STUTTGARTER BEITRÄGE ZUR PRODUKTIONSFORSCHUNG

FELIX BEUKE

Online Motion Planning for Dual Arm Industrial Robots

Universität Stuttgart

Fraunhofer IPA

STUTTGARTER BEITRÄGE ZUR PRODUKTIONSFORSCHUNG BAND 98

Herausgeber:
Univ.-Prof. Dr.-Ing. Thomas Bauernhansl
Univ.-Prof. Dr.-Ing. Kai Peter Birke
Univ.-Prof. Dr.-Ing. Marco Huber
Univ.-Prof. Dr.-Ing. Oliver Riedel
Univ.-Prof. Dr.-Ing. Dipl.-Kfm. Alexander Sauer
Univ.-Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl
Univ.-Prof. a. D. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper

Felix Beuke

Online Motion Planning for Dual Arm Industrial Robots

FRAUNHOFER VERLAG

Kontaktadresse:
Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 07 11/9 70-11 01
info@ipa.fraunhofer.de; www.ipa.fraunhofer.de

STUTTGARTER BEITRÄGE ZUR PRODUKTIONSFORSCHUNG

Titelbild: © chesky - stock.adobe.com

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über www.dnb.de abrufbar.
ISSN: 2195-2892
ISBN (Print): 978-3-8396-1561-4

D 93

Zugl.: Stuttgart, Univ., Diss., 2019

Druck: Mediendienstleistungen des Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart
Für den Druck des Buches wurde chlor- und säurefreies Papier verwendet.

Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 80 04 69, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 07 11 9 70-25 00
Telefax 07 11 9 70-25 08
E-Mail verlag@fraunhofer.de
URL http://verlag.fraunhofer.de

Online Motion Planning for Dual Arm Industrial Robots

Von der Fakultät Konstruktions-, Produktions- und Fahrzeugtechnik
der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

Vorgelegt von
Felix Beuke
aus Achim

Hauptberichter:	Univ.-Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl
Mitberichter:	Univ.-Prof. Dr.-Ing. Torsten Kröger
Tag der mündlichen Prüfung:	10.09.2019

Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen

2019

Kurzinhalt

Der Trend zu einer immer weitergehenden Individualisierung von Endkundenprodukten stellt die Domäne der Fertigungsautomatisierung vor eine Reihe von Herausforderungen. Eine dieser Herausforderungen betrifft die notwendige Flexibilität der eingesetzten Fertigungsmittel. Konnte eine automatisierte Fertigungsanlage in herkömmlicher Fertigung noch auf mehrere Jahre im Dauerlauf hin entworfen, projektiert und finanziert werden, so erfordert die Fertigung kundenindividueller Produkte eine vielfach erhöhte Variantenvielfalt und auf diese angepasste, kurzzyklische und flexible Fertigungsprozesse. Mit den geforderten geringeren Einzelstückzahlen und kürzeren Produktlaufzeiten geht eine Unsicherheit einher, die den Einsatz von Investitionsmitteln für hochspezialisierte Anlagen zunehmend unrentabel macht. Die eingesetzten Fertigungsmittel müssen deshalb bei Änderungen oder einem Wechsel des gefertigten Produkts möglichst flexibel weiterverwendbar sein. Vor diesem Hintergrund hat die kollaborative Industrierobotik, bei der Roboter und Mensch auf engem Raum und ohne Schutzzaun zusammenarbeiten, in den letzten Jahren einen Boom erfahren. In diesem Rahmen wurde auch die Entwicklung zweiarmiger Roboter vorangetrieben. Geräte von diesem Typ vereinen zwei Roboterarme in humanoider Anordnung an einem Oberkörper und decken dadurch potenziell eine Palette von Fähigkeiten ab, die bisher nur bei menschlichen Mitarbeitern vorhanden ist (z.B. zweiarmiges Handling, schnellere Arbeitsausführung durch Parallelarbeit). Gleichzeitig bringt die enge Anordnung der beiden Arme Probleme bei der Geräteapplikation mit sich, die die Vorteile der gesteigerten Flexibilität in vielen Fällen wieder zunichtemachen. Mit herkömmlichen Programmierverfahren ist es aufgrund der nah aneinander liegen Arme komplex und zeitaufwendig, koordinierte und kollisionsfreie Bewegungen zu programmieren. Um die Applikation zweiarmiger Industrieroboter zu erleichtern und die flexible Anwendbarkeit derartiger Geräte zu sichern, entwickelt die vorliegende Arbeit einen Bewegungsplaner und eine zugehörige Steuerungsarchitektur für Zweiarmroboter. Gemeinsam entlasten diese den menschlichen Programmierer von der Pflicht, die Kollisionsfreiheit auf Basis einer exakt spezifizierten zeitlichen Synchronisation der Arme sicherzustellen und führen dadurch zu einer deutlichen Komplexitätsreduktion bei der Applikation zweiarmiger Industrieroboter. Darüber hinaus ermöglicht das entwickelte Steuerungskonzept den Einsatz automatisierter Planer auf höheren Abstraktionsebenen.

Short Summary

The trend toward consumer products of ever-increasing individuality poses several challenges for the domain of manufacturing automation. One of these challenges relates to the flexibility of devices used in industrial automation. In classical low-variability and high-throughput automation, plants could be designed and financed with several years of constant uptime in mind. In contrast, automated manufacturing of more and more individualized goods requires adapted manufacturing processes with shorter cycles and more flexible automation equipment. The higher variability and shorter life-cycle of manufactured goods often entails an uncertainty that makes investment in highly-specialized but inflexible automation equipment unprofitable. This creates a growing need for more flexible equipment that can be reused when the produced goods or parts of the production process change. Against this backdrop, the field of collaborative robotics has received increased interest and development activity over the past years. These robots can work in close proximity to humans and do not need to be fenced off in dedicated safety areas, thus enabling a more flexible use of the equipment. The idea of flexibility is taken further in this context by the development of dual-arm robots that mimic the morphology of the human upper body with two arms mounted on one common body. While these devices potentially offer increased flexibility and can work on a set of tasks otherwise reserved for humans (e.g., dual-arm handling, parallel work with both arms) the close proximity of the arms brings about new problems in the application of these robots. Using available programming methods for creating collision-free and coordinated motions for dual-arm robots is often complex and leads to long application times, which takes away a considerable part of the added flexibility introduced by dual-arm robots. To enable users to leverage the potential flexibility of industrial dual-arm robots more easily, this work develops a coordinated motion planner and associated control infrastructure for dual-arm industrial robots. Together, these components relieve the programmer of the responsibility to prevent arm collisions by specifying an exact temporal synchronization for both arms. Instead, collision-free and coordinated motions are automatically planned and executed based on a geometrical model. Moreover, it can serve as an application basis for more high-level planners for dual-arm industrial robots. The proposed planner and control architecture are implemented and evaluated on a real dual-arm robot.

Contents

List of Figures

List of Tables

1 Introduction

1.1 The factory: a high-security prison for robots

Whether it be a nightmare or a utopia, building robotic machines with ever more human-like features and capabilities has been a long-standing dream of humankind. Countless references in novels, movies and other works of art bear abundant testimony. However, as visionary as these dreams might be, reality has so far fallen short by a long stretch. Even with the amazing advances in the field of robotics over the past decades, including the very recent advances in cognitive robotics enabled by machine learning technologies, we are nowhere near the brink of building robots that are able to match humans or even animals in their intricate sensorimotor design, their ability to abstract, reason and learn or their resourcefulness in using communication and collaboration to solve problems that could not be tackled by an individual alone. The lack of even the most basic of these capabilities is particularly striking in the domain of industrial robotics. Here, mechanically highly stiff robotic manipulators exist in a world of sensory deprivation: they do not perceive their environment, they do not sense the forces they exert on it and they are far from doing any kind of reasoning about the tasks they are executing day and night. Because of this, strong safety fences bar them from interaction with the rest of the world, which they would otherwise put at danger. This type of robot arguably has nothing to do with human-like capabilities – in essence, it is merely a positioning system with the added mathematical complexity and added kinematic flexibility of multi-link revolute kinematic chains.

The necessity of freeing industrial robots from this sensory deprivation has been recognized as a key enabler in paving the way towards a future in which robots could some day be recognized by their fictional counterparts as one of their kind. In the past decades, the academic community has been prolific in researching new techniques that could be the building blocks of this future: motion planning, reasoning, human-like body control and motion learning to name just a few prominent topics. However, practically none of these new technologies have made their way into the industrial robots being installed in factories in ever larger numbers as these lines are written (Heer 2017). These devices continue to function much in the same way that industrial robots have been functioning for decades, tailored specifically to the task of executing over and

over again a taught-in sequence of motions that move parts from one well-defined position to the next (e.g., as in pick-and-place applications) or precisely and repeatably follow a well-defined geometry in the workspace (e.g., as in welding or gluing applications).

1.2 Motivation

1.2.1 Technological push

The main reason why new technologies have been adopted by industrial robotics manufacturers very reluctantly if at all is rather simple. For many years, there was simply no need. The tools and programming languages developed for industrial robots were apt at managing the comparably low complexity of the highly structured and predictable worlds in which robots are typically applied. For decades, there has been no considerable shift in the complexity of problems handled by robot applications. Even today, a large number of tasks within a manufacturing process are still considered non-serviceable by classical industrial manipulators. Especially tasks that require skills such as perception or a high level of dexterity fall into this category.

In recent years however, new concepts in industrial robotics have started to be actively explored beyond the limits of academic research as well. The appearance of Universal Robot's UR series and similar robots has made many people rethink the concept of a strict physical separation between robot and humans, leading to a huge trend often called *cobotics* (shorthand for collaborative robotics). The introduction of Bosch's APAS combined this idea with a departure from classical text-driven robot programming, instead favoring a GUI-based graphical approach to teaching robot applications. In the same way, the application of machine learning algorithms to industrial automation tasks such as motion learning for assembly or peg-in-hole tasks might lift the requirement of strict determinism that has so far been prevalent in industrial robotics. A further area which has seen recent progress in commercially available devices is the design of the kinematics itself. Lately, more and more devices have appeared on the industrial robot market which mimic a human-like upper body design by uniting two robot arms on a common body. This design enables robots to use the same re-grasping and dual-arm handling strategies that humans use, thus broadening the application scope. Prominent examples of this device class which are commercially available include the ABB YuMi, Rethink Robotics' Baxter as well as Kawada's NextAge and Yaskawa's Motoman.

The advent of these devices has broadened the scope of application by virtue of the human-like design. But at the same time, it has also brought about a new kind of complexity in the

programming and application of industrial robots, caused by the physical closeness of two manipulators. The tools that have emerged from classical robot programming have been designed with classical multi-robot setups in mind, which feature statically planned motions and only very small overlaps in manipulator workspaces. The new humanoid design of dual-arm kinematics obviously violates the assumptions underlying these principles, and it's therefore little surprise that conventional robot teaching and motion planning techniques fail to provide a solution that makes these systems easily usable and applicable to the new and broader automation scope they potentially cover.

1.2.2 Economical pull

Leaving the technical point of view aside for a second, there are also a number of economical factors supporting the technological shifts beginning to emerge in the domain of industrial robotics. Increasingly strong market dynamics and volatility lead to a higher uncertainty in manufacturing. As the speed of innovation increases, the time horizons for return on invest must decrease to keep up. Where this is no longer possible, the agility and flexibility must be transferred into the purchased production assets themselves in order to allow for full asset utilization independent of short-cycled decisions about production strategy (Kleiner et al. 2011; Makris et al. 2012). In industrial robotics specifically, the ultimate goal of development is to enable a truly flexible manufacturing design, in which the degree of automation can easily be varied and adapt to the needs of varying output demands after production has already started. In such a setup, a collaborative, humanoid dual-arm robot design is highly desirable because it allows humans and robots to work side by side and, to some degree, even interchangeably.

1.3 Problem definition

In the case of industrial robots, the transfer of intelligence from human programmer and operator into the device controller is highly challenging. The complexity of the problems that humans can solve seemingly easily for themselves is overwhelming. In an industrial production setting, most prominent among these challenges are

1. recognizing parts of interest, obstacles and determining their positions,
2. conceiving high-level behaviors directed at fulfilling a given task and
3. computing and robustly executing low-level behavior to enact these high-level strategies.

These groups of problems are ordered by decreasing degree of abstraction, yet whichever the degree of abstraction, a whole world of complexity unfolds behind any one of these items. This work will focus on a subproblem from the third group arising specifically in the context of dual-arm industrial robotics: the problem of arm coordination. The close proximity of the arms allows the robot to act according to human-like motion patterns, e.g. grasping objects with two hands, passing or re-grasping objects between hands, working asynchronously on geometrically non-separated tasks (e.g., grasping parts from a common pallet). However, the proximity also brings about new challenges. When two robot arms move concurrently but independently in the same workspace, there is a high risk of collision if measures are not implemented on a higher level to avoid this problem. The individual arms contained in today's available dual-arm robot systems today are essentially just that: two single manipulators mounted in a human-inspired fashion on a common body, the robot torso. Any kind of higher-level control structure uniting these two arms into a unified system is absent from robot today's controllers. Instead, the responsibility and thus the intelligence for resolving this collision risk lies with the human programmer. This situation is undesirable for several reasons:

Non-expressive language for coordination The programming tools that have been developed with conventional multi-robot setups in mind rely solely on time and sequence as concepts for arm coordination and collision resolution. However, it is not always sensible or feasible to specify a fixed temporal relation between motions when only collision avoidance is needed, not strict temporal synchronization.

Static coordination Using currently available tools, the robot programmer establishes a static coupling between the arms' motions at programming time. While it is possible to branch between a number of predefined static couplings at runtime, it is impossible to alter this coupling, e.g. in order to react to non-determinisms in the execution time of robot actions. Instead, the flexibility of an individual arm is constrained, because for any desired motion the corresponding motion for the other arm has to be known at programming time as well to ensure non-colliding motions.

Use-case specific solutions The coordination tools available today force the programmer to build code that implicitly contains reasoning about the geometry of the specific application. A typical decision could look as follows:

```
if (Robot.Left.Position.x < threshold)
  Robot.Left.Move P1;
else
  Robot.Left.Move P2;
```

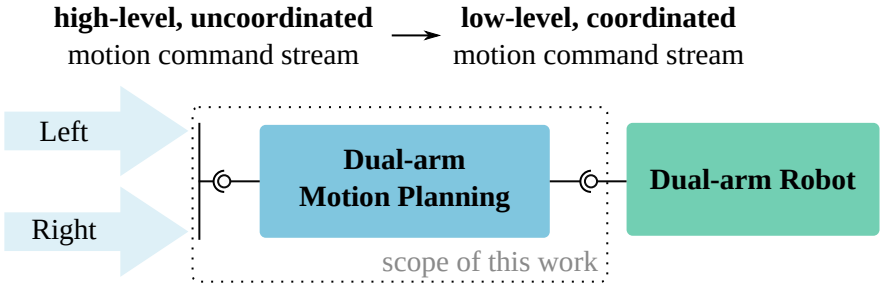

Figure 1.1: High-level sketch of the system under consideration

Implicitly representing geometry-based solutions in code like this is not only error-prone and hard to validate and maintain, but it is also application specific. Depending on the design, slight changes in the setup might be very hard to reflect in the code, making this solution very inflexible and not reusable.

Complex to understand For every new geometrical setup and application logic, the robot programmer has to reason not only about the logic of the task to be completed, but also about the exact geometry of the manipulator motions and their relative position with respect to each other. This makes the programming of dual-arm robot applications much more complex than the programming of conventional single-arm robots and requires expert knowledge and it raises the entry threshold and therefore also the scope of practical applicability for dual-arm robots.

Time-consuming Designing, implementing and validating arm coordination solutions based on the available tools is time-consuming and therefore limits the flexible applicability of dual-arm robots in a volatile production environment.

1.4 Solution proposal and contribution

To address these shortcomings, the goal of the work presented in this thesis is to develop an interface at the system level. This contrasts with the level of individual arms targeted by the available techniques. Figure 1.1 shows a high-level abstraction of this idea. Given a dual-arm robot system RS and an input stream of abstract, high-level motion commands MC for the individual arms, the solution developed in this work shall issue low-level motion commands to the robot controller that will move the robot arms to the desired positions while avoiding collisions between the arms or between an arm and the environment. The following overview gives a more precise definition of the terms used in this short description.

