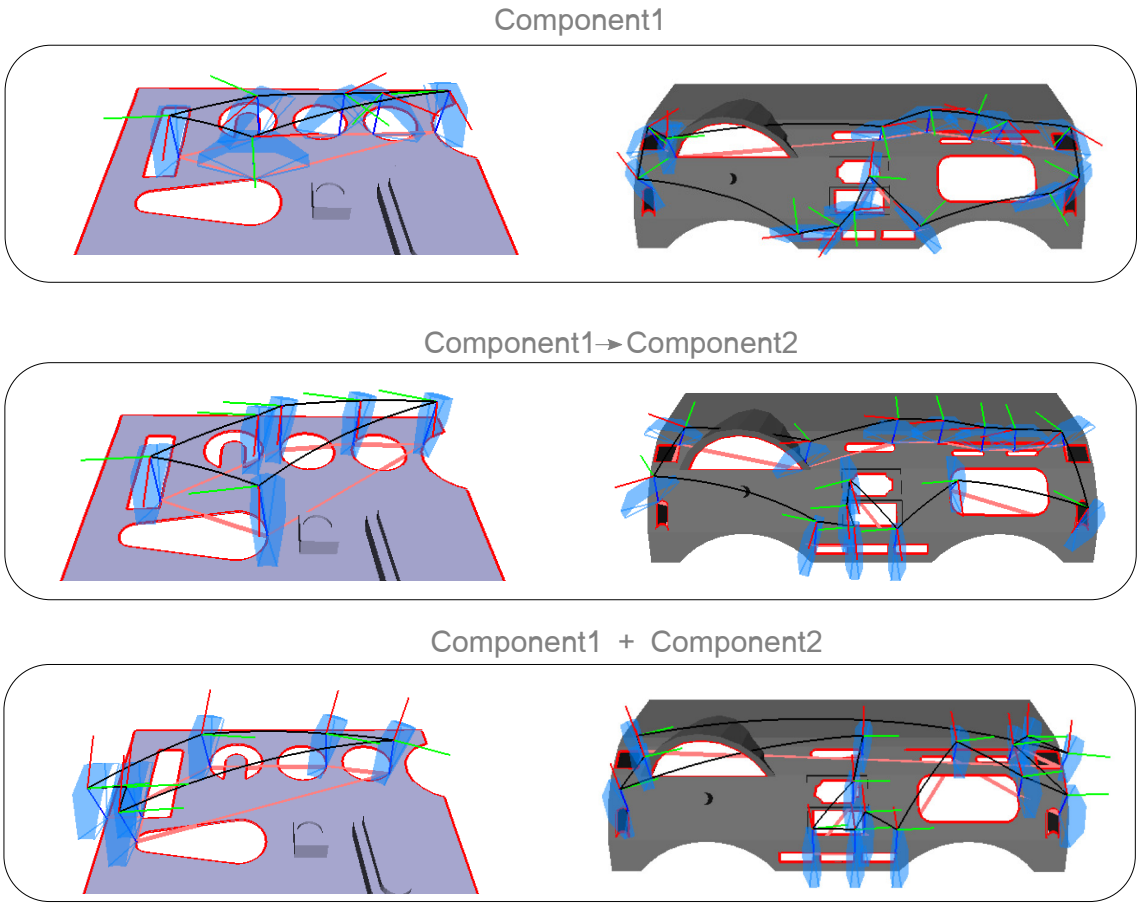

Figure 7.5: Obtained solutions for the two examples: car dashboard casing (right side) and plastic cover (left side). The robot end-effector path is depicted in black and task geometries are depicted in red. Both Component1→Component2 and Component1+Component2 minimized traveled distance in C-space.

Component1→Component2 was slower than Component1 (i.e., CIH) and accomplished optimization for the plastic cover scenario for the C-space and Time metrics in 20 s and 103 s, respectively. The Dashboard scenario, was optimized in 68 s. and 482 s. for the C-space and Time metrics, respectively.

The third evaluated approach, Component1+Component2, required much longer computational time, due to the fact that at every decision step of the CIH, the problem of the Component2 has to be solved. Therefore, solving of the plastic cover was finished for the C-space and Time metrics in 787 s and 2158 s, respectively. Optimization for the dashboard casing scenario required even more computational time: 6907 s. for C-space distance cost and 25622 s. for the Time metric.

Use of the Time metric instead of the C-space distance metric increases the run time, as it recalculates the robot trajectory multiple times. Since the two metrics—distance traveled in joint space and the time—are interconnected, it is possible to use

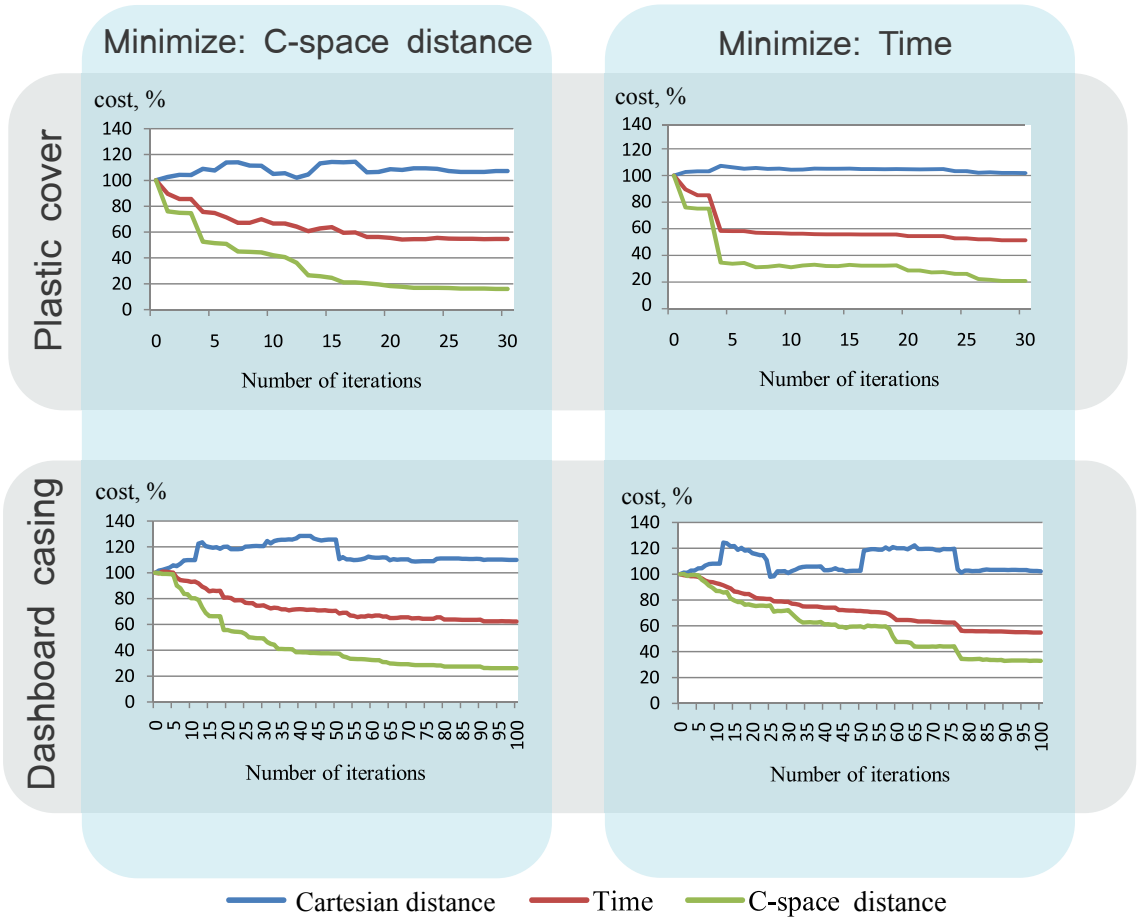

Figure 7.6: Convergence rate for the Component1 approach. During one iteration, one task is optimized.

the C-space metric in order to optimize the Time cost. Using the Time metric makes sense for applications where execution time is critical. After optimizing C-space distance or Time, a Cartesian end-effector path length increases on 2-10%. It means that a tool center point has to follow a longer root than in the initial solution. However, since optimality of the robotic costs (C-space and Time) is of a higher interest, T-space cost increase is not important.

To summarize, the approach Component1→Component2 is the best compromise between the short computational time of Component1 and the good solution quality of Component1+Component2. Component1 is faster than the other two approaches because it does not use robotic metrics. Component1+Component2 achieves better robot-based costs due to the fact that its constricting function explores not only possible positions of an entry point but also orientations and robot configurations, i.e., execute Component2 approach.

7.3 Evaluation of the Component3 Approach

In this section, we present an evaluation of the approach proposed for robot trajectory optimization of a relaxed task. We illustrate the importance of the problem with two robotic applications: a C-Arm robot performing a 3D-angiography and an industrial robot performing cutting-deburring tasks on the plastic cover from Section 7.2.