- **Robot System** Robot systems considered in this work consist of two separate and kinematically independent chains ("arms") consisting of revolute or translational joints. The arm positions are assumed to be fixed with respect to each other and the environment.
- **Motion Command Input Stream** As defined in this context, a motion command is a request to move one or both arms in the robot system to a desired pose. Such a command can either be asynchronous (requesting one arm to move regardless of the motion of the other arm) or synchronous (requesting a motion of both arms at the same time). The motion command input stream consists of one to many such commands generated at any time. The input stream is assumed to be generated by any kind of higher-level component. One possible implementation could interpret a hard-coded series of motion commands given by a human-designed motion script. However, commands might just as well be generated at runtime by a high-level behavioral planner which has access to both robot task and current state of the world.
- **Robot Interface** There is a variety of interfaces offered by typical dual-arm industrial robots. The low-level motion commands generated by the developed solution must target one of these interfaces.

A more detailed analysis of the constraints and requirements will be given in Section 4.2.

1.5 Outline

This thesis is divided into six chapters. After this brief introduction of this work's topic, the second chapter will introduce the most important mathematical concepts in robot motion planning. The following third chapter is dedicated to surveying the state of the art in the three relevant areas of industrial robot programming, manipulator path planning and automatic arm coordination algorithm. Chapter four will then provide an in-depth discussion of the dual-arm motion planning and control system introduced in this work. Evaluation results for the implemented demonstrator system will be given in the fifth chapter before a final conclusion and an outlook on future research directions concludes this thesis in chapter six.

2 Foundations

After the high-level introduction of the topic in the previous chapter, this second chapter will describe a number of foundational concepts in motion planning for industrial robot arms. It will introduce mathematical representations of robots, motions and operations commonly used in motion planning for kinematic chains. Further, the concepts of workspace and configuration space will be introduced together with the transformations between these spaces.

2.1 Kinematics

The very essence of robot design is centered around motion – manipulators are built to interact with their environment through motion, and large amounts of work have gone and are still going into computing the best motions for a given robotic system (Choi et al. 2017). Modeling these motions is usually split into the two parts kinematics and dynamics. The study of kinematics is concerned with describing the motions of a robot system while leaving its dynamics out of the picture. It is therefore suitable to describe the purely geometrical aspects of computing motion plans. This work focuses on motion planning for robotic manipulators. These typically consist of a series of rigid bodies linked together by revolute joints to form a serial kinematic chain. The following section will outline how these structures are modeled and give an introduction to important algorithms used in kinematics calculations for this type of manipulator.

In the context of Euclidean spaces, the location of an object is defined by its pose. The information about the object location contained in a pose can be divided into two categories. Three translational degrees of freedom define the position (or displacement) of a body, while a further three degrees of freedom describe the orientation and rotation of a body. Since there are six degrees of freedom in Cartesian space, any representation for a pose needs to comprise at least six coordinates. Poses of bodies are always described with respect to the pose of some reference frame r_w. Any such frame is described by the position of its origin O and a set of three vectors $\vec{x}, \vec{y}, \vec{z}$ forming an orthonormal basis for the space. The geometry of the object itself is also described with respect to some such frame r_b. Therefore, giving an object's pose is really giving the pose of the object's frame r_b with respect to the reference frame r_w. The

reference frame r_w is not special in and of itself. Rather, any frame that is static with respect to the observer of the scene can be used as reference frame.

2.1.1 Representing position

The 3-DOF translational part of a pose is called position. Representing it is straightforward. It can be given by a 3x1 vector

$$\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{2.1}$$

denoting the individual displacements p_x, p_y and p_z along the three axes defined by the base vectors $\vec{x}, \vec{y}, \vec{z}$ respectively and starting at the origin O of the reference frame. With the position defined in this way, it becomes obvious that the mathematical notation for specifying a position coincides with the representation of a displacement – the two concepts are interchangeable, because a position is defined in terms of a reference position (in this case, $\vec{O}$) plus a translation t_p. We thus have

$$p_i = \vec{O} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}. \tag{2.2}$$

It is notable that in a translation (i.e., change of object's position), all points on the body change their position with respect to the frame of reference. This feature sets translation apart from rotation, in which at least one point remains in its original position.

2.1.2 Representing rotation

While the description of position and translation is straightforward, representations of orientation and rotation take more diverse forms, each of which have their own advantages and disadvantages. As in the description of the positional part of the pose, all these representations can be used to specify orientation as well as rotation, i.e., change in orientation. In the domain of robotics, the most commonly used representations are (i) rotation matrices and (ii) Euler angles (Siciliano et al. 2016). Both of these representations will be described in the following sections.

One way of expressing the orientation of an object frame r_o with respect to a fixed frame of reference r_w (also, inertial frame) is to express the basis of r_o in terms of the basis of r_w. Doing this yields the so-called rotation matrix

$$ {}_jR_i = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix}, \tag{2.3} $$

in which each column is again the vector of a (now rotated) orthonormal basis expressed in terms of the original basis vectors. Multiplication of ${}_jR_i$ with a position vector p_i defined with respect to the frame i yields a second position vector

$$ p_j = {}_jR_i\, p_i. \tag{2.4} $$

This vector is oriented within frame j exactly as the original vector p_i is oriented within frame i. Consequently, its orientation in the original frame i is changed.

Rotating an object (i.e., frame basis vector or position vector describing some geometric feature of an object) about the x axis of a reference frame through an angle θ can be expressed by the rotation matrix with the values

$$ R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}. \tag{2.5} $$

For rotations about the axes y and z, the respective matrices take the forms

$$ R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \; R_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.6} $$

A rotation matrix encodes the three degrees of freedom of an orientation or rotation through nine matrix values. Since it comprises three more parameters than strictly needed to describe a 6-DOF system, it is not a compact representation. The six auxiliary relationships between the parameters are derived from the requirement that the three column vectors forming the matrix must form an orthonormal basis. This means that they must be pairwise orthogonal and of unit length.

The inverse rotation is given by the inverted rotation matrix R^{-1}. Because the auxiliary interdependencies between the matrix values constrain the matrix to be orthogonal, the inverse of a rotation matrix is equal to its transpose, $R^{-1} = R^T$. Rotations given in the form of rotation

matrices can be chained together by simple matrix multiplication. For example, rotating an object about its x axis by an angle θ and then about its new $\hat{y}$ axis by an angle γ can be expressed by the rotation matrix

$$R_{x,\hat{y}} = R_x R_y = \begin{pmatrix} \cos\gamma & 0 & -\sin\gamma \\ -\sin\theta\sin\gamma & \cos\theta & -\sin\theta\cos\theta \\ \cos\theta\sin\gamma & \sin\theta & \cos\theta\cos\gamma \end{pmatrix}. \tag{2.7}$$

As matrix multiplication is not commutative, the order of chaining matters, and objects are rotated about the axes resulting from the chain of previously applied rotations.

2.1.3 Euler angles

This multiplication property is exploited by the concept of Euler angles, which defines a minimal representation of rotation using exactly three parameters. These parameters used in the Euler angle representation describe three subsequent rotations about different axes. As the order of rotation about the axes matters, a rotation specification for Euler angles always has to be accompanied by the appropriate convention detailing the axes which the rotations relate to. One common choice from among the twelve possible conventions is the Z-Y-X convention, in which the object is first rotated about the z axis of the fixed frame, then about the y axis of the frame resulting from the first rotation, and finally about the x axis of the frame resulting from the second rotation. No matter which convention is used, the Euler angles representation suffers from a singularity. This occurs whenever the last rotation happens to be carried out about the same axis as the first. For the Z-Y-X convention, this is the case when the rotation about the second axis equals 90°.

2.1.4 Homogeneous transformations

Since the position and orientation of a body can be modeled in a decoupled fashion, it is convenient in modeling to treat the two parts of the pose separately. However, when computing representations of pose and transformations in Euclidean space, it is often convenient to work with a combined representation that encodes position and orientation together and allows for a more succinct formulation. Such a representation can be obtained from (2.2) in conjunction with (2.4). Superposing the translation and rotation of a vector r into one equation yields a transformed vector

$$r' = Rr + t. \tag{2.8}$$

This equation can be rearranged into the form $v' = Tv$ to yield

$$\begin{pmatrix} r' \\ 1 \end{pmatrix} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} r \\ 1 \end{pmatrix}. \tag{2.9}$$

Here, T is the 4×4 homogeneous transformation matrix. It contains the 3×3 rotation matrix R from (2.4) and the 3×1 translation vector t from (2.2). The other elements consist of a 1×3 zero vector and a scalar of value one. To match the dimensions of the homogeneous transformation matrix, the position vector is extended by a fourth entry that also equals one. The introduction of constant elements with values of zero and one is needed to yield a compact matrix representation. However, the additional multiplications needed to compute the matrix multiplication make this representation less computationally efficient.

Using the homogeneous transformation representation, it is easy to build a composite transformation between the frames 0 and n built from several piecewise transformations as in

$${}_0T_n = {}_0T_1 \, {}_1T_2 \, \cdots \, {}_{n-1}T_n. \tag{2.10}$$

This is a very good fit for describing the composite kinematic transformation along a serial chain of bodies linked by joints, which is the typical model for manipulator robots. This type of composite transformation is described in more detail in the following section.

2.2 Kinematic chains

The manipulators considered in this work are typically built from a number of rigid bodies linked together by joints. There are many types of joints. However, the joint type most commonly found in robot manipulators are revolute joints and prismatic joints. Revolute joints enable the linked bodies to rotate with respect to each other about the fixed joint axis, while prismatic joints allow a 1-DOF relative translation of the bodies along the joint's axis. Several of these links can be chained together to form the model of a serial chain manipulator (see Figure 2.1).

The geometry of each of the bodies forming the kinematic chain is described in a local frame of reference r_i for the respective body with index i. Therefore, specifying the full pose of the robot is equivalent to specifying the poses of the frames of each link with respect to the inertial frame. In theory, the local frames of reference can be located anywhere on the bodies. However, particular choices for the position of the frame with respect to the link's axis allow for a more succinct and convenient description of the kinematic chain with less parameters.

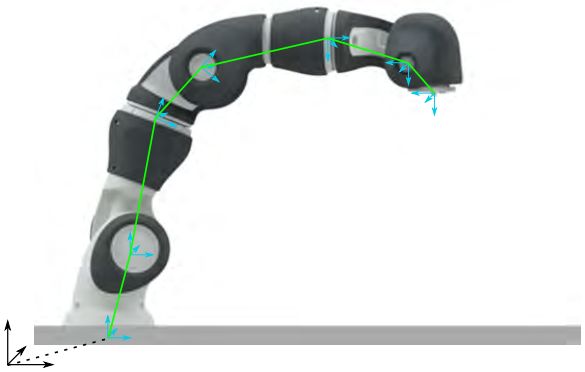

Figure 2.1: 7-DOF kinematic chain with revolute joints.

2.2.1 D-H parameters

There are different conventions governing the choice of reference frame placement. All of these are modifications of an original convention proposal by Denavit and Hartenberg (Denavit et al. 1955) which has subsequently come to be known by the name of Denavit-Hartenberg convention (or short, D-H convention). Within the D-H convention, each homogeneous transformation T corresponds to the overall transformation of one link. Each transformation itself is invariably composed of four more basic transformations about various axis. These are illustrated in Figure 2.2 and described by the respective transformation matrices

$$
\begin{aligned}
T &= R_z \mathrm{Trans}_z \mathrm{Trans}_x R_x \\
&= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.
\end{aligned} \tag{2.11}
$$

Matrix multiplication of these four basic transformation yields the transformation matrix for the whole joint:

$$
{}^{i-1}T_i = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.12}
$$

The parameters r, α, d and θ are called link length, link twist, link offset and joint angle respectively. For any given joint, the matrix T is a function of a single variable because three of the four parameters are fully defined by the link type and geometry. For a prismatic joint, the link offset d is variable while all other parameters are constant. For a revolute joint, only the joint angle θ is variable and all other parameters are fixed.

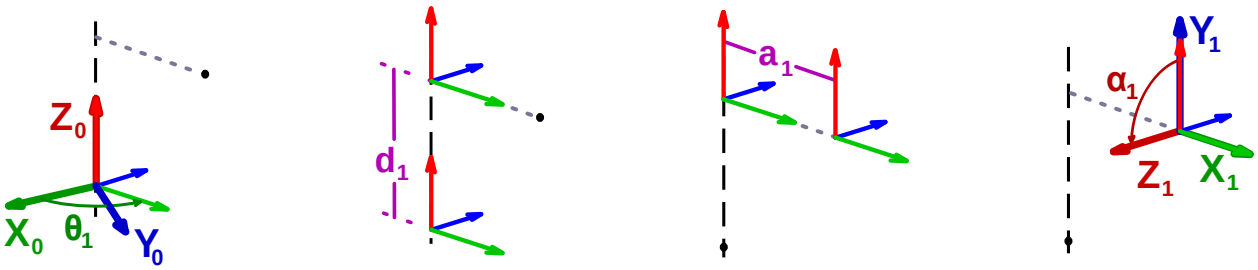

Figure 2.2: Orientation of the axis as governed by the D-H convention.

The D-H convention relies on just these four parameters to describe the offset between two subsequent robot joints, which is less than the six parameters needed to describe an arbitrary transformation in Euclidean space. This is made possible by careful placement of the reference frames. The convention requires that the z axis of the link's reference frame be pointed along its axis of rotation (or translation in the case of a prismatic joint). With the z-axis thus oriented, the orientation of the x-axis is determined next. For the first joint in the chain, the orientation of this axis can be freely chosen among all possible orientations perpendicular to the z-axis. However, for any subsequent joint, the D-H convention requires that the x-axis be pointed along the common normal of the newly defined z-axis and the z-axis of the previous joint.

Finally, the new y-axis is placed to complete a right-hand frame of reference. With the frame placed in this way, the four D-H parameters for each link can be derived from the relative position of two subsequent frames (see Figure 2.2). The link offset d is determined by the distance of the two subsequent x-axes along the z-axis of the previous joint. The link length r is the distance between two subsequent z-axes, i.e., the length of the common normal connecting both axes. The link twist α is the angle of rotation about the new x-axis needed to align the old with the new z-axis.

The parameters for the ABB YuMi used in the experimental part of this work were derived according to this convention and can be found in Table 4.1. The matrices obtained by following the DH convention can directly be substituted for the individual link transformation terms ${}_{i-1}T_i$ in (2.10).

2.2.2 Workspace, Configuration Space and Forward Kinematics

Spaces The previous sections have exclusively used the six-dimensional Cartesian space surrounding the robot as underlying concept for the description of object poses. In the context of robotics, this space is usually called *workspace*. This is the space where the robot interacts and 'works' with its environment. When manipulating objects in this space, the pose of paramount interest in this space is that of the robot end effector, because this is where the

robot arm makes desired physical contact with its surroundings. However, the pose of the end effector in this space does not generally uniquely describe the pose of all robot links in the workspace. In automated manipulator path planning, this knowledge is also important, since collisions have to be avoided along the whole robot chain and not merely at the end effector. Therefore, it is often convenient to work with a different space representation as the Cartesian space representation introduced before.

The vector space defined by the individual joint coordinates is called the *configuration space* of a robot. It is usually abbreviated as $\mathcal{C}$-space or simply $\mathcal{C}$. A point in this space uniquely defines the position of all joints in the kinematic chain and is called a *robot configuration*. In the computation of robot motion, it is often necessary to find correspondences between configuration space and workspace. The problem of finding a pose in workspace that corresponds to a given configuration in $\mathcal{C}$-space is called *forward kinematics* and will be discussed in the remainder of this section. Its solution can be given in closed form for any kind of rigid kinematic chain. The inverse problem of finding configurations that correspond to a given pose is called *inverse kinematic* problem and will be discussed in the following section.

Forward Kinematics The computation of a solution to the forward kinematic problem starts with a given robot configuration c which defines the n robot joint values $\theta_1 \dots \theta_n$. With this configuration, the D-H matrices given by (2.11) are fully defined. Since each of the matrices defines a geometric transformation between a subsequent pair of reference frames, the matrix multiplication of the individual transformation defines a transformation chain along each of the links, leading from the inertial frame to the end effector frame of the last robot link. If a tool is attached to the robot end effector, the tool transformation matrix has to be added as the last element in the chain. The matrix pose representation obtained from the multiplication can be decomposed into position and rotation parts that give the workspace position and orientation as described in Section 2.1.

Note that for any given robot configuration c, there exists a unique solution to the forward kinematics problem. This sets it apart from the inverse kinematics problem. Also, the computation of this solution is computationally very easy. However, the existence of a mathematical solution to the forward kinematics problem does not guarantee that the configuration c is actually a valid configuration for the robot arm, because self-collision between robot links or collisions with objects in the environment are not taken into account in the kinematics calculation.

2.2.3 Inverse Kinematics (IK)

Compared to the forward kinematics problem, finding the set of joint parameters θ that leads to a given workspace pose is less straightforward. Revisiting (2.10), it becomes evident why this is so: while in the forward kinematics problem, the left-hand side must be calculated from a known right-hand side, the inverse kinematics problem requires the calculation of the right-hand side of the equation from a known left-hand side. The set of equations to be solved here is highly nonlinear. Closed form solutions to this problem do not generally exist, and depending on the exact geometrical structure of the robot, no solutions or multiple solutions might exist (Duffy 1980). For kinematic chains with a length of more than six links, all poses within the robot's workspace (but excluding those on the boundary of the workspace) can be reached with infinitely many configurations. This makes the inverse kinematics problem even more challenging for humanoid manipulators with seven degrees of freedom [1]. Since it possess more independent degrees of freedom than its workspace has, this type of robot is called a *redundant* manipulator.

Closed-form solutions While no closed-form solution exists to the general IK problem, it is possible to find algebraic solutions for specific types of kinematic chains. For serial link manipulators with six joints, one of two conditions must be satisfied in order for an algebraic IK solution to exist (Mason 2001):

1. the kinematic chain features three parallel consecutive revolute joint axes, or
2. three consecutive revolute axes have an intersection in a single common point.

Aside from higher manufacturing costs, the mathematical complexity of the inverse kinematics problem is one of the reasons why conventional industrial manipulators rarely exceed six serial degrees of freedom and usually adhere to the second of the given rules for their three last joints. Kinematics with this design are said to possess a *spherical wrist* and a closed-form solution to the IK problem can be specified for these devices. The analytic solution depends on the geometry of the kinematic chain. Therefore, no general solution can be given for robots with six degrees of freedom. Instead, an appropriate solution has to be derived by taking into account the particular geometry of the manipulator.

Numerical solution Numerical solutions on the other hand do not depend on the geometry of the robot. They also apply to robots with more than six degrees of freedom. All numerical approaches to solving the IK problem are based on the robot Jacobian $J(q)$,

[1] This corresponds to the number of degrees of freedom found in the human arm.

which relates the end effector's linear and angular end effector velocity in workspace with the derivative of the n robot joint values $q_1 \dots q_n$ in joint space:

$$\dot{x} = J(q)\dot{q} \Leftrightarrow \begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial q_1} & \cdots & \frac{\partial x}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \omega_z}{\partial q_1} & \cdots & \frac{\partial \omega_z}{\partial q_n} \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{pmatrix} \tag{2.13}$$

This equation can serve as the basis for an iterative approach to the IK problem. Suppose the desired robot pose x_g is known along with the current joint angles q_c. We can compute the current pose x_c from evaluating the forward kinematic equation 2.10 at q_c and then define the current error vector e by

$$e = x_g - x_c. \tag{2.14}$$

We now have to find an increment Δq for the joint values that reduces e. (2.13) gives an estimate of how the workspace position x_c would change given the increment Δq:

$$\Delta x = e \approx J(q)\Delta q. \tag{2.15}$$

Intuitively, a suitable update step Δq can be computed by inverting J to obtain

$$\Delta q \approx J^{-1} e. \tag{2.16}$$

However, even in theory this approach only works when J is invertible. Two conditions have to hold for that to be possible: Firstly, J needs to be a square matrix. The Jacobian can only be a square matrix if the robot has exactly the same number of joints as there are degrees of freedom in the task. If just Cartesian position is considered in the workspace and orientation remains fixed, the number of degrees of freedom is three; in this case, a 3-DOF robot would possess a square Jacobian. If the full positioning problem including 3-DOF position and 3-DOF orientation is considered, the robot needs to have exactly six joints for the Jacobian to be a square matrix.

Secondly, J needs to have full rank to be invertible. Since J is a function of the robot configuration q, it is possible for the Jacobian to become ill-conditioned or lose rank. This happens when two or more of the robot axes align in such a way that an infinitesimal task space motion $\left(\begin{smallmatrix} \mathrm{d}v \\ \mathrm{d}\omega \end{smallmatrix}\right)$ cannot be obtained by any infinitesimal joint space motion $\begin{pmatrix} \mathrm{d}q_1 \\ \vdots \\ \mathrm{d}q_n \end{pmatrix}$. In these robot configurations, two or more rows of J become linearly dependent and the matrix loses full rank. Any configuration causing such a deterioration of rank is called a *robot singularity*. Even when the robot is not in singularity but only near such a configuration, the values of J^{-1} become

very large, leading to very high joint space velocities and instability in the numerical solution approach.

Although helpful for the understanding of the numerical approach to the solution of the IK problem, the direct inverse Jacobian approach is not used in practice for the given reasons. Instead, a number of other inversion techniques aim to reduce instability. These are also applicable to task-redundant robots (i.e., those possessing a larger number of joints than strictly necessary for the task). Consider again the differential forward dynamics given in (2.13), this time assuming a rectangular Jacobian $J(q) \in \mathbb{R}^{m \times n}$, with the number of task-space dimensions m smaller or equal to the number of robot DOFs n. For a given task space motion $\dot{x}$, the (possibly infinitely many) corresponding joint space motions are defined by

$$\dot{q} = J^{\dagger}\dot{x} + (I - J^{\dagger}J)\dot{q}_0. \tag{2.17}$$

This formulation uses the Jacobian's *pseudoinverse* $J^{\dagger}$, which is defined for a rectangular matrix J as

$$J^{\dagger} = J^{\top}\left(JJ^{\top}\right)^{-1}. \tag{2.18}$$

In (2.17), $\dot{q}$ is an arbitrary joint space velocity vector which is projected into the null-space of J by the projection operator $(I - J^{\dagger}J)$. The result is a so called self-motion $\dot{q}_s = (I - J^{\dagger}J)\dot{q}_0$, which keeps the end effector pose stable while internally reconfiguring the robot joints, exploiting the kinematic redundancy of the manipulator. One particular IK solution is obtained when self-motion is disallowed by letting $\dot{q}_0 = 0$, yielding the *pseudoinverse* IK solution

$$\dot{q} = J^{\dagger}\dot{x} \tag{2.19}$$

which can be approximated for use in an iterative numerical solution approach by

$$\Delta q \approx J^{\dagger}e. \tag{2.20}$$

While this solution is applicable to general kinematic chains, it is still susceptible to instability originating from an ill-conditioned Jacobian. Based on the pseudoinverse method, a number of techniques for removing this instability exist. These usually involve some form of damping of $J^{\dagger}$, e.g. damped least-squares inverse kinematics (Nakamura et al. 1986; Wampler 1986) or trajectory modification near singularities (Maciejewski et al. 1988).

2.3 Robot motion: paths and trajectories

The mathematical concepts discussed so far have been helpful in describing the robot geometry and associated configurations and workspace poses. However, the usefulness of a robot mechanism stems not from a static pose but rather from its capability of flexibly *changing* betweens poses, i.e. from its capability to move. In robotics, the idea of motion is generally expressed by making the pose and configuration representations discussed in Section 2.1 a function of some parameter. When the parameter of this function represents a general measure of progress from start to end of the represented motion, the function is called a *path*. In robotics, this function is commonly represented by the expressions $q(s)$ for a configuration space path and $x(s)$ for a task space path. The parameter s is a measure of progress along the robot path. This parameterization is usually normalized such that $s \in [0 \dots 1]$. If instead the parameterization is chosen such that

$$\mathrm{d}s = |x'(\tau)|\mathrm{d}\tau \Leftrightarrow s = \int_0^s |x'(\tau)|\mathrm{d}\tau \qquad \forall s, \tag{2.21}$$

the parameter s corresponds to the arc length of the represented space curve. This type of parameterization is called *arc-length parameterization*. While obtaining such a parameterization is not analytically possible for arbitrary functions x or q, operating with arc-length parameterized paths is often more convenient, for example in trajectory planning.

If the function's parameter is not a general measure of progress but directly corresponds to the time that passes as the robot progresses along the path, the function is called a *trajectory*. As in the representation of paths, these trajectories are usually expressed by the symbols $q(t)$ for a joint space trajectory and $x(t)$ for a task space trajectory, respectively. The trajectory function itself determines the position of the robot mechanism at any given time t. However, the choice of this function also governs the dynamics of the motion via its derivatives $\frac{\mathrm{d}^{(n)}}{\mathrm{d}t^{(n)}}q(t)$, which define the velocities, accelerations and jerks via the system dynamics equation. Both the path and trajectory functions are univariate and vector-valued. While task-space path and trajectory functions define a mapping $x : \mathbb{R} \mapsto \mathbb{R}^6$, the dimension of the image space of configuration space paths and trajectories depends on the number of joints n present in the manipulator, $q : \mathbb{R} \mapsto \mathbb{R}^n$.

In view of these definitions, the general motion planning problem for robots is the problem of finding an optimal function $q(t)$ which satisfies the dynamics equation and obeys the constraints that arise from the particular motion planning task under consideration (e.g., hitting a nail with a hammer held by the end effector with a given force or reconfiguring to reach a final pose x as quickly as possible under task space obstacle constraints). Since the general motion planning problem is often hard to solve, it is usually decomposed into the subproblems of path planning and trajectory planning. Instead of directly finding a suitable trajectory function, a path is first

found with the help of path-planning algorithms discussed in Section 3.2.1. In the subsequent trajectory planning problem, this path is reparameterized to find a trajectory along this fixed path. This decoupling is non-optimal in the sense that the found trajectory is optimal *only for that specific path*, not necessarily for the original general motion planning problem.

3 State of the art

The third chapter of this work will take a closer look at the state of the art relating to industrial dual-arm robots. Its discussion is split into three main themes: The first section will have a look at methods used in programming and teaching industrial robots to do their work as used today. Here, a special focus will be placed on concepts related to understanding how multiple arms can be coordinated using tools available today. The second section deals with the problem of motion planning itself. It will outline the relevant state of the art in path planning for robot arms. In the third section, this chapter will finish with a survey of state of the art algorithms used in multi-robot coordination.

3.1 Teaching Robot Applications

In its most general form, robot teaching is the task of translating human-comprehensible task descriptions into a representation that can be understood and executed by a robot. Depending on the domain of the robotics application in question, this problem has been addressed by a host of different methods in research to date. The following section will give an overview of approaches used in teaching applications of industrial manipulators.

3.1.1 Robot Programming Languages

The vast majority of industrial robot applications running in factories today are still taught using classical text-driven programming approaches that have been used and vastly remained unaltered for decades. These approaches are commonly divided into two different categories, i.e., *online* and *offline* programming. In this context, the terms online and offline describe the state of the robot during teaching. Online programming is carried out with a powered-up and running robot. Usually, the programmer has direct access to the device during online programming. In offline programming, the robot does not need to be running or even physically available. Instead, offline methods use virtual environments, emulators or other tools that can (partially) replace necessity for the presence of the physical robot during programming.

ABB : **MoveAbsJ** P2, v1000, fine, tool1;
Fanuc : **J** P[2], 70%, FINE;
Kuka : **PTP** P[2] (...);

ABB : **MoveL** P2, v1000, fine, tool1;
Fanuc : **L** P[2], 70%, FINE;
Kuka : **LIN** P[2] (...);

Figure 3.1: Point to point and linear robot motion commands in different languages. Their effect in Cartesian space is indicated by the blue line in the right part of the figure.

The foundation that unites classical methods of industrial robot programming is formed by a variety of robot programming languages. All classical programming approaches, however simple or sophisticated, ultimately have to target one or several of these languages. In many cases, they are the only available interface to the manipulator hardware. The interpreters and runtimes for these languages run only on the specific motion controllers supplied together with the robot hardware. Languages used by different manufacturers are therefore incompatible and cannot be used across different manufacturers.

Basic motion commands While there can be considerable syntactical differences, the actual motion commands offered vary only slightly in functionality. This section will introduce some of these motion commands and explain their functionality, thereby essentially defining the behavior an industrial robot can or cannot exhibit today. Some basic motion commands are offered by all robot programming languages without exception. The most basic among these is the command for point-to-point motion, which is shown in Figure 3.1 according to the syntax of three major robot programming languages. This command simply moves the individual joints of the robot such that each joint's current configuration angle transitions into a given goal configuration angle. The motions of the individual joints are synchronized such that all joints reach their individual goal configuration at the same time. This motion type is the most unconstrained of all motion types. Therefore, it is also the quickest way for the manipulator to transition between any two given configurations. However, as can be seen from the pictograph in Figure 3.1, point-to-point motions can lead to end-effector paths in Cartesian space which are very hard to anticipate by a human without actually computing the forward kinematics for the motion (shown in blue). The motion commands supplied by the different

robot languages do require a different syntax and also vary slightly in functionality (e.g., the ABB command accepts a value for maximum end-effector velocity to define the speed parameter while the Fanuc command requires a percentage of maximum motion speed). However, they essentially describe and encapsulate the same behavior - moving the robot arm from one configuration to the next. A second command implemented by any commercially available robot is linear motion, in which the end effector follows a straight line while transitioning between start and goal poses. For a given set of start and goal poses, the execution of this command takes more time as the equivalent point-to-point command. However, the manipulator motion is more predictable in most cases.

Advanced motion functionality Apart from the basic versions of these motion commands, a plethora of variations exist to account for the peculiarities of different applications in different domains of manufacturing. Some of these commands trigger a signal at a specified distance from the end point, others make the end effector follow spline contours and still others let the user specify a maximum contact force and stop the motion once this limit is reached. All of this advanced functionality supplied through robot motion commands is constrained to the sensor set a typical industrial manipulator is endowed with: angle encoders in the joints and current measurement for the joints' actuators. Therefore, industrial robots essentially move blindly and without any further possibility to perceive their environment. Even the advanced motion commands move the manipulator into collision with objects in the environment at full speed if the sequence of motion commands is not carefully crafted to avoid such collision. Guaranteeing collision-free motions with the described motion commands is more subtle than it might seem at first glance, because the actual motion depends not only on the statically defined goal state but also on the robot's start state at the beginning of the motion. Because of the dynamic nature of robotic planning systems, this state is never explicitly specified. Instead, it is implicitly assumed to be whatever state the robot was brought into by the motion command (partially) executed previously.

Avoiding collision in collaborative setups The problem of avoiding collision is aggravated in a collaborative setup, where two or more manipulators work in close vicinity of each other. In this situation, it is not enough to assure that motion commands are collision-free with respect to the static environment. The commands of the different manipulators also have to be synchronized among each other, such that programming a deterministically collision-free motion is possible at all. For these situations, robot manufacturers supply a special API for the temporal synchronization of the motions of several robots controlled by a distributed controller architecture.