Proposed in Component3 approach was ran on the following hardware: Intel Core i7 3.20 GHz with 32 GB of RAM, running Ubuntu 12.04. The approach was implemented in Python in OpenRAVE. Inverse kinematics for KUKA KR30L16 was calculated analytically by the IKfast generator.

For the evaluation, we used an industrial robot with 6 degrees of freedom. To limit joints' velocities and accelerations, we set the constraints listed in Table 7.5.

Table 7.5: Upper bounds for joints' velocity and acceleration.

Joint's Number	Maximum velocity (Rad/s)	Maximum acceleration (Rad/s^2)
Joint 1	1.74	4.36
Joint 2	1.39	4.36
Joint 3	1.39	4.36
Joint 4	4.01	8.72
Joint 5	2.87	8.72
Joint 6	4.34	17.45

In this evaluation, we optimize the jerk of a robot's C-space trajectory. We use the following objective function which minimizes the maximum jerk values among all joints throughout trajectory execution:

$$\max_{i\in[1,\dots,ndof]}(\max_{t\in[0,\dots,T]}(\frac{\partial^3 Traj_{R_i}(t)}{\partial t^3})) \rightarrow min \ , \tag{7.1}$$

where $ndof$ is the number of robot degrees of freedom, $Traj_{R_i}(t)$ is the C-space trajectory of the i-th joint, T is the movement duration.

Jerk minimization reduces the error of a path tracker while movement execution. In addition, trajectories with a small jerk reduce wear of the robot and, as a consequence, increase its life span [90].

A motion law was optimized using an idea similar to the algorithm proposed by Chettibi et al. [24]. We first generate nodes on the motion law curve with equal distances between each other and then improve their positions with a Pattern Search.

7.3.1 Case Study: C-arm Robot for 3D-angiography

An angiography is a medical technique which is applied to visualize the inner parts of a human body. It is often used to visualize veins and arteries. This is done by injecting a radio-opaque contrast agent and then performing X-ray scans.

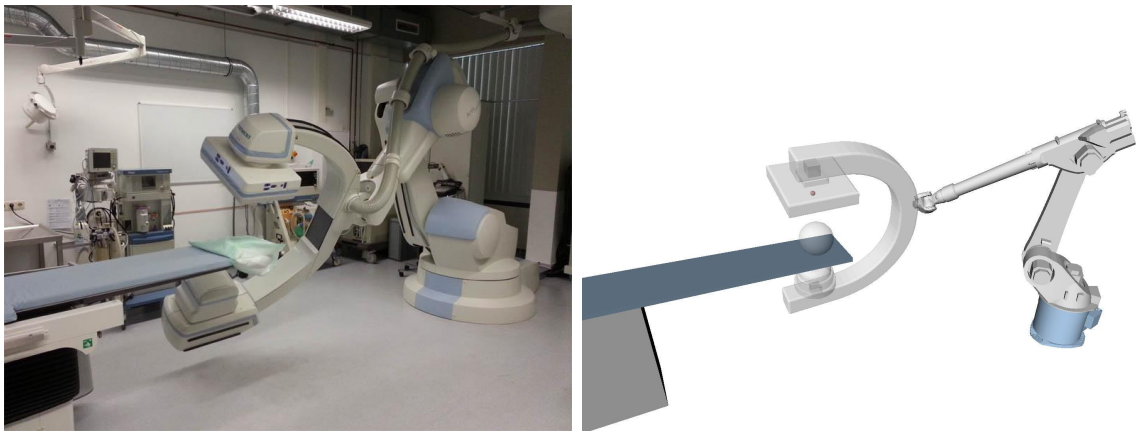

Figure 7.7: Layout of the robot equipped with the C-arm.

A special type of angiography is a rotational angiography. To perform a rotational angiography, an X-ray emitter is rotated around a patient focused on a point of interest. It acquires multiple picture-scans taken from different position. These images are then combined into a 3D volume. Rotational angiography is normally performed with a C-arm. It is a rotational horseshoe-shaped device, equipped with an X-ray source and a detector.

One disadvantage of a C-arm is that it is not mobile and often occupies a lot of space in a surgery room. In order to increase mobility of C-arms so that they can cover all parts of a patient's body, they are mounted on industrial robots, see Figure 7.7 and Figure 7.8. We consider only the degrees of freedom of the robot and omit degrees of freedom of the C-arm for simplicity.

It is critical to know the exact position and time when each picture was taken. Following an imprecise trajectory influences the quality of the final 3D volume, i.e., makes it blurry. The path of the X-ray source is specified for a certain task without considering robot kinematics. One possible way to obtain a high-quality trajectory of a source is to make the robot trajectory smooth by minimizing jerk in its joints.

This application scenario allows a certain freedom for a path. For example, a source can be closer or further from a point of interest (in our scenario this deviation is 0.02 m.). In addition, an approaching vector can have a deviation of several degrees, e.g., 6° for our scenario. This freedom results in truncated-cone via-volumes, which a C-arm source path has to visit. A path and its freedom are shown in Figure 7.8. In any point of a via-volume, the approaching vector of a source is directed to a point of interest called the isocenter. A similar freedom description was used for a laser-welding application [59].

Two different velocity profiles are considered: a trapezoidal velocity profile (case "A") and minimal-jerk velocity profile optimized for an initial path (case "B"), see Figure 7.9. In the case "A", a desired trapezoidal velocity allows us to take pictures with a constant velocity in the middle segment of a path but this leads to an "awkward"

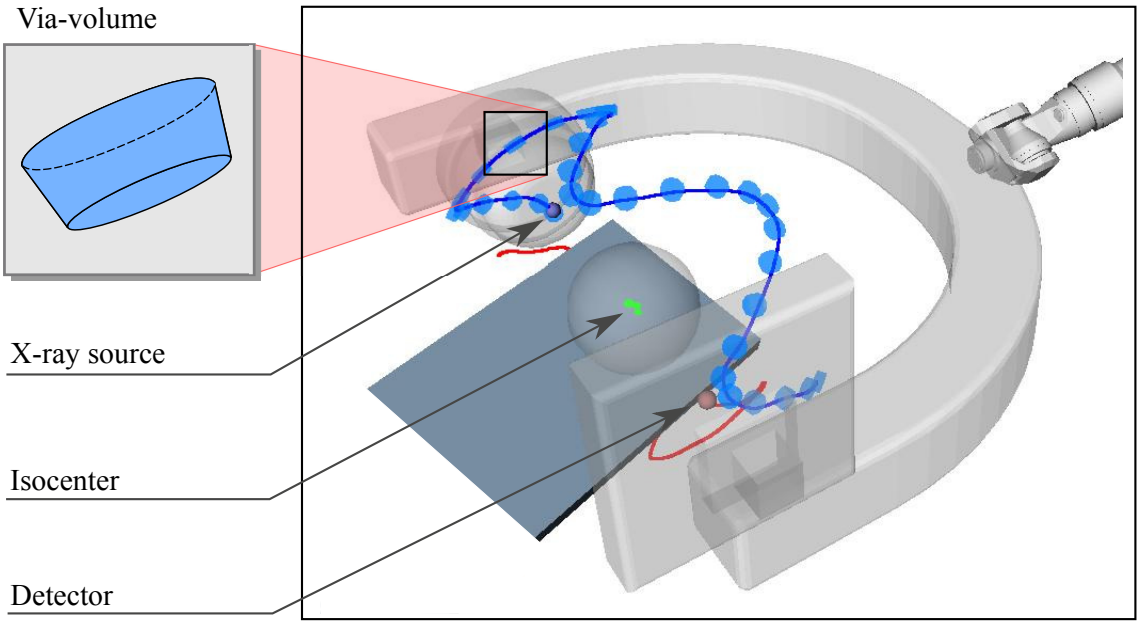

Figure 7.8: The path of the C-arm X-ray source is depicted with blue, the path of the detector is red. The point of interest is the middle of the sphere. The relaxed path is depicted with light-blue volumes.

robot C-space trajectory with high jerks, i.e., the cost of an initial trajectory, i.e., the maximal jerk, is 123 $\mathrm{Rad/s^3}$. In the case "B", a motion law was optimized to obtain a minimal-jerk C-space trajectory for the initial path. The obtained velocity profiles are depicted in Figure 7.9. A trajectory cost after motion law optimization for the case "B" is 30.2 $\mathrm{Rad/s^3}$. Since C-arm robot movements are predefined and constant, it is possible to calculate them offline, i.e., there is no restriction on run time.

The proposed approach Smoothed RBA significantly improved quality of initial trajectory. After applying the proposed heuristic for 10 iterations, the cost was decreased to 26 $\mathrm{Rad/s^3}$ for the case "A" and to 17.5 $\mathrm{Rad/s^3}$ for the case "B". Due to multiple calls of inverse kinematics, the computational time is 63 min.

The rate of convergence is depicted in Figure 7.10. This study shows that the end-effector path relaxation decreases a robot trajectory cost (maximal jerk) regardless of whether a motion law was optimized or not. It is even more efficient to relax a path in conjunction with the motion law optimization.

7.3.2 Case Study: Plastic Cover

After a sequence of tasks and entry points is calculated, optimization of each effective task can be performed. In this section, we evaluate the Component3 approach in the plastic cover scenario.

We perform the evaluation on two extreme effective tasks—a rectangle and a circular task. A circle is a very smooth path for the robot's tool center point to process.

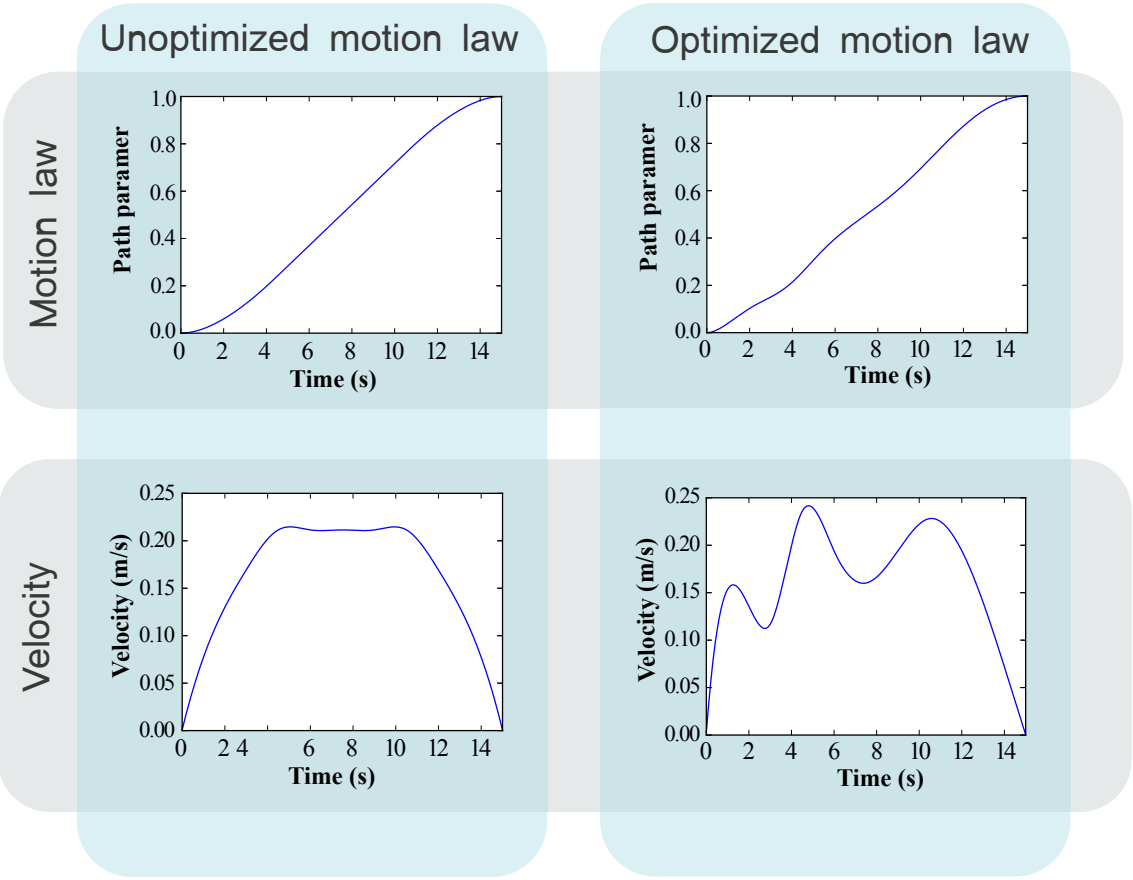

Figure 7.9: Given motion laws and computed end-effector velocities.

It results in a smooth C-space trajectory with no high jerks. In contrast, a rectangle shape has 4 corners where a knife must be turned very quickly at 90 degrees. In reality, a velocity profile should be chosen in a such way that the TCP makes a stop at a corner, then the direction of the knife is changed and the movement continues. We present an extreme case where a TCP does not make a full stop, but rather continues following a motion law that is "uncomfortable" for a robot to execute. It results in an exceedingly high jerk in the 6th robot's joint, as it has to rotate the knife in a corner of a rectangle almost immediately. We show that even in this extreme case, relaxing a task and applying the Component3 approach can minimize jerk value by distributing the load to other joints.

Rectangular-shaped Effective Task

In the following Smoothed RBA from Component3 is evaluated on the Rectangular-shaped effective task from the plastic cover case study.

We evaluate Smoothed RBA with two different motion laws, see Figure 9.1. The first motion law leads to a trapezoidal velocity profile that is "uncomfortable" for a robot to execute, therefore, the maximum robot trajectory jerk is 525.01 Rad/s^3. Such high

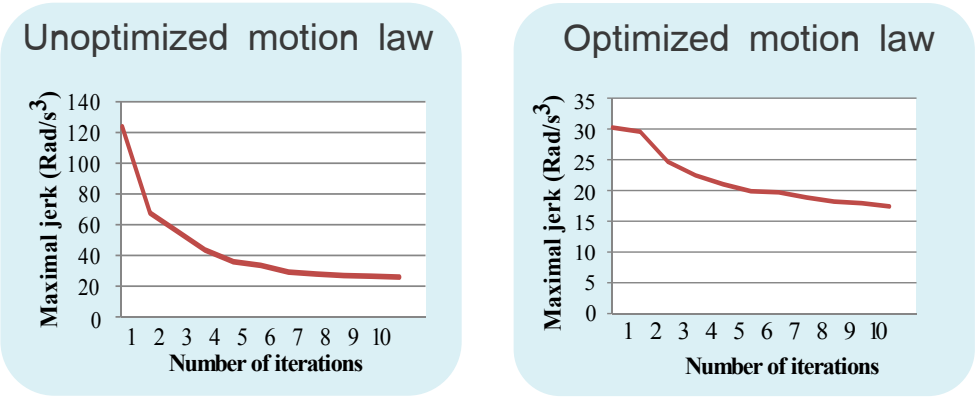

Figure 7.10: Convergence rates of optimization for the given motion laws.

jerk is caused by a jerky rotation of the 6th joint in the corners of the rectangle. In the second case, a motion law is optimized for the initial path with approach presented in [24]. Motion law optimization required 361 s. The corresponding maximal jerk equals to 290.14 Rad/s^3. The highest jerk jump happens on the 14 *s* of execution, see Figure 9.3. Therefore, the motion law optimization algorithm decreases the velocity in the area around 14 *s*, see Figure 9.1 The initial values joints' angles for the unoptimized motion law are depicted in Figure 9.3.

The method proposed for the Component3—Smoothed RBA—significantly improved a value of maximal jerk for both motion laws. The results are shown in Figure 9.3. After performing an optimization, for the case with the unoptimized motion law, the maximal jerk is 8.5 Rad/s^3. For the case with the optimized motion law, the jerk value is equal to 8.61 Rad/s^3. The difference between the initial and optimized end-effector paths are depicted in Figure 7.11.

Smoothed RBA was executed for 25 iterations for both: optimized and unoptimized motion law. The overall optimization required 8980 s.

Circle-shaped effective task

Similar to rectangular-shaped task, the evaluation is performed with two motion laws, see Figure 9.2. The first motion law is unoptimized and maximal jerk of the robot trajectory is 25.58 Rad/s^3. In the second case, motion law was optimized for the initial path. The highest value of a jerk appeared on 6-7 s., see Figure 9.4. Therefore, the motion law optimization aimed at decreasing the velocity in these time points, see Figure 9.2. The jerk of the robot trajectory for the optimized motion law is 4.18 Rad/s^3. The results achieved on this instance are reported in Figure 9.4.

For the case with the unoptimized motion law, Smoothed RBA reduced the maximal jerk value to 5.06 Rad/s^3 and, for the case with the optimized motion law, to 2.96 Rad/s^3.