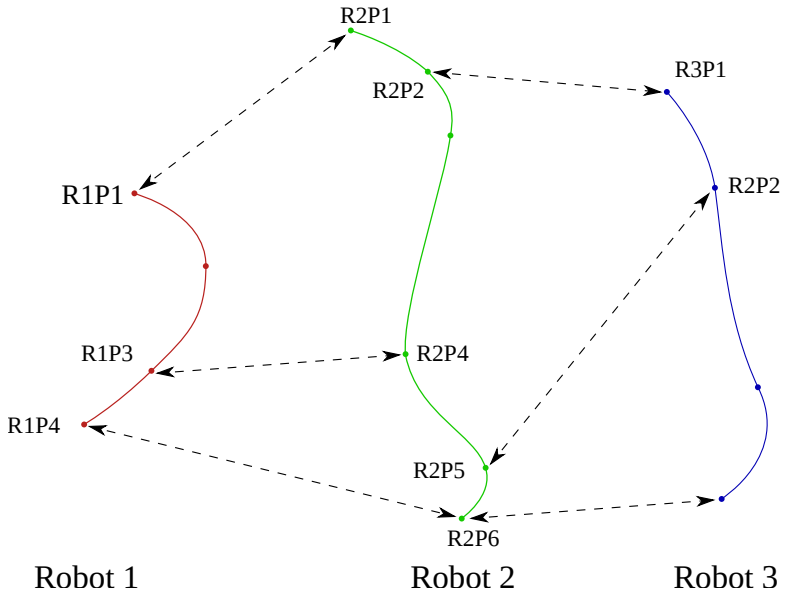

Figure 3.2: Temporal motion synchronization of three robots working in close vicinity. Goal points of individual motion commands are indicated by dots, synchronization points are indicated by dashed lines.

Consider as an example the three robot paths shown in red, black and green in Figure 3.2. Each of these paths is defined through an ordered sequence of commands much like the ones illustrated in Figure 3.1, including four motion commands for the red and green paths and six for the path shown in black. If no additional measures were taken, the three robots would move along their specific paths asynchronously and the programmer would have no way of ensuring that the arms do not collide during their motion. In industrial robotics, this problem is currently addressed by robot manufacturers through special software options like ABB MultiMove or KUKA RoboTeam, which allow temporal synchronization of motion commands across multiple robot controllers. In Figure 3.2, this concept is indicated by the dotted blue arrows: All robots start their motions at their respective start points at the same time. They move with such a speed that they pass by the points connected by the dashed arrows at the same time and arrive at their respective target locations also at the same time – hence the name *temporal* synchronization. In the same case without synchronization, robot no. 3 would arrive at its point R3P2 just as robot no. 2 is passing point R2P3. Since these points are located very closely to each other in the workspace, a collision would ensue. By synchronizing the motion in the shown way, robot no. 3 is delayed, reaches the point only after robot no. 2 has cleared the collision zone, and collision is avoided.

It is important to note that this approach to resolving motion coordination between the arms is fundamentally static. Whenever the possibility of collision between arms arises in a multi-arm system, it has to be resolved by statically specifying a synchronization logic similar to the one illustrated in Figure 3.2 *at programming time*. This essentially means that for every

Figure 3.3: An example of a teach pendant for an industrial robot.

motion command that is at risk of causing a collision, all concurrent motions of all other arms in the vicinity have to be fully specified. This has far-reaching implications as far as usability of multi-arm systems is concerned. In the special case of a dual-arm robot system with its geometrically close manipulators, almost all situations in which both arms move concurrently pose the risk of collision. For these systems, the described temporal synchronization approach to arm coordination severely limits the flexibility and usability of dual-arm robots as one single system[2].

3.1.2 Online programming

As opposed to offline programming, the term online programming is used for all methods that directly employ the physical robot in teaching the robot the desired behavior. In many cases, online teaching of industrial robots relies on a special input device called teach pendant. An example of such a device is displayed in Figure 3.3. These input devices are directly connected to the robot controller and often feature a screen to show various information about the robot's state (robot code, current arm state, etc.) and allow the user to move the robot manually, e.g., via a joystick. Traditionally, industrial robots operate mainly in position-control mode. Therefore, classical teaching methods primarily serve to convey desired workspace or configuration space positions to the robot and define the motions between these. In online teaching, this is achieved by physically moving the robot arm to desired positions and then saving the associated pose or configuration to robot controller memory. Moving the arm can either be done by directly or via an interface device like a teach pendant. Moving the arm directly has the benefit of allowing larger motions more quickly, while operation via a teach

[2]For a case study of a real use case requiring more flexibility see Section 4.2.

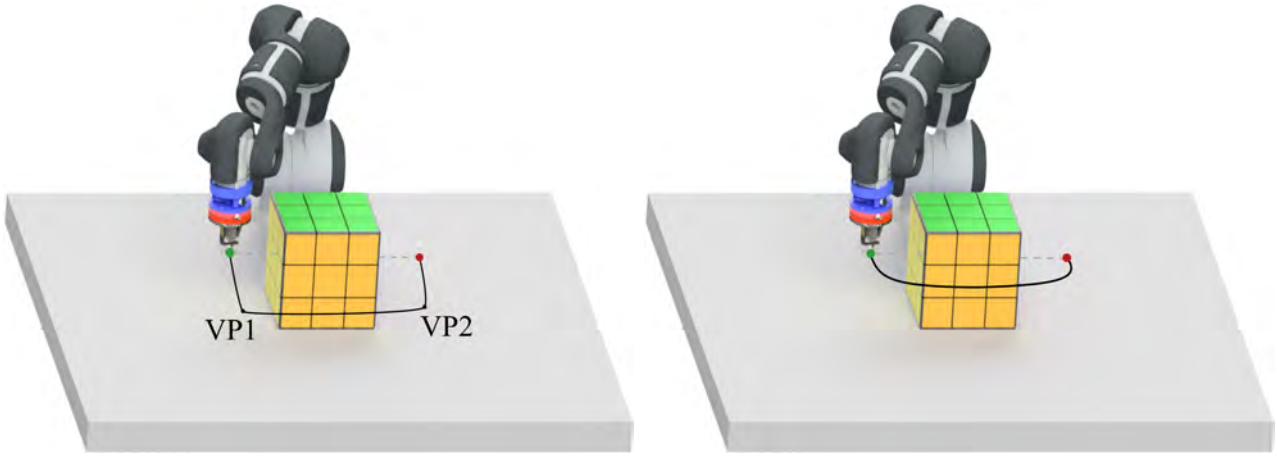

Figure 3.4: **Left:** Manually taught collision-free path defined through via points (generated online). **Right:** Automatically planned collision-free path (generated offline).

pendant allows for very precise motion. Because of these advantages and disadvantages, both approaches are usually used alongside each other when teaching robots online.

Once the key positions for a given task have been taught to the robot, motions can be defined based on these points. Taught positions can be used either as via points or as goals for a motion. In the classical teaching of robots, no automatic path planning algorithms from current research are usually used. Therefore, robot paths have to be made collision-free with respect to the environment by choosing suitable via points for the desired motion. Figure 3.4 illustrates this situation. If the robot end effector were to approach the goal directly, it would collide with the obstacle obstructing the direct path. In contrast to automatic path planning (see Section 3.2.1), manual teaching uses one or more via points to define a motion that avoids collision.

When all via points needed to define a collision-free motion are known, the logic of the robot motion can be defined. As discussed in Section 3.1.1, industrial robots usually come with their own manufacturer-specific controller and associated programming language that is used for this purpose. Robot languages like Kuka's KRL or Fanuc's Karel (see Section 3.1.1) ship with an instruction set that includes different kinds of motion commands implemented for the robot. Using ABB's RAPID as an example, the collision-free motion shown in Figure 3.4 would be represented as shown in Listing 3.1.

```
MoveL VP1, v1000, z30, noTool;
MoveL VP2, v1000, z30, noTool;
MoveL PGoal, v1000, fine, noTool;
```

Listing 3.1: Simple via-point motion in ABB Rapid.

In this program snippet, the robot is instructed to first move to the position `pVia` that has been defined in the previous step of teaching. Having cleared the obstacle collision zone, the

robot is then instructed to move on to the actual goal position `pGoal`. The other parameters of the motion define the desired speed, the tool transformation used in the inverse kinematics calculation and the magnitude of the path blending used in the transition zone between the two motion segments. In online teaching, these commands are selected, parameterized and edited directly on the interface device supplied by the robot. The motion program built in this way can be enriched by more complex logic through any of the syntactical constructs allowed by the robot programming language. When online programming is finished, the program can be saved to the controller and executed.

3.1.3 Offline programming

As robot programs become more complex, developing them directly on the teach pendant becomes tedious and inconvenient. This is where offline programming – the second of the two branches of classical robot programming methods – comes into play. As opposed to online robot programming, offline programming methods are not bound to using the teaching interfaces supplied by the robot hardware directly. Instead, development is typically carried out with the help of OEM or third-party development software running on a general-purpose computer that is physically and logically separated from the robot hardware. Unleashing development from the interface devices offered by robot hardware, this kind of robot programming enables more productivity in designing complex robot applications. However, the link to the real robot hardware and the geometry of the physical station is lost. Therefore, offline programming tools have to rely on models of the robot itself as well as of the geometry surrounding the robot as a basis for programming. As a second key element, these environments often include a tool to develop and inspect text-based robot programs (see Figure 3.5). A survey of available offline programming software with an at least partial focus on industrial robotics can be found in (Pan et al. 2012).

The benefits offered by offline programming come at a cost. Accurate enough models of the robot and workspace geometries are not always easy to obtain and commercial offline programming software is often very expensive. Therefore, this approach is usually used in larger-scale robot applications where reduction of machine downtime and optimal cycle times are a priority. Nevertheless, the model-based approach of offline robot programming is powerful and makes it possible to solve problems that cannot be addressed satisfactorily by online programming. A number of vendors offer software with a focus on different aspects of offline robot programming. The most important aspects are:

- path planning (useful in geometrically constrained environments)

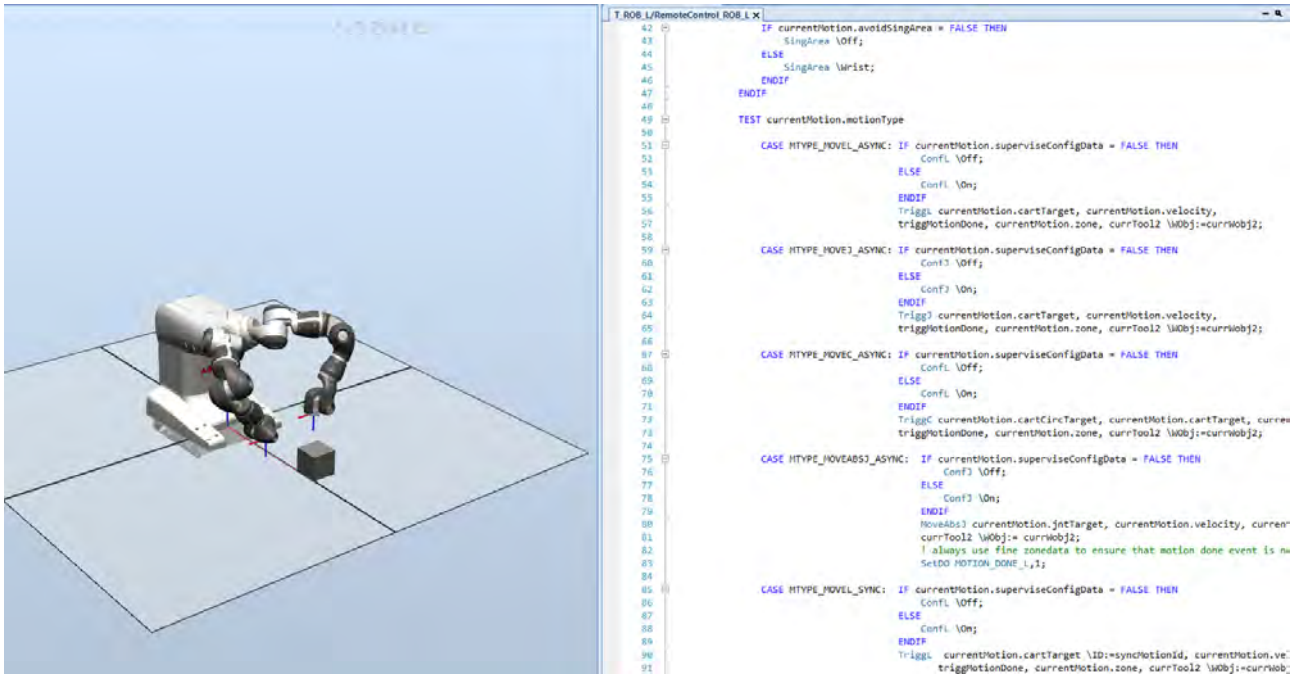

Figure 3.5: Offline Programming Environment ABB Robot Studio.

- code generation from 3D geometrical task representations
- layout optimization (reachability, cycle time)
- task sequence optimization
- generation of spline contour curves from 3D models (e.g., for welding/gluing applications)
- trajectory optimization (length, cycle time, energy)
- motion optimization for specific tasks (e.g., welding)

Once the robot motion has been defined, robot code for the respective target platform is generated. This step couples the offline programming with online program execution. However, this step is not always straightforward. As the name implies, offline programming is carried out without using the physical robot and is instead based on models of the robot and the environment. This entails a correspondence problem. Models can never capture the real world in full detail. For example, exact information about the relative position of the robot with respect to the environment is often not available. For this and related reasons, offline programming alone is not sufficient in many cases. Instead, the programs generated offline have to be manually edited and tuned in a final step with the system running to account for the errors of the models used in offline programming.

3.2 Motion planning

As we have seen already, motion is the fundamental concept that makes a robot a robot. Thus, the motion planning problem is the overarching theme in robotics. When interpreted broadly, anything a robotic system does is driven by motion planning. Surely, calculating joint trajectories and motor torques is. However, seemingly unrelated things like perceiving the environment also are. After all, the reason why a robotic system tries to perceive an obstacle is to be able to compute an evading motion. Driven by the recent advances in machine learning techniques, the problem of motion planning has indeed been approached from this all-encompassing perspective. For example, several researchers have tried to use end-to-end control approaches to calculate motion based directly on raw perception data (Levine et al. 2016; Hoppe et al. 2017) or used other input data for end-to-end learning (Pierson et al. 2017).

Although recently featuring very popularly in academic research, these methods have not yet seen noteworthy use in production applications for lack of maturity. Instead, for better tractability, the end-to-end control problem is usually broken down into the subproblems

1. Behavior planning
2. Motion planning, which itself is often broken down into
    (a) Path planning and
    (b) Trajectory planning
3. Trajectory execution

Of these, only the items 2(b) and 3 are usually implemented as native robot functionality. All higher-level functionality exists only in separate programs as described in 3.1.3. For the sake of brevity, the following discussion of the state of the art will focus on the area of this work's contribution, which mainly relates to the core motion planning problem of item 2.

3.2.1 Path Planning

Once a desired behavior for the robot has been determined, either statically by a human programmer or by some dynamic behavior generation mechanism, the question arises if and how that behavior can be carried out without collision in the current workspace. For example, an industrial manipulator mounted in a work cell must avoid collision with itself, the static structures of the cell and also with other moving objects that can access the same workspace. This means that motion planning comprises a temporal and a spatial domain, i.e., it needs

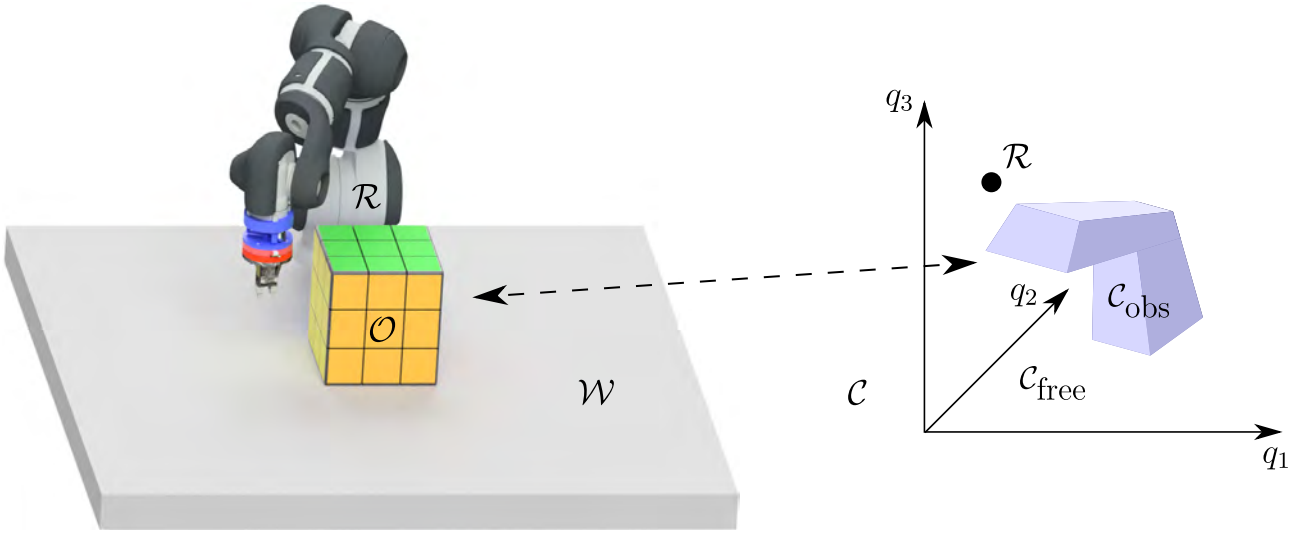

Figure 3.6: Workspace and configuration space regions.

to answer the questions of *where* to move and *when* to move there. Because answering both questions at the same time is very complex for real systems operating in real environments, they are often treated one by one and sequentially. This decoupling strategy gives rise to the subproblems of path planning and trajectory planning, which will be discussed in this section.