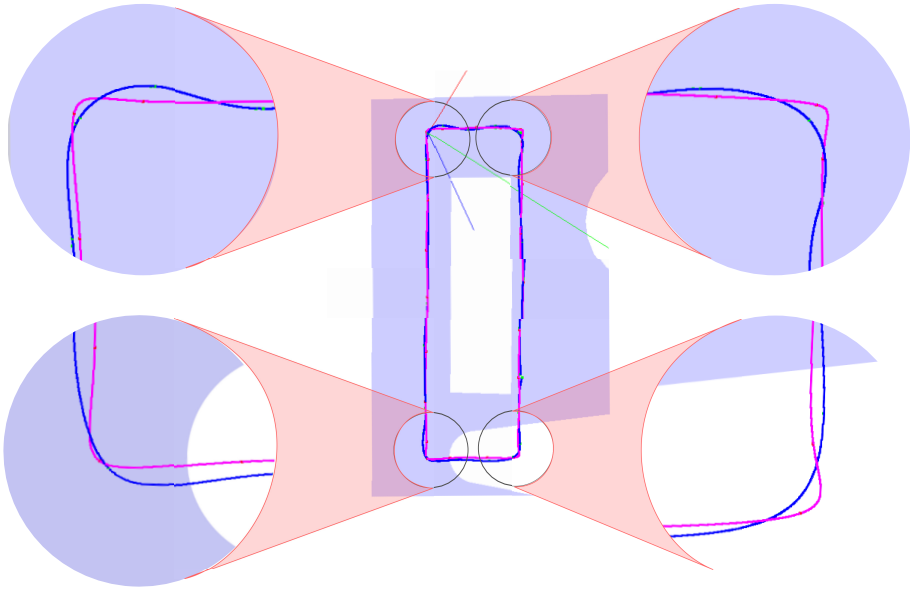

Figure 7.11: The initial path of the robot's end-effector for the rectangular task is designated with purple. The optimized path is blue.

Conclusion

It was shown that relaxing an effective task can lead to a significant reduction of a trajectory cost, in particular, the maximal jerk. This improvement is achieved regardless of whether the motion law was defined by an industrial process or was optimized for a given initial end-effector path. Moreover, even in extreme cases like a rectangular-shaped task with an unoptimized motion law, the algorithm provides significant improvements. During computation of these approaches, no rounding was done to avoid the accumulating error [74]. The resulted were rounded only for the purpose of better reading.

8. Conclusion

The typical workflow of an industrial robot consists of performing effective and supporting tasks. Even though effective tasks are often strictly defined, in many cases industrial processes allow some freedom of task execution. In this thesis, we proposed a way to formalize this freedom and to make use of it for optimizing both supporting and effective robot trajectories. Since the search space of the problem appeared to be very large, we proposed an approach that decomposes the problem into three components that can be applied in combination with each other or with other algorithms.

The problem Component1 uses a set of relaxed effective tasks as an input. Its goal is to find a sequence of relaxed effective tasks and a Cartesian entry point for each task such that the total Euclidean distance between the entry points is minimized. The problem Component1 was solved by modeling it as a Traveling Salesman Problem with Neighborhoods (TSPN). In order to solve the TSPN, it was decomposed into a Traveling Salesman Problem (TSP) and a Traveling-a-sequence-of-Polygons Problem (TPP). In contrast to state-of-the-art approaches, we proposed to solve the TPP for every iteration of a TSP solver. Two approaches were proposed to find a solution for the TSPN: Constricting Insertion Heuristic (CIH) and Constricting 3-Opt (C3-Opt). We evaluated these approaches on three sets of instances: (i) instances with known optimum, on which we presented a deviation from optimum, (ii) instances with "stretched" ellipses on which we demonstrated how well the heuristics react on more complicated areas, and (iii) instances for the Close-Enough TSP (CETSP), on which we compared the proposed heuristics with approaches for a related problem. Since the C3-Opt is a tour-improvement heuristic and its behavior depends on an initial tour, we evaluated it in combination with three tour construction algorithms: random tour generation, the nearest neighborhood algorithm and the CIH. The combination of the CIH and the C3-Opt appeared to be a trade-off between solution quality and run time. Both CIH and C3-Opt have an insignificant dependence on their parameters and repeatedly provide good results. The proposed heuristics are simple to implement, have

a short computational time and can be efficiently used to solve the TSPN and, as a consequence, sequencing of relaxed effective tasks.

In the problem Component2 a sequence of relaxed effective tasks is given. The goal is to find a C-space tour such that the cost of the corresponding C-space trajectories is minimized. In order to solve this problem, we modeled it as a TPP and solved the TPP, taking into account domain-specific knowledge like a robot-based cost. The existing methods for the TPP could not be applied directly because in this problem, we have to find a tour not for polygons, but rather for relaxed effective tasks that are multi-dimensional areas. The complexity of the problem grows even further, as every potential entry point leads to several possible inverse kinematics solutions, and the chosen inverse kinematics solution influences the global cost. Therefore, a solution should be chosen based on inverse kinematics solutions in other areas. The TPP with robot-based information is solved in multiple stages nested in each other where every outer stage depends on the optimization results of its inner stage. We used the Rubber-Band Algorithm (RBA) for optimization, but modified the local search techniques to extend the algorithm for this problem. We refer to this RBA extension as Nested RBA. The proposed approach was evaluated on scenarios from a cutting-deburring domain. The results showed that improving a task sequence adapted for a robot by searching for new entry points with our algorithm significantly reduces trajectory cost.

The result of solving the Component1 and the Component2 sequentially or in combination is a sequence of C-space entry points with the minimal cost of the corresponding supporting trajectory. The next step for solving the original problem of the thesis is to optimize every effective trajectory.

The problem Component3 takes a relaxed effective task and a C-space entry point. Its goal is to find an end-effector path that leads to a minimal cost C-space trajectory. It is possible to model this problem as a TPP by discretizing the given relaxed effective tasks into a sequence of volumes. We proposed a heuristic that is based on the RBA. We refer to it as Smoothed RBA. It constricts volumes using a smooth curve instead of a polyline. We evaluated the Smoothed RBA on two scenarios: a scenario from the medical domain where a robot equipped with a C-arm performs a 3D-angiography and on a cutting-deburring scenario where a robot cuts differently shaped contours. The evaluation was made for two motion laws. The first one leads to a trapezoidal velocity profile and is “uncomfortable” for a robot to execute, i.e., it causes high jerks in a C-space trajectory. The second motion law was optimized to reduce the jerk. The proposed Smoothed RBA heuristic significantly reduced the cost of the C-space trajectory in both cases, regardless of whether the motion law was optimized or not.

Future work

Although the proposed approaches can significantly reduce the cost of a robot trajectory, there is always room for future work. There are certain factors that motivate future work: (i) improving solution quality, (ii) extending the areas of application, and (iii) reducing computational time.

Improving solution quality

In this thesis, we decomposed effective and supporting movements. We assumed that a robot's tool makes a full stop in the entry point of a relaxed effective task. In reality, it is possible that a robot does not make a full stop but continues performing an effective task using the inertia of supporting movements. For example, after reaching an entry point an end-effector with a knife can smoothly start cutting without stopping, similar to trajectory of an airplane where it touches a runway while landing. Therefore, simultaneous planning of effective and supporting tasks potentially reduces cost.

For closed-contour tasks, we assumed that the task must be performed without interruption. However, in reality it is sometimes possible to interrupt the performance of a task in order to execute another one, and to finish the first task later. Such paths have better cost (see example in Figure 8.1). Nevertheless, this significantly increases the search space, as task break points have to be considered. Since such situations are domain specific, the feature of task interrupting was omitted in this thesis.

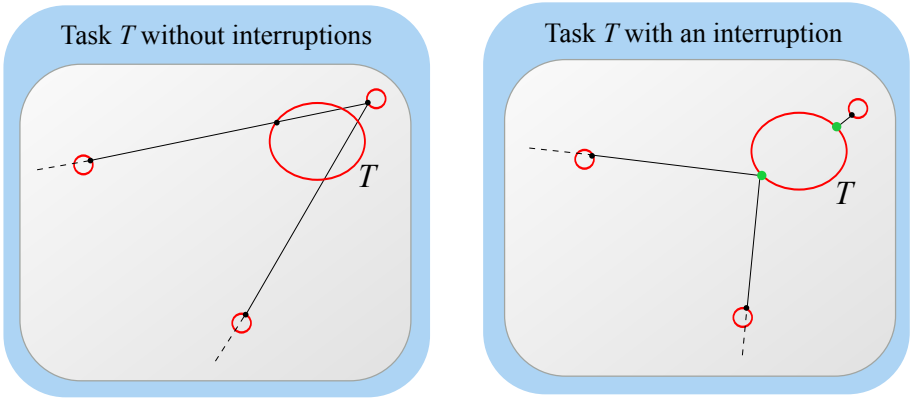

Figure 8.1: The distance of a supporting path (black) on the left picture is 50% longer than on the right. Effective tasks are depicted in red. Points of task interruption are depicted in green.

Component1 and Component2: In this thesis, the position of a robot's base was considered constant. However, the location of a robot's base greatly influences the cost of a C-space trajectory obtained for an end-effector path. In many applications, the location of a robot's base is not important, or at least can vary within a certain area. A robot's base can also be mounted on a moving platform, e.g., Kuka YouBot [20]. A possible future direction is to involve this degree of freedom into the planning process.

Component3: One way to achieve better results is to generalize the problem further by relaxing a motion law, as in the current problem formulation it is considered to be given and fixed.

Extending the areas of application

Currently, we assumed that there are no precedence constraints between the tasks. Nonetheless, there are industrial applications where a certain task has to be accomplished before another one. It was motivated by [34] who illustrated such constraint

with a case where a robot has to pickup items from a conveyor and drill them afterwards. Adding partial order constraints into the planning process would give the programmer better control of the optimization process.

We assumed that industrial environments are modeled in such a way that they do not contain unnecessary obstacles. In this case, collision avoidance can be excluded from the path planning process. Nonetheless, in complex environments, collision-free planning is required, e.g., when a robot needs to perform spot-welding inside a car frame or in the presence of other robots. In that case, the space where a robot can move is highly limited. Involving a collision-free planner in the planning would increase the application domain of the proposed approaches.

Reducing computational time

Currently in Component3 we choose evenly spread via-points and perform optimization sequentially for each of the via-points inside the corresponding neighborhood. However, changing the locations of two different via-points has different influences on the cost of a robot trajectory. A possible direction for future work is to optimize first positions of those via-points that decrease a trajectory cost the most. For example, for a rectangular task, first the points located in the corners are preferred. However, it is not so trivial because moving a corner point increases the distances to the neighboring via-points in a path, thus increasing the jerk. Therefore, it might be a good idea to optimize the certain tuples of via-points, e.g., a point in a corner, two points before and two points after the corner point for a rectangular task.