In the basic formulation of the problem, robot path planning is concerned with finding a path between two robot poses, the so-called start and goal poses. Since the path problem considers only the kinematic aspect of motion, the problem is deemed solved once a path function $q(s)$ is known along which the robot $\mathcal{R}$ can transition between the two states without causing a collision in the workspace $\mathcal{W}$. Therefore, geometric models representing $\mathcal{R}$ and $\mathcal{W}$ are needed by any path planning algorithm.

Planning spaces The first question that arises in the problem of robot path planning relates to the definition of the path function $q(s)$ itself. The first and intuitive association with the word *path* probably is that of a two- or three-dimensional space curve in the workspace representing the path of the robot end effector. However, this is not a suitable path representation in most situations. The reason is that path planning algorithms need to be able to check if any given portion of the path causes a collision between the geometric models for $\mathcal{R}$ and the workspace obstacle region $\mathcal{O} \subset \mathcal{W}$. Therefore, the image space of the path function must uniquely define the robot's configuration. As discussed in Section 2.2.2, this is not true for end effector paths. While a curve in the workspace does not generally define such a configuration, a curve in configuration space does. This is why most path planning algorithms operate on the basis of the configuration space.

Consider again the obstacle shown in Figure 3.6. The colored cube covers the workspace obstacle region $\mathcal{O} \subset \mathcal{W}$. Let the portion of $\mathcal{W}$ occupied by the robot in a given configuration q be given by $\mathcal{R}(q) \subset \mathcal{W}$. Then, the obstructed portion $\mathcal{C}_{\text{obs}}$ of the $\mathcal{C}$-space is given by

$$\mathcal{C}_{\text{obs}} = \{q \in C | R(q) \cap O \neq \emptyset\}. \tag{3.1}$$

In other words, all the configurations q which cause an intersection between the three-dimensional shapes of the manipulator and other objects belonging to the workspace lie in the $\mathcal{C}$-space obstacle region. The complement of the obstructed space is the so-called free space $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$. This is the part of the configuration space available for path planning. As can be seen from (3.1), the general definition of the obstructed and free subspaces can only be given in implicit form[3]. Moreover, it depends on the three-dimensional shape of the manipulator and the obstacles present in the robot's workspace. These two problem features make even the basic kinematic path-planning subproblem discussed here PSPACE-complete (Canny 1988). Some extensions of this problem targeted at better capturing the reality of robot motion (i.e., uncertainty, dynamic obstacles, etc.) have even been shown to be NEXPTIME-hard (Choset et al. 2005).

The following sections will discuss the two most important path planning approaches with applications to robotic manipulators. First, the sampling-based planning framework will be introduced. All path-planning approaches in this category rely on drawing randomly distributed samples from $\mathcal{C}$-space and using these to build a random geometric graph that characterizes the topology of the free space. Methods belonging to this category are most widely used in robotic path planning today. The subsequent section will briefly touch upon an alternative method that sees less frequent use.

3.2.2 Sampling-based path planning

Sampling-based path planning currently is a very active research area in robotic motion planning. At the time of this writing, the collection of sampling-based motion planners maintained by the well-known *Open Motion Planning Library* (Sucan et al. 2012) contains well over forty different planners. Although the algorithms used in these planners differ substantially, they all rely on a common foundation for robot path planning. They do not try to construct an explicit representation or approximation of either C_{free} or C_{obs}. Instead, this group of planners employs

[3] An explicit representation of these can only sometimes be given in the special case of two-dimensional spaces (LaValle 2006). Very efficient path planning algorithms exist for these special cases. However, they do not extend to configuration spaces with more than two dimensions. Since this makes them unsuitable for the more general case of manipulator path planning, they will not be discussed here further.

different techniques for drawing sample configurations from configuration space and then connecting these samples to build a graph in free space. To determine whether a drawn sample lies in C_{free} or C_{obs}, these planners rely on a collision checker module which accepts a single robot configuration and checks the geometric model of the space for non-empty intersections, thus labeling the configuration valid or invalid. In a subsequent step, sampling based-planners employ a variety of techniques to connect the validly drawn samples to a graph representing possible robot configurations in the free space. After such a graph has been built, a connection between any two configurations in this graph can be found by graph search algorithms. Therefore, the output of sampling-based path planning algorithms (the *path*) is an ordered sequence of configurations $\{q_1, q_2, \ldots, q_n\}$ which connects the start and goal configurations q_s and q_g. The sampling-based path planning algorithms can be classified into two groups according to the number of times the constructed graph is used for path extraction. *One-shot* algorithms build a new graph for each new combination of start and goal configurations, while *multi-query* methods reuse and possibly refine the same graph for new path planning problems. Since one-shot algorithms solve the path planning problem in a single step, they're more suited to serve as an example for the illustration of basis concepts used in sampling-based planning.

Sampling-based algorithms: Example As an example, consider the *RRT-Connect* by Kuffner et al. (Kuffner et al. 2000) and used as a basis for the collision-free planner in this work. In this sampling-based path planning algorithm, two random graphs are grown from start and goal configurations respectively until a connection is found and a path can be extracted. Consider the main logic of RRT-Connect given in Algorithm 1:

Algorithm 1: RRT-Connect (q_{start}, q_{goal})

```
1  T_a.init(q_start);
2  T_b.init(q_goal);
3  for k = 1, ..., k_max do
4  |   q_rand ←SampleFree;
5  |   if not (Extend(T_a, q_rand) = Trapped) then
6  |   |   if Connect(Tb, q_new) = Reached then
7  |   |   |   return Path;
8  |   |   end
9  |   |   Swap(Ta, Tb)
10 |   end
11 end
```

Path search starts with two graphs $\mathcal{T}_a = (V_a, E_a), \quad \mathcal{T}_b = (V_b, E_b)$ rooted at the start configuration q_{init} and goal configuration q_{goal}, respectively. The initial vertex sets are consequently

$V_a = \{q_{\text{start}}\}$ and $V_b = \{q_{\text{goal}}\}$. Accordingly, the edge sets are initialized to $E_a = E_b = \emptyset$. In each iteration of the main algorithm loop, a random configuration is drawn from $\mathcal{C}_{\text{free}}$ sampled according to some probability distribution by the function SampleFree. Note that this sampling is usually not based on $\mathcal{C}_{\text{free}}$ directly. Instead, new configurations are sampled from the entirety of $\mathcal{C}$. These samples are checked for membership of $\mathcal{C}_{\text{obs}}$ via the collision-checker and rejected if found invalid. In this case, new random configurations are sampled until a valid sample from $\mathcal{C}_{\text{free}}$ is found. Once a valid sample has been drawn, the function Nearest selects the vertex x_{nearest} from V that is closest to x_{rand}.

In the next step, the graphs are extended with the drawn samples in an alternating fashion. In each iteration, one of the two graphs is extended toward the newly sampled configuration within the function Extend. Depending on the success of the extend operation, this function can return one of three possible results: *Advanced, Reached,* or *Trapped*. Consider the logic of this routine given in Algorithm 2:

Algorithm 2: Extend $(\mathcal{T}, q)$

```
1  q_near ←Nearest(q, T);
2  if StepToward(q, q_near, q_new) then
3  |   T.addVertex(q_new);
4  |   T.addEdge(q_near, q_new);
5  |   if q_new = q then
6  |   |   return Reached;
7  |   else
8  |   |   return Advanced;
9  |   end
10 else
11 |   return Trapped;
12 end
```

The first step during graph extension is the search for the vertex q_{near} already present in the graph which is closest to the newly sampled random vertex. Subsequently, the tree currently being extended is grown toward this new configuration step by step, starting from the closest vertex. The function StepToward interpolates a new vertex q_{new} which is at most ϵ distance units away from q_{near} and closer to the random sample q by the same distance. If this initial extension succeeds (i.e., no obstacle is hit directly), a connection attempt from the other tree to the current tree is made within the function Connect, which simply iterates the Extend procedure until either an obstacle is hit or the trees are successfully connected:

If the connection attempt in Algorithm 1, line 8 was successful in this iteration, the start and goal graphs have been connected, and thus a solution path can be found. If the connection

Algorithm 3: Connect($\mathcal{T}, q$)

```
1 repeat
2 |  S ←Extend(T, q);
3 until not (S == Advanced);
4 return S;
```

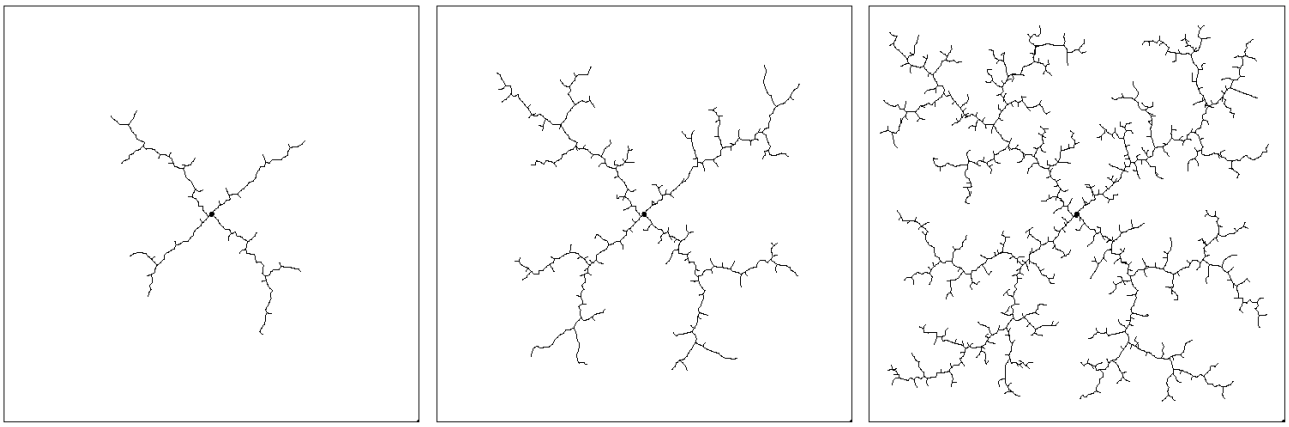

Figure 3.7: The basic process of sampling-based path planning.

attempt was blocked by an obstacle and CONNECT returned *Trapped*, the algorithm continues its next iteration with a new random sample and swapped trees. Therefore, in the next iteration, the other tree is extended iteratively via the connect function.

A graphical representation of this process is given in Figure 3.7. The trees grown by the RRT-connect algorithm after k iterations are shown here for a hypothetical two-dimensional configuration space with a priori unknown obstacle regions and ascending values for k. After initialization ($k = 0$), the trees contain only the start and goal configurations and the configuration space has not been explored yet. After 100 iterations of the main loop (lines 3 to 11 in Algorithm 1), both trees have grown to uniformly explore the configuration space. However, the obstacles have blocked all connection attempts of the obstacles so far. After $k = 127$ iterations, the algorithm has found a collision-free path connecting both configurations.

Variations of sampling-based algorithms Other sampling-based path planning methods follow the same general framework of (i) sampling from $\mathcal{C}$, (ii) building a graph to represent $\mathcal{C}_{\text{free}}$ and (iii) extracting a path when both start and goal configurations are contained in a connected component of this graph. However, the algorithms exhibit large varieties in the details of when and how exactly these steps are performed (Elbanhawi et al. 2014). Since the sampling process is at the heart of sampling-based path search, an abundance of different techniques have been proposed for this step. For example, non-uniform sampling techniques with an increased sampling density near obstacle boundaries (Amato et al. 1996; Boor et al.

1999) or in narrow passages (Sun et al. 2005; Hsu et al. 2004) have been explored to focus the search on geometrically difficult regions of the $\mathcal{C}$-space. This topic is still a very active research area. Similarly, many different techniques have been explored for building a graph from the valid samples, which is the second cornerstone of sampling based planning. In standard RRT-connect as described above and other early methods, edges are added to the graph once and remain there for the graph's lifetime. In contrast, recent research has explored the idea of reconnecting existing vertices as new samples are drawn, thereby optimizing the connectivity between vertices in the graph and yielding shorter paths between any two vertices as planning continues. This idea spawned research in the class of optimizing sampling-based planners such as *RRT** (Karaman et al. 2011), *BIT** (Gammell et al. 2015) and *RRT♯* (Arslan et al. 2013), which all incorporate ideas from optimal graph-search algorithms like *A** directly into the planning process.

In contrast to the some of the more specialized path planning algorithms not considered in this discussion, sampling-based path planners do not provide completeness. This means that they are not guaranteed to finish in finite time, returning either a valid solution or the proven non-existence thereof. Instead, drawing new sample points continues as long as no solution has been found, thereby effectively increasing the sampling density and the ability of the algorithm to capture narrow regions of the free space that might provide a solution. For example, in case the free space contains more than one connected component and start and goal configurations lie in different components, the algorithm would not return. However, sampling-based algorithms do provide a weaker form of completeness. Because of their basic working principle, the user is guaranteed to eventually obtain a solution *if one exists*. This feature is called *probabilistic completeness*. Some planners are also *asymptotically optimal*, i.e. instead of guaranteeing to find any solution eventually, the returned solution will converge toward the optimal solution as algorithm runtime approaches infinity.

3.2.3 Artificial potential fields

Another noteworthy but now less widely-used group of path planning algorithm relies on a different approach. Instead of treating the configuration space as a black box and exploring it by random sampling, a deterministic potential function $U : \mathbb{R}^n \to \mathbb{R}$ is defined over the configuration space to model the obstacle region as shown in Figure 3.8. In the simplest case, this function is the superposition of a navigation potential U_n with a global minimum at the goal configuration of the desired motion and an obstacle potential U_o:

$$U = U_n + U_o \tag{3.2}$$

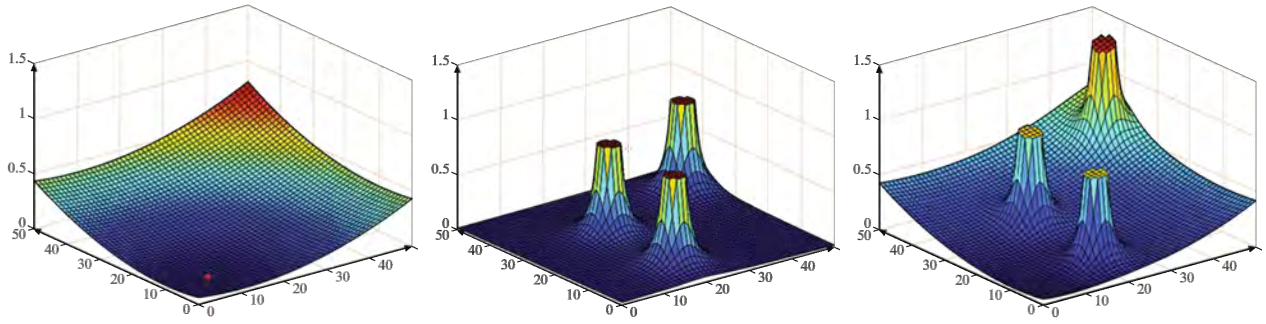

Figure 3.8: **Left:** Navigation potential. **Center:** Obstacle potential. **Right:** Combined potential field (Siciliano et al. 2016).