9. Appendix A
Evaluation Results of **Component3**

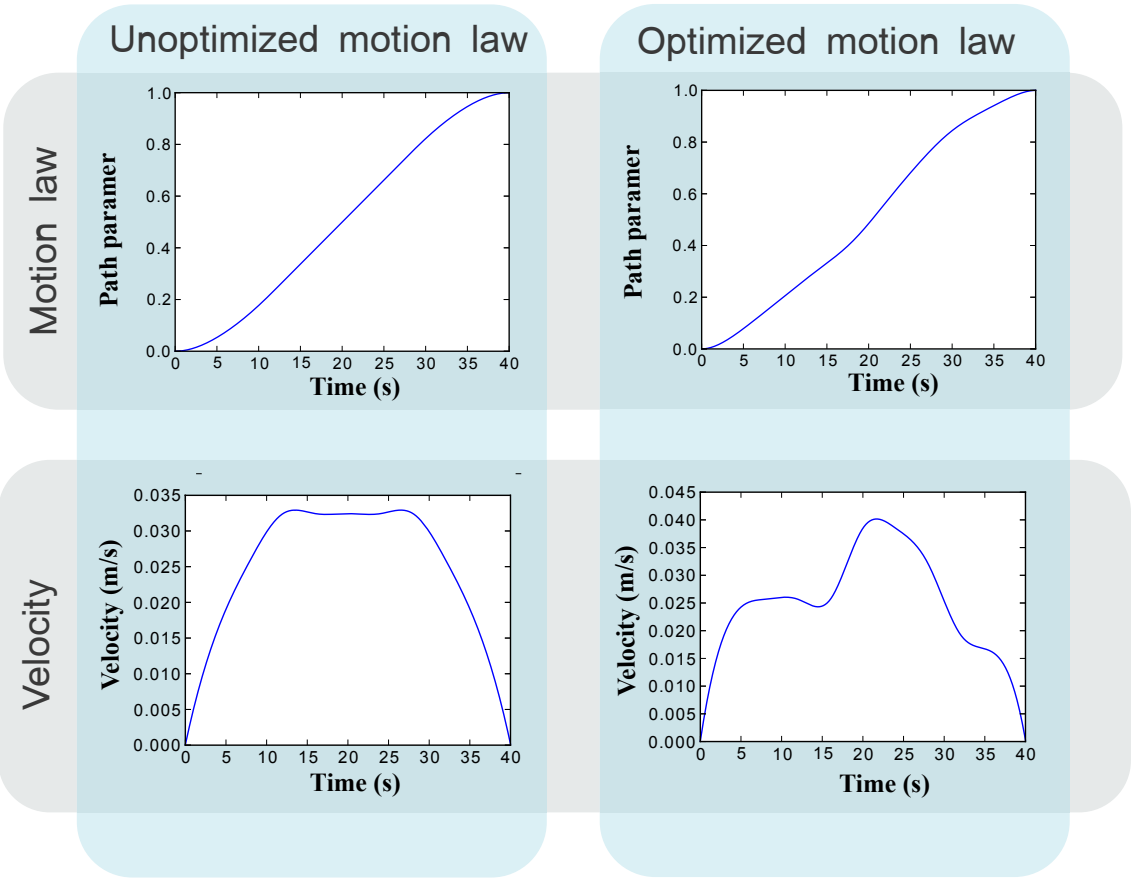

Figure 9.1: Rectangular task: motion law optimization

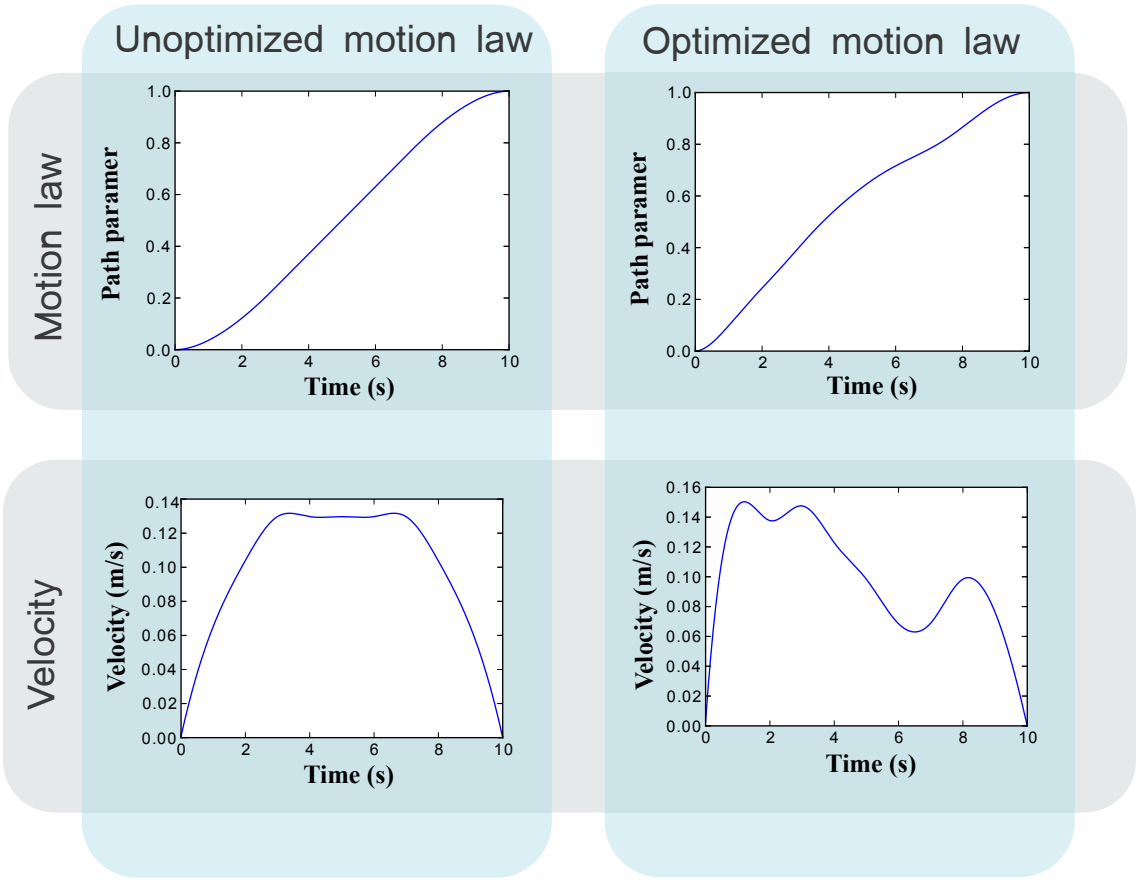

Figure 9.2: Circle-shaped task: motion law optimization

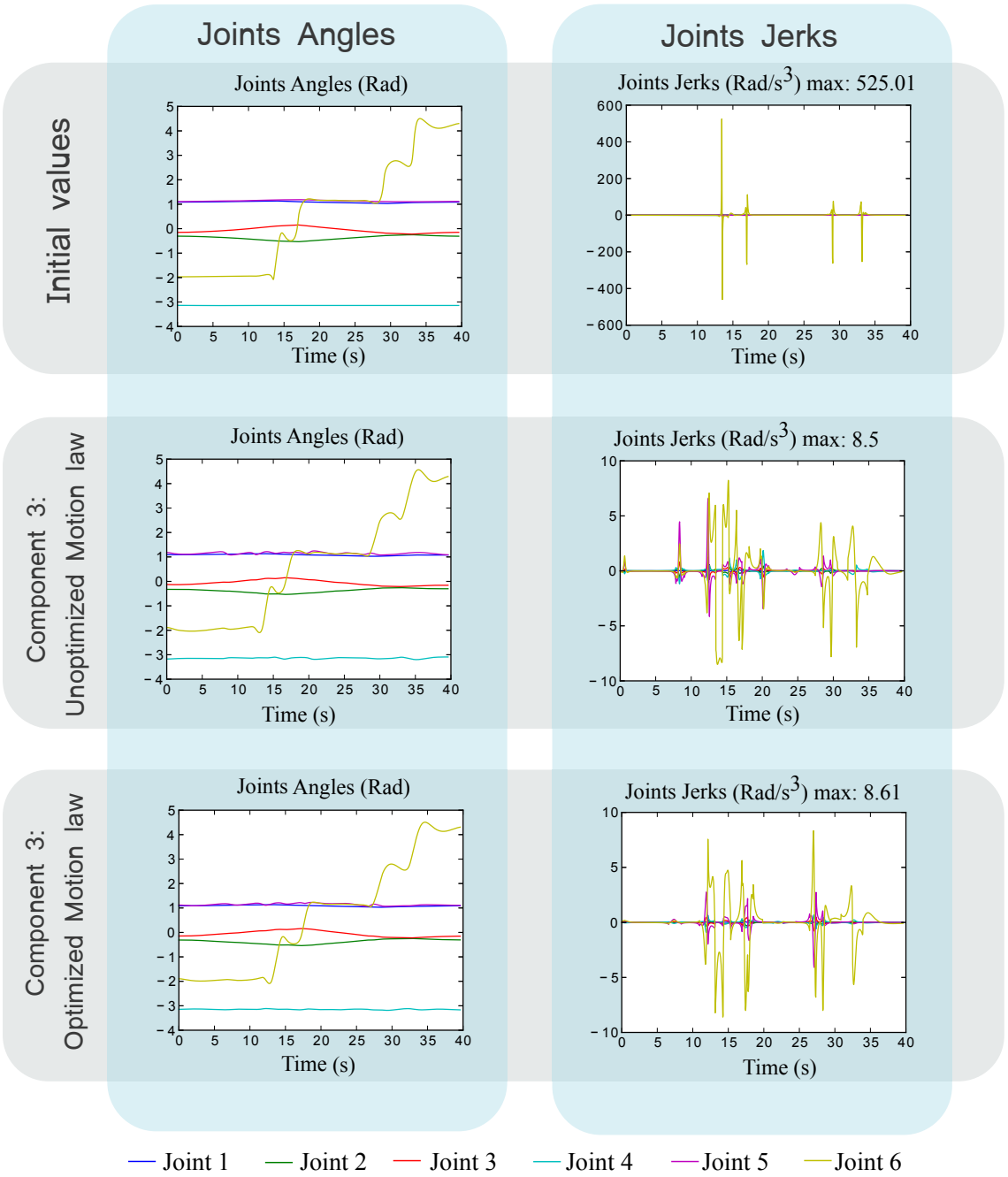

Figure 9.3: Results achieved by the Smoothed RBA from Component3 on the instance with a rectangle-shaped task.

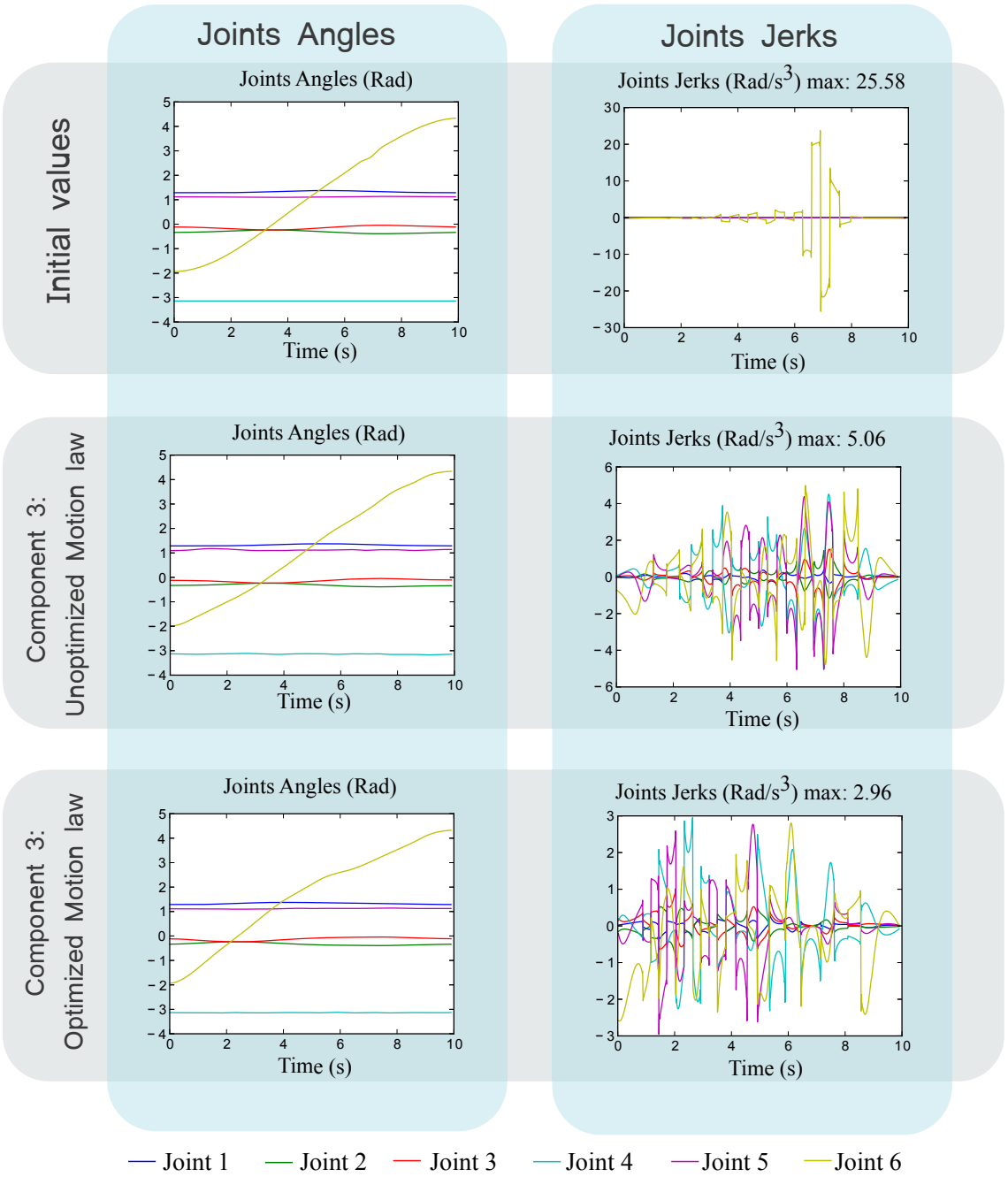

Figure 9.4: Results achieved by the Smoothed RBA from Component3 on the instance with a circle-shaped task

Bibliography

[1] L. L. Abdel-Malek and Z. Li. Robot location for minimum cycle time. *Engineering Costs and Production Economics*, 17(1):29–34, 1989. (cited on Page 29 and 30)

[2] L. L. Abdel-Malek and Z. Li. The application of inverse kinematics in the optimum sequencing of robot tasks. *The International Journal Of Production Research*, 28 (1):75–90, 1990. (cited on Page 29 and 30)

[3] S. Alatartsev and F. Ortmeier. Path planning for industrial robots among multiple under-specified tasks. In *Magdeburger-Informatik-Tage 2. Doktorandentagung*, 2013. (cited on Page 4 and 5)

[4] S. Alatartsev and F. Ortmeier. Improving the sequence of robotic tasks with freedom of execution. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014. (cited on Page 4, 5, and 68)

[5] S. Alatartsev, M. Güdemann, and F. Ortmeier. Trajectory description conception for industrial robots. In *German Conference on Robotics (ROBOTIK)*, pages 365–370, 2012. (cited on Page 4 and 5)

[6] S. Alatartsev, M. Augustine, and F. Ortmeier. Constricting insertion heuristic for traveling salesman problem with neighborhoods. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2013. (cited on Page 4 and 5)

[7] S. Alatartsev, V. Mersheeva, M. Augustine, and F. Ortmeier. On optimizing a sequence of robotic tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013. (cited on Page 4, 5, and 27)

[8] S. Alatartsev, A. Belov, M. Nykolaychuk, and F. Ortmeier. Robot trajectory optimization for the relaxed end-effector path. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2014. (cited on Page 4 and 5)

[9] S. Alatartsev, S. Stellmacher, and F. Ortmeier. Robotic task sequencing problem: A survey. *Journal of Intelligent and Robotic Systems*, 2015. (cited on Page 4)

[10] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2007. (cited on Page 25 and 40)

[11] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218, 1995. (cited on Page 27 and 41)

[12] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 97:43–69, 2003. (cited on Page 41)

[13] N. Aspragathos. Cartesian trajectory generation under bounded position deviation. *Mechanism and machine theory*, 33(6), 1998. (cited on Page 68)

[14] A. A. Ata. Optimal trajectory planning of manipulators: A review. *Journal of Engineering Science and Technology*, 1:32, 2007. (cited on Page 35)

[15] A. A. Ata and T. R. Myo. Optimal point-to-point trajectory tracking of redundant manipulators using generalized pattern search. *International Journal of Advanced Robotic Systems*, 2(3), 2005. (cited on Page 36)

[16] K. Baizid, R. Chellali, A. Yousnadj, A. Meddahi, and T. Bentaleb. Genetic algorithms based method for time optimization in robotized site. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1359–1364, 2010. (cited on Page 18, 29, and 30)

[17] D. Berenson, S. Srinivasa, and J. Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research (IJRR)*, 30(12):1435–1460, October 2011. (cited on Page 55 and 56)

[18] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer Publishing Company, Incorporated, 2008. (cited on Page 9)

[19] G. Biggs and B. MacDonald. A survey of robot programming systems. In *Australasian Conference on Robotics and Automation*, 2003. (cited on Page 2 and 22)

[20] R. Bischoff, U. Huggenberger, and E. Prassler. Kuka youbot-a mobile manipulator for research and education. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. (cited on Page 99)

[21] F. Bock. An algorithm for solving traveling-salesman and related network optimization problems. In *Unpublished manuscript associated with a talk presented at the 14th ORSA National Meeting*, 1958. (cited on Page 43)

[22] W. Bu, Z. Liu, and J. Tan. Industrial robot layout based on operation sequence optimisation. *International Journal of Production Research*, 41:4125–4145, 2009. (cited on Page 29 and 33)

[23] M. Cefalo, G. Oriolo, and M. Vendittelli. Planning safe cyclic motions under repetitive task constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013. (cited on Page 56)

[24] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi. Minimum cost trajectory planning for industrial robots. *European Journal of Mechanics-A/Solids*, 23(4): 703–715, 2004. (cited on Page 36, 90, and 94)

[25] S. Y. Chien, L. Q. Xue, and M. Palakal. Task planning for a mobile robot in an indoor environment using object-oriented domain information. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 27(6):1007–1016, 1997. (cited on Page 24)

[26] Y. Chien, A. Hudli, and M. Palakal. Using many-sorted logic in the object-oriented data model for fast robot task planning. *Journal of Intelligent and Robotic Systems*, 23(1):1–25, 1998. (cited on Page 24)

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001. (cited on Page 32)

[28] J. Craig. *Introduction to robotics: mechanics and control*. Pearson, 2005. (cited on Page 9 and 36)

[29] A. Croes. A method for solving traveling salesman problems. *Operations Research*, 5:791–812, 1958. (cited on Page 86)

[30] M. W. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah. *Throughput Optimization in Robotic Cells*, volume 101 of *International Series in Operations Research & Management Science*. Springer US, 2007. (cited on Page 23)

[31] R. Diankov and J. Kuffner. OpenRAVE: a planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA, July 2008. (cited on Page 84)

[32] N. S. F. Doria, E. O. Freire, and J. C. Basilio. An algorithm inspired by the deterministic annealing approach to avoid local minima in artificial potential fields. In *International Conference on Advanced Robotics (ICAR)*, pages 1–6, 2013. (cited on Page 24)

[33] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Annual ACM symposium on Theory of Computing*, pages 473–482. ACM Press, 2003. (cited on Page 27 and 42)

[34] S. Dubowsky and T. Blubaugh. Planning time-optimal robotic manipulator motions and work places for point-to-point tasks. *IEEE Transactions on Robotics and Automation*, 5:377–381, 1989. (cited on Page 18, 28, 29, and 99)

[35] Y. Edan, T. Flash, U. M. Peiper, I. Shmulevich, and Y. Sarig. Near-minimum-time task planning for fruit-picking robots. *IEEE Transactions on Robotics and Automation*, 7(1):48–56, 1991. (cited on Page 29 and 30)

[36] K. M. Elbassioni, A. V. Fishkin, and R. Sitters. Approximation algorithms for the euclidean traveling salesman problem with discrete and continuous neighborhoods. *International Journal of Computational Geometry and Applications*, pages 173–193, 2009. (cited on Page 41)

[37] G. Erdős, Z. Kemény, A. Kovács, and J. Váncza. Planning of remote laser welding processes. In *CIRP Conference on Manufacturing Systems*, 2013. (cited on Page 29, 35, and 37)

[38] J. Faigl, V. Vonásek, and L. Preucil. A multi-goal path planning for goal regions in the polygonal domain. In *European Conference on Mobile Robots (ECMR)*, 2011. (cited on Page 27, 29, 35, and 37)

[39] C. Fragkopoulos, K. Abbas, A. Eldeep, and A. Graeser. Comparison of sampling based motion planning algorithms specialized for robot manipulators. In *German Conference on Robotics (ROBOTIK)*, 2012. (cited on Page 25)

[40] P. J. From, J. Gunnar, and J. T. Gravdahl. Optimal paint gun orientation in spray paint applications – experimental results. *IEEE Transactions on Automation Science and Engineering*, 8(2):438–442, 2011. (cited on Page 40, 56, and 69)

[41] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013. (cited on Page 11)

[42] A. Gasparetto and V. Zanotto. Optimal trajectory planning for industrial robots. *Advances in Engineering Software*, 41:548–556, 2010. (cited on Page 36)

[43] I. Gentilini, F. Margot, and K. Shimada. The travelling salesman problem with neighbourhoods: MINLP solution. *Optimization Methods and Software*, pages 1–15, 2011. (cited on Page 29, 34, 37, 40, 41, 48, 75, 76, 77, and 79)

[44] I. Gentilini, K. Nagamatsu, and K. Shimada. Cycle time based multi-goal path optimization for redundant robotic systems. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2013. (cited on Page 29, 35, and 37)

[45] F. Glover, G. Gutin, A. Yeo, and A. Zverovich. Construction heuristics for the asymmetric TSP. *European Journal of Operational Research*, 129:555–568, 2001. (cited on Page 25)

[46] L. B. Gueta, R. Chiba, J. Ota, T. Ueyama, and T. Arai. Coordinated motion control of a robot arm and a positioning table with arrangement of multiple goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2252–2258, 2008. (cited on Page 29, 32, and 35)

[47] L. B. Gueta, J. Cheng, R. Chiba, T. Arai, T. Ueyama, and J. Ota. Multiple-goal task realization utilizing redundant degrees of freedom of task and tool attachment optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1719, 2011. (cited on Page 29, 35, and 37)

[48] R. Hassin and A. Keinan. Greedy heuristics with regret, with application to the cheapest insertion algorithm for the TSP. *Operations Research Letters*, 36(2): 243–246, 2008. (cited on Page 43)

[49] K. Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000. (cited on Page 43)

[50] I. T. Hernádvölgyi. Solving the sequential ordering problem with automatically generated lower bounds. *Operations Research Proceedings*, pages 355–362, 2003. (cited on Page 43)

[51] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961. (cited on Page 45, 59, and 69)

[52] Y. Huang, L. B. Gueta, R. Chiba, T. Arai, T. Ueyama, and J. Ota. Selection of manipulator system for multiple-goal task by evaluating task completion time and cost with computational time constraints. *Advanced Robotics*, 27(4):233–245, 2013. (cited on Page 29, 33, 55, and 65)

[53] P. Jiménez. Survey on model-based manipulation planning of deformable objects. *Robotics and computer-integrated manufacturing*, 28(2):154–163, 2012. (cited on Page 24)

[54] D. S. Johnson and L. A. McGeoch. *Local search in combinatorial optimization*. John Wiley and Sons, London, 1997. (cited on Page 41 and 43)

[55] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 566–580, 1996. (cited on Page 24)

[56] K. Klasing, D. Wollherr, and M. Buss. Joint dominance coefficients: A sensitivity-based measure for ranking robotic degrees of freedom. In *Advances in Robotics Research*, pages 1–10. Springer Berlin Heidelberg, 2009. (cited on Page 85)

[57] E. Kolakowska, S. F. Smith, and M. Kristiansen. Constraint optimization model of a scheduling problem for a robotic arm in automatic systems. *Robotics and Autonomous Systems*, 62(2):267–280, 2014. (cited on Page 11, 27, 29, and 31)