The gradient $\nabla U = \left[\frac{\partial U}{\partial q_1}, \ldots, \frac{\partial U}{\partial q_n}\right]^{\mathrm{T}}$ is a vector, which can intuitively be understood as a force acting on the robot. In analogy with principles of describing and electric potential, the robot can be modeled in configuration space as a particle with positive point charge as shown by the red dot in the left figure. Employing simple gradient descent on U, the robot will be guided from any start configuration to the goal configuration. When obstacles are present, they can be modeled via a positive potential as seen in the central part of Figure 3.8. Superposition of both potentials as per equation 3.2 yields the overall potential used for finding a path.

Figure 3.8 shows the situation for a robot with three degrees of freedom – therefore, the configuration space is a Euclidean space. Here, the potential of obstacles can be modeled comparatively easily. However, modeling obstacle potentials for the case of industrial manipulators with six or more degrees of freedom is less straightforward. As in sampling-based path planning, the reason for this is that the three-dimensional shape of obstacles cannot be explicitly transformed to configuration spaces with an arbitrary number of dimensions. Instead of modeling obstacle potentials directly in $\mathcal{C}$, the obstacle potential is modeled in the workspace of the robot. The manipulator itself is then defined via multiple control points (at least one per link). The force resulting from U is calculated for each of these points and then transformed to $\mathcal{C}$ via inverse kinematics methods as described in Section 2.1. Compared to sampling-based methods, artificial potential methods thus employ a conceptually different approach to path planning. Sampling-based path planning algorithms serve to *explore* the space of possible solutions by random sampling that yields an ever-finer characterization of $\mathcal{C}_{\text{free}}$. If a solution exists, this exploration will eventually find it. In contrast, the group of artificial potential field methods focus exclusively on *exploitation* of the predefined guiding function U and do not explore the solution space. It is generally very difficult to define a potential function without local minima. For this reason, path planning via artificial potential functions often gets stuck in one of the local minima and does not find a solution path even if one exists. Consequently, recent research in the area of APF-based motion planning has moved into the direction of sampling

based approaches, effectively transforming them into sampling-based techniques as well (Byrne et al. 2014).

3.3 Coordination of multiple industrial robots

The coordination of multiple robots is a wide field. Example applications include use cases as disparate as the coordination of multiple unmanned aerial vehicles for terrain charting (Zhang et al. 2013; Bhattacharjee et al. 2011), the bi-manual actuation of a turning valve with a dual manipulator mounted on a flying platform (Korpela et al. 2014) or the scheduling of multiple mobile intra-logistics platforms for use in industrial manufacturing (Kleiner et al. 2011). A number of groups also research the general problem of regrasping and bimanual manipulation with closed kinematic chains for object handling (Harada et al. 2012) or even dual-armed catching of flying objects (Salehian et al. 2017). The number of independent robots participating in the coordination ranges from two to several hundred (van den Berg et al. 2011; Regele et al. 2006). Similarly, the control space dimensions for the single robots range from just two in 2D Cartesian space control to seven and more in redundant kinematic manipulators. Moreover, focus can be put onto very different levels of the coordination problem, with related work ranging from the incorporation of high-level mission information (Ulusoy et al. 2013) to low-level low-energy communication strategies for embedded devices (Yan et al. 2013).

The different characteristics of each of these situations often allows for very different algorithmic approaches to robot coordination. For example, while a coordination algorithm targeting the relatively simple case of a 2D Cartesian control space may easily be able to coordinate the motion of several mobile platforms translating in the x-y plane, the underlying ideas might scale not even to a 3D space, let alone to the higher-dimensional spaces relevant in the coordination of manipulators (Surynek 2010). The subsequent overview of state-of the art techniques for multi-robot coordination will therefore focus on the subset of algorithms that are at least theoretically applicable to the problem discussed in this thesis. To reiterate, the discussion will focus on the coordination of two or more serial-link manipulators with six or more DOFs executing tasks that are not inherently or mechanically dependent. An examples of such a task can be found in Section 4.2.

The state of the art in this area is twofold. On one hand, industrial dual-arm manipulators are available that exhibit system-level coordination of the arms to some measure. The state of the art in industrially available devices will be surveyed in the first of the two sections. On the other hand, the topic of dual-arm coordination has also been explored in academic research. Approaches found here but not in commercially available devices will be discussed in the second part.

Figure 3.9: Rethink Robotics' dual-arm Baxter robot features a motion controller actively avoiding collisions between the arms.

3.3.1 Dual-arm coordination in industrial applications

In some factory automation environments, robots need to cooperate in order to solve an automation task together with other robots or humans. In the majority of today's industrial applications featuring multiple robots, the cooperation is not very tight, i.e., the robots' workspaces exhibit only small overlap and no synchronized motion between different manipulators is required by the task. In these simple cases, the dual-arm coordination problem can often be solved manually by the human robot programmer. Single robots are programmed as if other manipulators were non-existent, and the orchestration of the different devices is handled by a central control unit through careful temporal sequencing of the motions. In this layout, a human programmer is responsible for tracking workspace occupation and for ensuring that only such motion commands are triggered that will not cause a collision of different manipulators. Virtually all vendors of commercially available robots offer functionality that allows such a temporal synchronization across multiple robot arms and controllers. Details can be found in Section 3.1.1.

As the integration of multiple arms becomes tighter, as is the case in dual-arm systems, this coordination approach becomes more and more disadvantageous and challenging to implement. In recognition of this problem, some vendors of multi-arm devices do include some kind of system-level functionality targeted at facilitating the dual-arm teach-in process. Arguably, the algorithmically most advanced device in this category is Rethink Robotics' *Baxter* robot shown in Figure 3.9. This dual-arm 2x7-DOF robot features self-collision avoidance between the arms. To achieve this, the robot controller runs an over-approximating collision model of each link, which is a simplified representation of the arm geometry. When contact between link models is

detected during robot motion, avoidance torques are applied to the respective joints to avoid collisions between the arms (Rethink Robotics 2015). The exact details of the algorithm in use are unpublished. However, physical interaction with the device strongly suggests that some form of task space-based inverse kinematics control[4] technique is used (Wang et al. 2017; Petrič et al. 2011). This method is similar in design to the artificial potential field methods described in Section 3.2.3, with the difference that the potential field U isn't defined over the whole workspace. Rather, repulsive forces are calculated for a select number of points, usually based on minimum-distance considerations, and then translated to avoidance velocities/torques in joint space. Just as artificial potential field methods, this collision avoidance method does not explore the solution space but rather exploits a predefined collision avoidance strategy by applying the torques calculated as a function of the current task-space geometry. When used as a dual-arm path planning strategy, the same considerations about local minima apply that have been discussed in the respective section of this work.

In contrast to the Baxter robot, which targets lightweight industrial use case scenarios as well as the scientific research community, other vendors of industrial robots focus more on deterministic predictability, reliability, repeatability and precision. These are features highly in demand in many industrial automation scenarios. This is one of the reasons why algorithms similar to the ones driving the Baxter robot haven't so far appeared in other industrial robots. However, the fundamental difference brought about by the tight integration of arms in dual-arm devices has been acknowledged by other vendors as well. Similarly to the Baxter robot, the controller of ABB YuMi's robot includes a simplified geometric model of the device's body and arms. During motion execution, this model is constantly updated and impending collisions are recognized. If the respective control feature is enabled, both robot arms go into emergency stop mode when evaluation of the model shows that two links are coming too close to each other. Similar functionality is available in the controllers of several other robot manufacturers, e.g., Motoman (Yaskawa 2006) and Kuka (KUKA Roboter GmbH 2013).

While all these features are based on a geometric model of the robot and essentially different implementations of the same idea, they do vary slightly in functionality. For example, while the ABB YuMi's geometric model is fixed and cannot be altered by the end user, the collision model of the Motoman controller can be supplemented by a number of user-defined geometric primitives (i.e., cylinders and spheres) to account for different end effector and tool geometries. Regardless of these slight differences, all of these features are not using the geometric model to actively plan and avoid collision. Instead, they trigger a complete stop of all motions and are therefore meant to prevent or limit the damage caused by a programming error by the human

[4] For a video showing such collision avoidance behavior, refer to https://www.youtube.com/watch?v=e_8BOq-mVJk

robot programmer. In contrast to the approach available in the Baxter robot, they do not provide active assistance in the system-level application of dual arm robots.

3.3.2 Dual-arm coordination in academic research

In academic research, there are two main algorithmic groups of approaches that solve coordinated motion planning problems: centralized and decoupled ones (Latombe 1991). In centralized planning, all robots are represented as a single robot by uniting their configuration spaces into a large composite configuration space. The dimensionality of the composite space is equal to the sum of degrees of freedom (DOF) of all involved robots (Sánchez et al. 2001). The decoupled approach splits the problem into two stages. In the first stage, individual paths are planned assuming a static environment and ignoring collisions between robots. In the second stage, the motion law is modified to avoid collisions between robots. One approach to solving this problem is to find a new motion law in coordination space, the idea of which was initially proposed by O'Donnell et al. (O'Donnell et al. 1989).

Centralized Planning approaches Centralized planning approaches take all the degrees of freedom in the system into account at the same time. For dual-arm manipulators, the number of degrees of freedom in the system usually equals the sum of all elementary degrees of freedom. In the presence of dynamic obstacles, execution time has to be considered as an additional degree of freedom. Incorporating this idea into sampling-based planning leads to planners that operate in configuration-time-space as laid out for example by van den Berg (van den Berg et al. 2006) and others (Bennewitz et al. 2001; Tsai et al. 2009; Liu et al. 2010). However, planning in such a composite space is difficult for high-dimensional manipulators, because path and trajectory planning are efficiently merged into one planning stage and constraints like joint velocities and acceleration cannot be taken into account efficiently[5]. The following section will give an overview of works that have tried to apply the centralized planning paradigm to the coordination of multiple manipulators.

Sanchez and Latombe propose a single-query derivative of the Probabilistic Roadmaps planner (PRM) with lazy collision checking (Sánchez et al. 2002). They propose to apply this planner to a multi-manipulator setup with two to six 6-DOF manipulators executing welding tasks in an automotive body shop. They also provide extensive simulation results of planner test runs in this environment. For coordination, three different planner paradigms are examined; firstly, the proposed planner is used as a centralized planner that plans paths for all robot joints at the

[5]This is the reason why path and trajectory planning are usually treated as a decoupled problem in state of the art approaches (Pham 2014).

same time. The implicit assumption here is that all tasks and all inter-robot task couplings are statically defined and known a priori. The second proposed planning paradigm is called *global coordination* by the authors. In this scheme, the proposed planner is used as a decoupled planner which calculates a coordination plan on an a-priori known path ordering over all robots. In the third coordination paradigm, the same algorithm is used in a sequential pairwise coordination setup, in which C-spaces are defined for pairs of two arms but state validity is checked for each state across all the arms for which plans have already been computed.

Most robot coordination approaches applicable to high-DOF systems rely on concepts originally developed for path planning for single manipulators, such as the sampling-based paradigm or the formulation as an optimization problem. An approach proposed by Ćurković and Jerbić (Ćurković et al. 2010) marks an exception to this rule. Instead, they investigate the use of a genetic algorithm for arm coordination, namely a modified version of a cooperative coevolutionary algorithm. Mimicking the processes of genetic evolution, an individual in the population is modeled as the string of 2D-joint configurations between end and start. These individuals are mutated with a given probability to form new members of the population. Then, a number of fitness criteria based on e.g. inter-robot collision and total motion distance are applied to select the best members of the current population until a collision-free path has been found. The proposed approach is applied to a simulated system of two SCARA-type robots with two links each, operating in a planar workspace. Application to a real system is explicitly placed beyond this paper's scope by the authors for a number of reasons.

Decoupled planning approaches Since centralized planning approaches are still too computationally expensive for online use, we focus on the decoupled approach in this work. Although completeness is sacrificed by constraining the coordination to precomputed paths, it is significantly faster than the centralized approach. This is due to the fact that the dimensionality of the coordination space is much lower than the dimensionality of the composite configuration space. Further details on the coordination problem can be found in surveys by Smith et al. (Smith et al. 2012) and Yan et al. (Yan et al. 2013). These approaches are the only viable ones to use for situations in which online-capability through replanning is a requirement. In fact, a case for the use of online-capable dual-arm planning in robot manipulation was prominently made very early on by Li and Latombe (Li et al. 1995). Almost 25 years ago, they maintained that

> *" Off-line planning is virtually useless in dynamic environments that involve events whose occurrences in time and space are not precisely known ahead of time. On the other hand, while on-line planning can potentially deal with such environments, it raises difficult temporal issues which have not been thoroughly addressed*

> *by previous research. [...] On-line motion planning has the potential to significantly reduce the development time and implementation cost of these cells, while increasing their throughputs. Moreover, since the timing of the operations no longer requires off-line prior analysis, cells can also be more flexible; for instance, they may be dynamically assigned new tasks without interrupting current operations."*
> (Li and Latombe, 1995)

Now, almost a quarter century later, the challenges as well as the opportunities set out then remain just as valid. In subsequent years, several researchers have tried to solve the challenges and exploit the benefits. The following will give an overview of the state-of-the art approaches that are most relevant to the algorithm described in this work and identify potential for improvement.

Lee et al. (Lee et al. 2014) propose a decoupled planning algorithm for a dual-arm robot working on a packaging task for mobile phone cases. They propose a decoupled approach with a priori knowledge of all robot paths for the individual arms. A collision-free coordination of the arms is computed via a custom-built search of a path-based, discretized coordination space (Todt et al. 2000). The algorithms searches the space in one of three directions – advancement along both robots' paths and advancement along just one of the robots paths. By default, greedy exploration is chosen and backtracking is employed when greedy search fails. To satisfy the assembly sequence constraints, the authors maintain an index of assembly constraints that is taken into account in the setup of the configuration space. It also constrains the search strategy as necessary, effectively leading to a subdivision of C-space into multiple cells. Computational evaluation of the proposed approach shows that the runtime for the coordinated planning algorithm is on the order of more than 700 s, around half of which are used for path planning and half of which are used for coordination. Because of these properties, the proposed approach is essentially an offline coordination approach which leverages the decoupled paradigm to allow for increasing the centralized planning benchmark for their use case, which weighs in at an average time of more than 1000 seconds.

As one of the few of its kind, the solution proposed by Lee et al. was explicitly designed with real-time capability as a specific goal (Lee et al. 2001). Quite reasonably, the authors argue that uncertainties in execution runtime even across iterations of the same task vary considerably – subsequently, static offline coordination is prone to producing highly suboptimal plans in reality and an event-based online-capable control strategy is necessary in many cases. They propose to implement such a planner by learning a discretized representation of the free and obstructed subspaces of the coordination space. They implement their approach for a system of two simulated SCARA-type double-link manipulators. Since the morphology of the learned subspaces is inherently dependent on preplanned paths and their exact sequence, one

learned representation can handle exactly one task sequence in one static environment with no runtime uncertainties. Also, because of the exponential growth of memory required to store the representation of the two coordination subspaces, this approach does not scale well to applications with 6- or 7-DOF arms.

Afaghani and Aiyama ascribe the same importance to online planning capability in their approach to collision-free online coordination of two Motoman HP3J and UPJ manipulators which are controlled individually based on higher-level runtime decisions about required motions (Afaghani et al. 2015). Whenever a new online motion command is processed, the proposed approach treats the currently moving robot as the master and plans a delay time for the execution of the other arm's new motion. This delay is chosen such that no collision occurs. To be able to compute possible collisions in a real-time capable manner, robot links are modeled as line segments. As an extension of the collision-map method proposed by Lee and Lee (Lee et al. 1987), they propose a custom collision checking algorithm which takes into account the robot line segment model and not just the end effector. Based on this, a union of colliding time segments as per the original trajectory is computed and suitable delay times for the start of the free motion are calculated. Upon the execution of the next motion command, the distribution of master vs. free robot is swapped between the arms.

Ögren et al. (Ögren et al. 2012) propose a completely planning-free control-based approach to dual-arm robot coordination in their work. The specific workspace task to be accomplished in the paper is to wash a frying pan held by one arm with a sponge held by the other arm. This task is formulated as a non-solvable optimal control problem which is transformed into a locally admissible control problem that can actually be solved. Collision avoidance between the two arms as well as between one arm and a square-shaped table obstacle present in the workspace is modeled as an inequality constraint on the control inputs and based on minimum end effector distance. Since some of the objective functions of the control problem are time variable, a desired motion can be given as a reference trajectory that is tracked by the controller. The control scheme is evaluated in a Matlab simulation of two 6-DOF Puma 560 robots with a humanoid-like workspace overlap and different values of various controller parameters. The application to a real robot system is not demonstrated. Moreover, the stability of the approach for arbitrary reference trajectories is questionable.

In recent work, Stenmark et al. (Stenmark et al. 2016) describe the process of teaching a dual-arm YuMi robot the task of wrapping a gift box in paper and decorating it. No external tools or algorithms are used in the teaching of the task to tackle the dual-arm coordination problem. Instead, the application is taught entirely via lead-through teach-in and manual coding of the motions in ABB's proprietary language RAPID. The authors cite a number of challenges specifically related to dual-arm lead-through programming. In particular, the programming of

situations with physical contact between the arms required three teaching operators simultaneously. Also, considerable effort had to be spent on designing a correct arm synchronization and keeping up that synchronization during debugging sessions. As a conclusion, the authors find that the currently offered dual-arm system-level teaching approaches do not yet allow a full and intuitive exploitation of features specific to dual- or multi-arm systems. This suggests, that the arm coordination problem needs a good solution in the non-coupled version as discussed in this thesis as well as in the mechanically coupled version which is beyond this document's scope.

Akella et al. (Akella et al. 2004) proposed an approach which takes a set of manipulator paths with velocity profiles and performs uniform velocity scaling operations with mixed integer linear programming. In contrast, we provide more flexibility by allowing velocity profiles to not only be scaled or delayed in time but also to be arbitrarily changed so that they do not necessarily maintain their shape. An extension of the previously described approach was developed by Peng et al. (Peng et al. 2005). They coordinated multiple robots by non-uniform velocity scaling, taking into account kinodynamic constraints. The idea is to identify collision segments along robots' paths and then optimize the velocities based on a mixed integer nonlinear formulation of the problem. In contrast, our focus is not on incorporating a specific method for trajectory generation into coordination planning, but rather to be compatible with various existing approaches to planning minimal-time motions for manipulators.

A hybrid approach that merges two stages of the decoupled approach was proposed by Saha et al. (Saha et al. 2006). The basic idea is to solve the problem incrementally. While planning a certain robot path in configuration space, the algorithm treats robots, for which paths have already been planned, as moving obstacles. In contrast to fully prioritized approaches, the algorithm allows higher-priority robots to tune their velocity along computed paths. In our approach, we do not manually prioritize arms, but rather let the coordination algorithm determine which arm has to slow down when based on a cost function.

Spensieri et al. (Spensieri et al. 2013) presented an approach to the coordination problem in which they assumed that the task sequence is fixed, static and known before the coordination. In this paper, we assume that the task sequence is not known beforehand but instead is incrementally discovered during the execution.

Recently, Kimmel et al. (Kimmel et al. 2016) precomputed coordination spaces for all combinations of tasks for the left and right arms. A sequence of coordination spaces that leads to a minimal-cost path is found. They assumed that, for example, when the left arm is committed to a certain task, the right arm has to wait for the left arm's execution to finish in order to start its own new task. The approach proposed in this paper is more reactive, as it starts to replan immediately after the new task is commanded. Therefore, the maximum delay is not the

Planner	online-capable	Robot model	Real system	Further limitations
(Lee et al. 2014)	no	accurate	yes	–
(Lee et al. 2001)	yes	end effector	yes	–
(Afaghani et al. 2015)	no	line segments	yes	prioritized arms
(Stenmark et al. 2016)	no	none	yes	manual coordination
(Peng et al. 2005)	yes	point/disc	no	–
(Saha et al. 2006)	no	accurate	no	prioritized arms
(Spensieri et al. 2013)	no	accurate	yes	static task sequence
(Kimmel et al. 2016)	no	accurate	no	path-based only
This work	yes	accurate	yes	–

Table 3.1: Comparison of different arm coordination algorithms and their properties.

remaining execution time of the other arm, but rather the time needed for replanning, which is usually much shorter.

In summary, to the best of our knowledge, there is no approach that is online capable and has been shown to work in a real automation environment. In contrast to the described papers, we do not need to know the whole sequence of tasks before starting the motion. Our approach can start execution as soon as the first motion command is received. When motion commands arrive during execution, it refines the current plan accordingly and allows for non-stop robot motion. For a compact overview of the evaluated approaches, refer to Table 3.1.

4 Responsive coordination for industrial manipulator systems

Chapter four of this thesis will introduce the main contribution of the work: an online-capable motion planning and execution framework for dual-arm industrial robots. Some of the ideas presented in this part of the work have been previously published in the following refereed research papers and patent applications:

1. F. Beuke, M. Vorderer, S. Junker, S. Schröck: *Digitale Lösungsansätze für Montagesysteme von morgen.* wt Werkstattstechnik online 9 (2016): 577 - 582.
2. F. Beuke, S. Alatartsev, S. Jessen: *Handhabungseinrichtung mit Roboter sowie Verfahren und Computerprogramm*, patent application, 2017
3. S. Alatartsev, F. Beuke, S. Jessen: *Responsive and Reactive Coordination of Multiple Robot Arms*, patent application, 2018
4. F. Beuke, S. Alatartsev, S. Jessen, A. Verl: *Responsive and Reactive Dual-Arm Robot Coordination.* IEEE International Conference on Robotics and Automation, 2018
5. F. Beuke, S. Alatartsev, S. Jessen, C. Hanel, A. Verl: *Online Motion Planning for Dual-Arm Industrial Robots.* Proceedings of ISR 2018: 50th International Symposium on Robotics, Munich, 2018.

The following discussion will unite the ideas presented here into a coherent picture and provide a more in-depth view of the developments necessary to implement the presented ideas.

4.1 Introduction

One current trend in industrial robotics is to move into the direction of dual-arm robots with anthropomorphic kinematics (Smith et al. 2012; Makris et al. 2017). When automating workspaces previously operated by humans, such a system design often allows all tasks occurring at the workstation to be serviced, while equally-sized single-arm robots can only

handle some of the tasks. Today, dual-arm robots are programmed as if they were two separate kinematics (KUKA Roboter GmbH 2013). As described in 3.3, this makes programming very challenging, as a programmer has to take care that the two arms do not collide with each other. It also leads to high deployment times and inflexible robot code that highly depends on a specific setup. The problem of finding collision-free trajectories for two arms operating independently in a shared workspace is called coordination. Kant et al. (Kant et al. 1986) were first to address this issue in the mid-eighties. Since then, multiple algorithms have been developed (Sánchez et al. 2001; Peng et al. 2005).

Despite a broad algorithmic foundation and multiple dual-arm kinematics commercially available, using both arms concurrently is still difficult in real or near-real applications. For example, during Amazon Picking Challenge (APC) held at IEEE International Conference on Robotics and Automation in 2015, teams were asked to make their robots grasp objects from a shelf. Using two arms is one possible way to achieve quick picking, and the majority of teams (15 out of 23) used dual-arm robots. However, as Correll et al. (Correll et al. 2016) noted, only a few attempts were made to make concurrent use of the two arms in the system despite the benefits of bi-manual manipulation. This points to the existence of a clear gap between available algorithms and their applicability in the real world, which prevents their power from being leveraged. Similarly, our own experience with the industrial application of dual-arm robots is consistent with the conclusions from APC. One of the main reasons for this gap is that existing algorithms are unable to quickly react to the outer world and refine their action plans appropriately. Instead, they are built to produce plans offline that can be used only in static environments (Kimmel et al. 2016; LaValle et al. 1998). However, online-capability has been recognized long ago as an important prerequisite for algorithms operation in uncertain and dynamic worlds (Li et al. 1995).

In the automation of real-world manufacturing tasks, robots often need to start acting upon observing some event, e.g., a sensor signaling that a part is available for pickup. That is why state-of-the-art motion controllers work in an online fashion, i.e., they begin to plan and execute the requested motion instantaneously upon receiving a motion command. One industrial example of a dual-arm application following this paradigm is shown in Fig. 4.1. Here, the dual-arm robot has to use both arms to pick parts from the central source pallet and place them into the outer sink pallets. The logic of the task is computed incrementally by a high-level planner during task execution. This planner reacts to sensory information about the state of the pallets and assigns pick-and-place tasks to the arms independently. The arms have to execute these commands without colliding with each other.

Consequently, the main contribution of this work is an algorithm that can plan and execute motions simultaneously. The solution introduced here targets two objectives that other planners

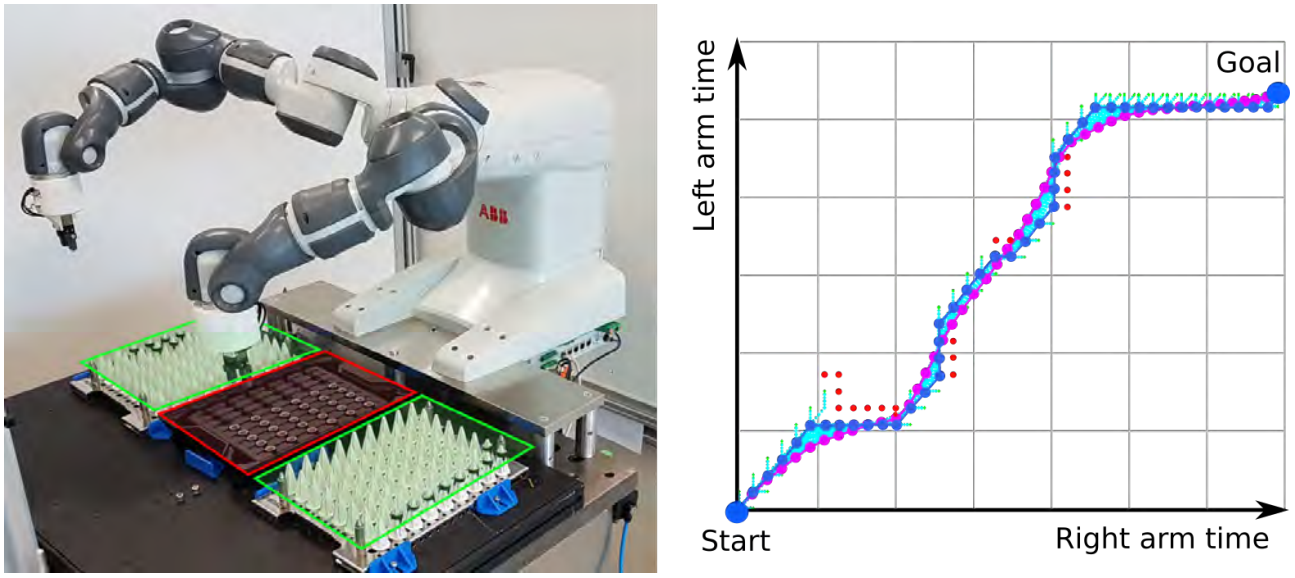

Figure 4.1: **Left**: Industrial dual-arm robot ABB YuMi moves parts from source pallet (red) to destination pallets (green). **Right**: computed initial plan (blue) and smoothed plan (purple) in coordination space that changes velocities of arms to avoid collisions. Colliding states are shown in red.

lack: encapsulating arm coordination in a *responsive* and *reactive* fashion. In this context, we define responsive to mean that the system response time to a motion command is shorter than the time for motion execution. This allows for non-stop motions that will achieve low cycle-times and high throughput. Being reactive means that the system can refine its current plan as new motion commands are triggered during the execution of previous ones. As our approach to coordination is both responsive and reactive, we will further refer to it as R&R coordinator.

The R&R coordinator enables the use of a dual-arm robot as if it were one robot by encapsulating all coordination-related problems away from the programmer. To achieve the stated objectives, it works in both plan and act stages of an architecture based on a hybrid sense-plan-act paradigm (Arkin 2008). Through this, it interleaves coordination planning with real-time motion execution. It builds on underlying ideas for the core arm coordination part that has been applied to several related domains, e.g. in path planning for teams of mobile robots (Bruce et al. 2002) or shortest-path search algorithms on dynamic graphs (Likhachev et al. 2008). However, there has been no viable application of this paradigm to the online-capable coordination of multiple manipulators. Following this paradigm, the R&R coordinator takes time-optimal trajectories planned on the fly and reparameterizes their motion laws to avoid collisions in the current dynamic obstacles situation. Reparameterization is done by finding the shortest path in coordination space (O'Donnell et al. 1989), see the example in the right part of Fig. 4.1. The following sections will provide a detailed overview on the techniques used and the multi-arm demonstrator system developed in the work leading to this thesis.

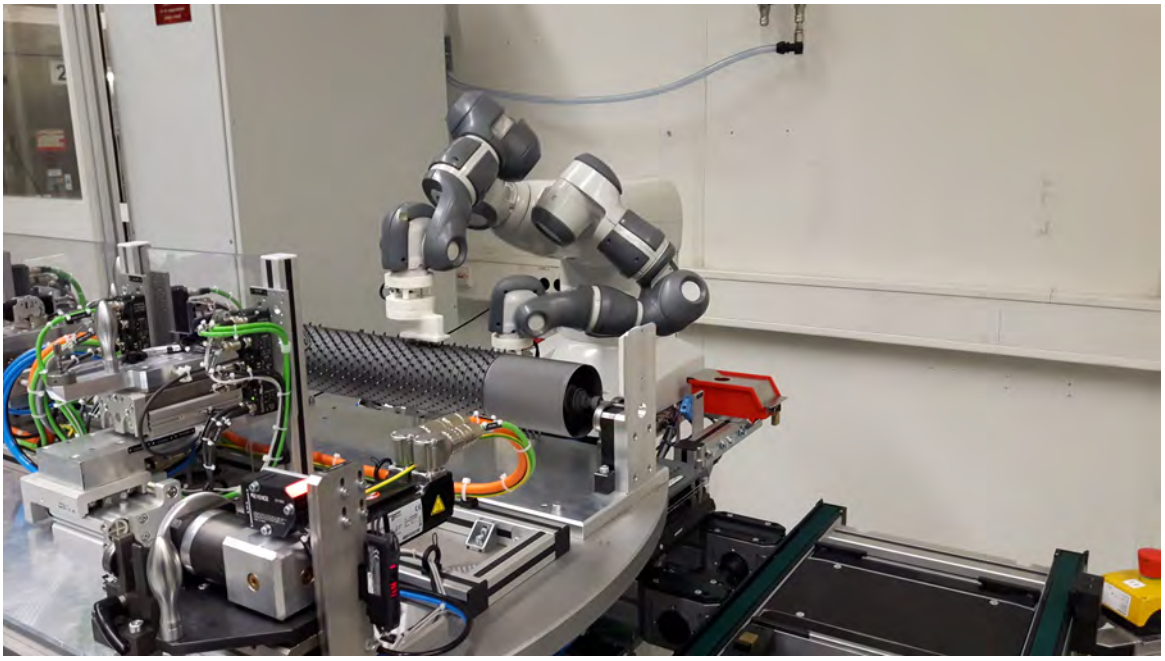

Figure 4.2: A dual-arm robot working on a palletizing task.