[58] J. Z. Kolter and A. Y. Ng. Task-space trajectories via cubic spline optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009. (cited on Page 69)

[59] A. Kovács. Task sequencing for remote laserwelding in the automotive industry. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2013. (cited on Page 27, 28, 29, 34, 35, 37, 40, 68, and 91)

[60] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000. (cited on Page 22 and 24)

[61] L. Lattanzi and C. Cristalli. An efficient motion planning algorithm for robot multi-goal tasks. In *IEEE International Symposium on Industrial Electronics (ISIE)*, 2013. (cited on Page 29 and 34)

[62] S. LaValle. *Planning algorithms*. Cambridge University Press, 2006. (cited on Page 8, 24, and 25)

[63] S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44:2245–2269, 1965. (cited on Page 43)

[64] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963. (cited on Page 30)

[65] H. Liu, X. Lai, and W. Wu. Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints. *Robotics and Computer-Integrated Manufacturing*, 29(2):309–317, 2013. (cited on Page 36)

[66] A. Loredo-Flores, E. J. González-Galván, J. J. Cervantes-Sánchez, and A. Martinez-Soto. Optimization of industrial, vision-based, intuitively generated robot point-allocating tasks using genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(4):600–608, 2008. (cited on Page 29 and 31)

[67] S. Macfarlane and E. A. Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 19(1):42–52, 2003. (cited on Page 72)

[68] O. Maimon. The robot task-sequencing planning problem. *Robotics and Automation, IEEE Transactions on*, 6(6):760–765, 1990. (cited on Page 23 and 24)

[69] N. Mansard and F. Chaumette. Task sequencing for high-level sensor-based control. *IEEE Transactions on Robotics*, 23(1):60–72, 2007. (cited on Page 24)

[70] A. A. Masoud. A harmonic potential approach for simultaneous planning and control of a generic UAV platform. *Journal of Intelligent & Robotic Systems*, 65 (1-4):153–173, 2012. (cited on Page 24)

[71] W. Mennell. *Heuristics for solving three routing problems: close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem*. PhD thesis, University of Maryland, 2009. (cited on Page 27, 28, 41, 76, 81, and 82)

[72] V. Mersheeva and G. Friedrich. Routing for continuous monitoring by multiple micro UAVs in disaster scenarios. In *European Conference on Artificial Intelligence (ECAI)*, pages 588–593, 2012. (cited on Page 43)

[73] J. S. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *Annual symposium on Computational geometry (SoCG)*, pages 183–191, 2010. (cited on Page 41)

[74] R. H. Mole. The curse of unintended rounding error: a case from the vehicle scheduling literature. *Journal of the Operational Research Society*, pages 607–613, 1983. (cited on Page 95)

[75] R. Montemanni, D. H. Smith, and L. M. Gambardella. A heuristic manipulation technique for the sequential ordering problem. *Computers & Operations Research*, 35(12):3931 – 3944, 2008. (cited on Page 25)

[76] P. Oberlin, S. Rathinam, and S. Darbha. Today's traveling salesman problem. *IEEE Robotics & Automation Magazine*, 17(4):70–77, 2010. (cited on Page 42)

[77] A. Olabi, R. Béarée, O. Gibaru, and M. Damak. Feedrate planning for machining with industrial six-axis robots. *Control Engineering Practice*, 18(5):471–482, 2010. (cited on Page 36)

[78] X. Pan, F. Li, and R. Klette. Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons. In *Canadian Conference on Computational Geometry*, pages 175–178, 2010. (cited on Page 44, 59, and 69)

[79] Z. Pan, J. Polden, N. Larkin, S. V. Duin, and J. Norrish. Recent progress on programming methods for industrial robots. In *Proceedings for the joint conference of 41st Internationel Symposium on Robotics and 6th German Conference on Robotics*, 2010. (cited on Page 2 and 22)

[80] Z. Pan, J. Polden, N. Larkin, S. Duin, and J. Norrish. Automated offline programming for robotic welding system with high degree of freedoms. In Y. Wu, editor, *Advances in Computer, Communication, Control and Automation*, volume 121 of *Lecture Notes in Electrical Engineering*, pages 685–692. Springer Berlin Heidelberg, 2012. (cited on Page 41)

[81] J.-F. Petiot, P. Chedmail, and J.-Y. Hascoët. Contribution to the scheduling of trajectories in robotics. *Robotics and Computer-Integrated Manufacturing*, 14(3): 237–251, 1998. (cited on Page 29, 31, and 32)

[82] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In *Technological Innovation for Sustainability*, pages 139–146. Springer, 2011. (cited on Page 23)

[83] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes – the art of scientific computing*. Cambridge: Cambridge University Press, 3rd edition, 2007. (cited on Page 45)

[84] C. Rego and F. Glover. *The traveling salesman problem and Its variations*, chapter Local search and metaheuristics, pages 309–368. Kluwer Academic Publishers, 2002. (cited on Page 44)

[85] G. Reinhart, U. Munzert, and W. Vogl. A programming system for robot-based remote-laser-welding with conventional optics. *CIRP Annals-Manufacturing Technology*, 57:37 – 40, 2008. (cited on Page 18, 29, and 31)

[86] M. Saha, G. Sánchez-Ante, and J.-C. Latombe. Planning multi-goal tours for robot arms. In *International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3797–3803, 2003. (cited on Page 29 and 32)

[87] M. Saha, T. Roughgarden, J.-C. Latombe, and G. Sánchez-Ante. Planning tours of robotic arms among partitioned goals. *Robotics Research*, 25:207 – 223, 2006. (cited on Page 29 and 32)

[88] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. X. Wang. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, 103(5):169–176, 2007. (cited on Page 42)

[89] B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer Verlag, Berlin, 2008. (cited on Page 22)

[90] D. Simon. The application of neural networks to optimal robot trajectory planning. *Robotics and Autonomous Systems*, 11:23–34, 1993. (cited on Page 90)

[91] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic. Dual arm manipulation – a survey. *Robotics and Autonomous Systems*, 60(10):1340–1353, 2012. (cited on Page 24)

[92] T. B. Smith, J. Barreiro, D. E. Smith, V. SunSpiral, and D. Chavez-Clemente. ATHLETEs feet: multi-resolution planning for a hexapod robot. In *ICAPS: Scheduling and Planning Applications Workshop (SPARK)*, 2008. (cited on Page 55 and 65)

[93] S. N. Spitz and A. A. G. Requicha. Multiple-goals path planning for coordinate measuring machines. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2322–2327, 2000. (cited on Page 29 and 32)

[94] S. Srivastava, S. Kumar, R. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. In *CORSE Journal*, volume 7, pages 97–101, 1969. (cited on Page 25)

[95] M. Stilman. Task constrained motion planning in robot joint space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007. (cited on Page 55 and 56)

[96] K. Vicencio, B. Davis, and I. Gentilini. Multi-goal path planning based on the generalized traveling salesman problem with neighborhoods. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014. (cited on Page 27, 29, 34, and 37)

[97] C. Wurll and D. Henrich. Point-to-point and multi-goal path planning for industrial robots. *Journal of Robotic Systems*, 18(8):445–461, 2001. (cited on Page 27, 29, and 32)

[98] C. Wurll, D. Henrich, and H. Wörn. Multi-Goal Path Planning for Industrial Robots. In *International Conference on Robotics and Application*, 1999. (cited on Page 25, 27, 29, and 31)

[99] E. K. Xidias, P. T. Zacharia, and N. A. Aspragathos. Time-optimal task scheduling for articulated manipulators in environments cluttered with obstacles. In *Robotica*, volume 28, pages 427–440. Cambridge Univ Press, 2010. (cited on Page 29 and 33)

[100] H. Yang and H. Shao. Distortion-oriented welding path optimization based on elastic net method and genetic algorithm. *Journal of Materials Processing Technology*, 209(9):4407–4412, 2009. (cited on Page 18, 28, 29, and 31)

[101] Z. Yao and K. Gupta. Path planning with general end-effector constraints. *Robotics and Autonomous Systems*, 55:316–327, April 2007. (cited on Page 55)

[102] P. Zacharia, E. Xidias, and N. Aspragathos. Task scheduling and motion planning for an industrial manipulator. *Robotics and Computer-Integrated Manufacturing*, 29:449 – 462, 2013. (cited on Page 18, 27, 29, and 33)

[103] P. T. Zacharia and N. A. Aspragathos. Optimal robot task scheduling based on genetic algorithms. *Robotics and Computer-Integrated Manufacturing*, 21:67–79, 2005. (cited on Page 27, 29, 30, and 33)

[104] K. Zhang, E. G. Collins Jr, and D. Shi. Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(2):21, 2012. (cited on Page 23)