4.2 Requirements for industrial applications

Before looking at the details of the proposed solution, this section will derive key requirements for a dual-arm robot motion planning system. Its focus will be on a system operating in a factory under real-life conditions, featuring an example use case inspired by a real application. The robot shown in Figure 4.2 has to place needles from one source pallet into a cylinder-shaped fixture measuring approximately one meter in length. To match the cycle time performance of the human operators previously working on this task, both arms have to pick and place parts concurrently. Uncertainties in this task originate from two different sources. The first uncertainty lies in the fill state of the source pallets. These are not always filled up according to the same pattern. Instead, individual parts can be missing or larger areas of supplied source pallets can be empty, requiring a branch in strategy that is resolved at application runtime.

The second source of uncertainty in the shown application lies in the sink fixture. It is placed under heavy stress during the manufacturing cycle and unforeseeable deformations can cause absolute errors in place positions of up to 15 mm. When placing the parts to the expected positions fail, the robot applies a search strategy of variable duration until the position is found. Both of these uncertainties require the robot to make decisions about appropriate motions on the fly, breaking up any statically defined coupling between the motions of the two arms. Since cycle time has to be minimized it is not acceptable for both arms to react to the uncertainties of only one arm.

The first requirement for a motion planning algorithm for this environment follows directly from the described uncertainty. A successful planner has to be online-capable. That means it must be able to react on the fly to events occurring during runtime, e.g., *"Part not available"* or *"Place position missed"*. It also follows that the planner should not make any assumptions about a task ordering that is fixed and known a priori.

Since the planner must be dynamic in nature, it must also be able to produce plans quickly. When a runtime event like the ones discussed above occurs, a stop in robot motion directly affects the overall cycle time of the application. Although no fixed upper threshold can be given for computation time, the usual time for producing plans should be small in comparison to typical durations of robot motions in the application. The runtime of any (re)planning occurring as a reaction to a runtime event must conclude on the order of tens to a few hundreds of milliseconds, such that its time is negligible as compared to motion execution time.

It is not unusual for the exact geometry of the task setup to vary constantly or at least to change every once in a while. For example, in the given setup, it is likely that the type and exact position of fixtures and pallets could vary. The planner should be general enough to handle these cases. Thus, a further requirement that follows from this is that the planner should not assume any details about the geometry of the task. The algorithm should provide a general unified interface for the supply of necessary data about the planning environment, such as a 3D model or similar.

In the use case shown here as an example, both robot arms operate concurrently and at the same level of priority. No single arm can generally be given priority over the other arm, because the task itself does not incur this. Depending on the sequence of tasks assigned to the arms, it might be favorable for one arm to yield sometimes and for the other time to yield in other instances. Therefore, an automatic planner should not require the user to specify a static priority order for the arms.

The robot seen in Figure 4.2 is operating in a structured world with geometrically complex surroundings. Its arms are mounted in very close vicinity to each other, such that almost all parts of the robot can collide with each other. A successful planner must be able to represent the typical geometries present in such an environment faithfully enough for real use. It cannot use oversimplified robot models (e.g., end effector-only or simple line segments) or workspace models.

As with any automation system, when responsibility is shifted away from the human and over to an automated system, the user loses control over what exactly the automated system does. Since the user can't be held responsible any more, the automated system itself must work robustly under all reasonable circumstances. In the context of an automated motion planning

system as discussed here, this means that all motion commands sent to the individual arms must succeed as long as they lie within the work envelope of the underlying robot kinematic.

4.3 Centralized Coordination Planning

The first step in designing a dual-arm robot motion planner is to layout its overall design. As discussed in Section 3.3, two general paradigms have been applied to multi-robot path planning: centralized planning and decoupled planning. In a separate work pertaining to the research presented here, the applicability of centralized coordination planning to the industrial manipulator domain was evaluated (Müller 2016). Different state-of-the art concepts in centralized planning were assessed to select the most promising path for more in-depth research in this area. Building on the subdimensional expansion framework introduced by Wagner (Wagner et al. 2015), a centralized coordinated motion planner for a dual-arm robot is proposed, implemented and evaluated. The subdimensional expansion framework splits up centralized coordinated path planning into two phases. In a first step, sampling based-techniques are used to generate paths for one arm while ignoring the presence of the other arm. In the second phase, a state space search over the whole composite state space of all arms is conducted. However, as long as no collision is detected, the search over the graph only expands along the robots' predetermined paths. As soon as a collision is detected when following these paths, the search space is grown to the full-dimensional composite space of the colliding robots to find an evading motion.

To reach minimal values for planning time, the centralized planner is implemented on top of a multi-query deterministic roadmap approach representing the connectivity of the statically free workspace[6]. It is applied and evaluated in five different planning scenarios. Depending on the amount and intensity of the collisions present in the individually planned paths, computation times for finding coordinated collision-free paths were found to be on the order of several to several tens of seconds. The main reason for this lengthy planning time was found to be the high dimensionality of the search in collision regions. The conducted research therefore suggests that centralized planning approaches are not yet able to be used inside the replanning loop of a reactive robotic system.

[6]Using such an approach for a high-dimensional system like considered here quickly brings up the question of memory use. Several optimizations were incorporated to obtain a roadmap that would not grow out of the used RAM capacity of 16GB. This memory-runtime tradeoff leans heavily on the side favoring lower runtimes to evaluate centralized planning in a best-case scenario.

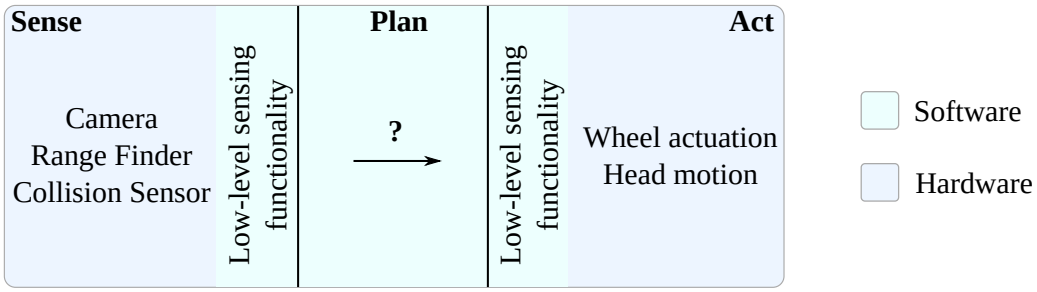

Figure 4.3: Shakey the robot's components classified by functional categories.

4.4 Decoupled Coordination Planning - System architecture

Since the performance of centralized approaches is too slow to take advantage of the benefits of this kind of approach, the developed system has to follow the decentralized paradigm. This chapter provides a detailed insight into the coordination system that has been implemented to satisfy the requirements laid out in Section 4.2. Based on the high-level system architecture shown in Figure 4.6, the single components of the demonstrator system, their implementation and their functionality within the system are described. Particular focus is given to the coordination components, which are at the heart of the coordination algorithm developed in the work that was the foundation of this thesis. The underlying algorithm and details of the implementation are given in the corresponding section 4.6. Understanding the single components is easier when the particular choice of architecture is clear. For this reason, this chapter sets out to give a high level overview of the architectural choice itself and how it fits to the problem that it aims to solve.

4.4.1 Architecture paradigms

Architectural history of software-driven robotic systems starts in the second half of the 1960s with Stanford University's Shakey the robot (Nilsson 1969). As the title of the paper "*A Mobius Automaton: an Application of Artificial Intelligence Techniques*" prominently announces, Shakey was a mobile platform built to propel itself in its environment. To this end, it was equipped with a number of sensors, namely, a TV camera, a triangulation-based range finder and sensors for detecting collisions. To move autonomously, it was able to turn its two wheels either in unison or in opposite sense to move straight or to turn, respectively. It was also able to tilt the camera and range finder mounted on its "head" to enable perception of the desired part of its environment. The hardware present on this platform naturally falls into one of the two categories shown in Figure 4.3. The camera, range finder and collision sensor

Figure 4.4: Scheme of a unidirectional Sense-Plan-Act architecture.

are all designed to perceive certain aspects of the robot's environment and therefore belong to the "Sense" category. The motors driving the wheels and tilting the head are designed to let the robot interact with the environment (e.g., reposition to get a closer look at an object of interest, move an object by pushing it). Therefore, they can naturally be grouped together into the "Act" category. The bare hardware in both categories is accompanied by software responsible for low-level access to the hardware devices (in this case programmed in assembler language), e.g. trigger the camera or turn the motor driving the wheels by a specified amount of steps. To make the robot system as a whole work in a meaningful way, it is imperative to close the central gap shown in Figure 4.3 between the sense and act parts of the system. The makers of Shakey chose to implement a software system that does this in three distinct steps that are executed subsequently and in a fundamentally one-directional way:

1. use the sensor hardware to fill in the parameters of a predefined world model to reflect the current world state,

2. use this world model to decide ("plan") which action to take based upon the current world model, and

3. enact this plan through the actuators.

Since the second of these three steps determines a plan for the supposedly desired robot behavior, this module has been called the Plan module. The resulting architecture fundamentally implements a sequential workflow and has therefore come to be known as the sense-plan-act (SPA) paradigm (Brooks 1986). This architecture essentially decouples the different steps that connect the sensing part of the hardware with the acting parts (see Figure 4.4). Therefore, it is not very well suited for dynamic applications in which the state of the world changes quickly. A plan that is computed based on some snapshot of the world state and then executed blindly may be unsuccessful or even dangerous if the world state changes during plan execution. In the decades after the first conception of Shakey, different architectures were therefore designed with more dynamic environments in mind. However, since sensors and actuators remain to be the fundamental building blocks of robotics systems, the categorization by the SPA scheme is still valid[7].

[7]Note that in other architectural contexts, the terminology for the plan stage might vary. For example, in a typical subsumption architecture (Brooks 1986), the plan stage involves the selection of one of several predefined behaviors based on sensory input and is called subsumption. This, however, does not constitute a principal difference from an architectural viewpoint.

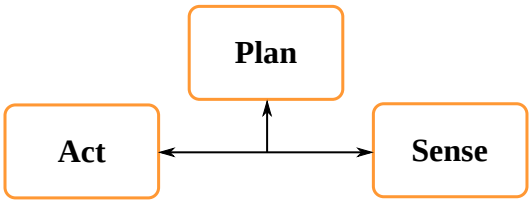

Figure 4.5: Scheme of a hybrid sense-plan-act architecture.

The solution developed in this work solution targets a dynamic world. From the point of view of a single robot arm, the dynamic is caused by the a priori unknown motion of the other arm that may be triggered asynchronously at any time. To deal with dynamic environments, the architecture invariably needs to be able to monitor the state of the world and incorporate the acquired information into the current plan. Planning and plan execution have to run in parallel. Such architectures are called hybrid (Arkin et al. 1997), as they combine a deliberative approach (the high level task-oriented planning) with a reactive approach (the world state-dependent execution of this plan). Systems of this kind typically operate in view of different time horizons (Gat 1998). On the basis of the Sense-Plan-Act paradigm, such architectures can better be depicted as shown in Figure 4.5. In contrast to the sequential architecture used by Shakey and other systems, communication channels in hybrid architectures are bidirectional. This enables the reactive component of the planner to take into account the world state and produce plans that are valid in a dynamic world.

4.4.2 R&R system architecture

Based on these considerations, the architecture driving the reactive and responsive dual arm coordination system developed in this work is shown in Figure 4.6. On a conceptual level, the architecture is divided into the three parts already introduced in the previous section of this chapter, i.e., sense, plan and act. The overall system architecture and the Responsive and Reactive Coordinator together with its function within this architecture form the core contribution of this work. Before these are described in more detail in Section 4.6, the following sections will give a brief overview of the components that use available technology. It will also detail how these components are orchestrated by the architecture.

Sense The Sense part of the architecture unites all the system parts that can supply information about the current state of the world. It contains the 3D sensor and all related software. The scanner detects objects in the robot workspace and delivers a semantic description of the workspace annotated with poses whenever an updated world model is required from the planner. Since processing of the point cloud rendered by the 3D sensor is computationally

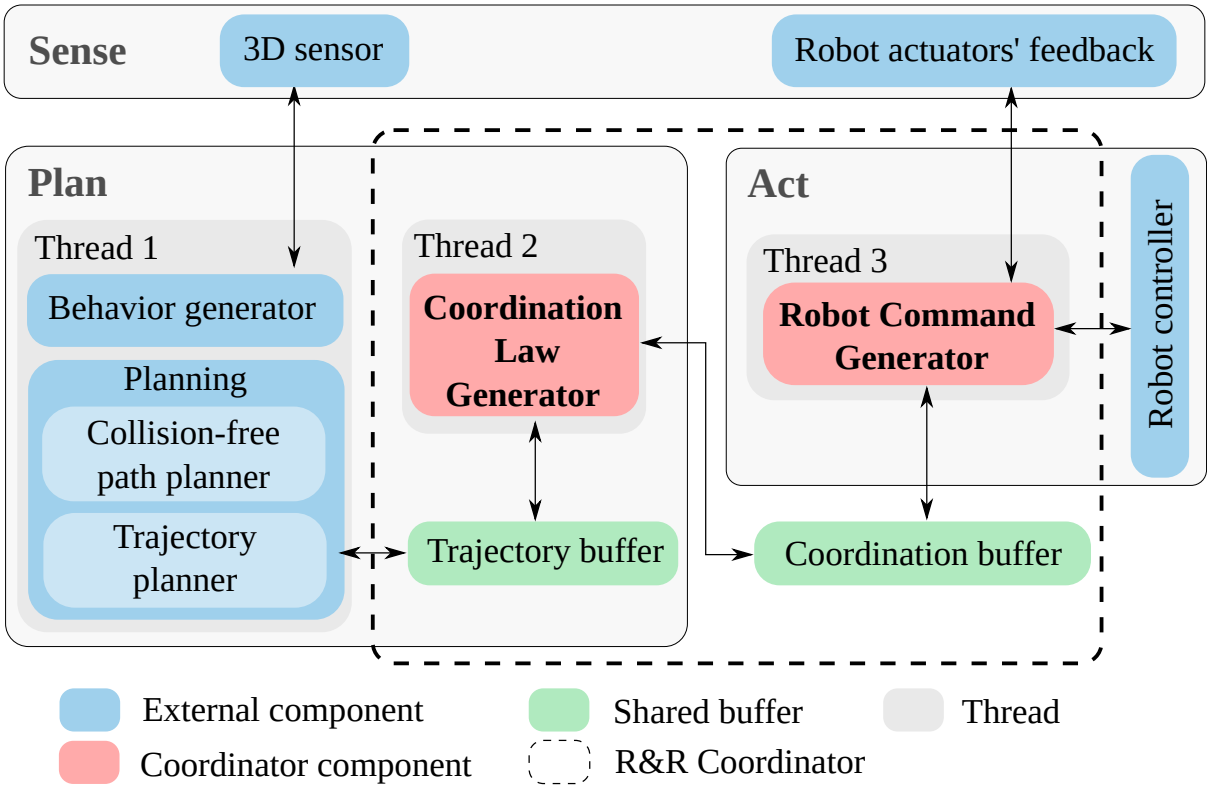

Figure 4.6: The architecture of the whole coordination system shown based on a hybrid sense-plan-act scheme.

expensive, this step cannot yet be executed at a high frequency. Therefore, the 3D sensor input is only used by the deliberative parts of the planner that operate at low frequencies (once per motion command). The sensing part of the hardware is completed by the intrinsic robot sensors. Just like any industrial robot manipulator, the robot system used in the testing and validation of the implementation feature joint angle encoders that measure the absolute position of each of the robot's revolute joints. This completes the measurement of the world state as used in our test setting. In particular, the system is not designed to consider external obstacles that move unpredictably in the robot's workspace, such as humans or other objects that the central planning system is unaware of[8]. However, industrial automation environments targeted by the system developed here rarely allow uncontrolled human interaction with the robot for safety reasons (see Section 4.2). Therefore, this is an acceptable trade-off that enables a better performance of the overall system.

Plan The Plan part of the architecture comprises a larger group of components than the Sense part. At the highest level of abstraction, the behavior generation module determines a high-level action plan for the robot arms. The implementation of this component will be de-

[8]A treatment of this problem for single arm manipulators can be found for example in (Kunz et al. 2010).

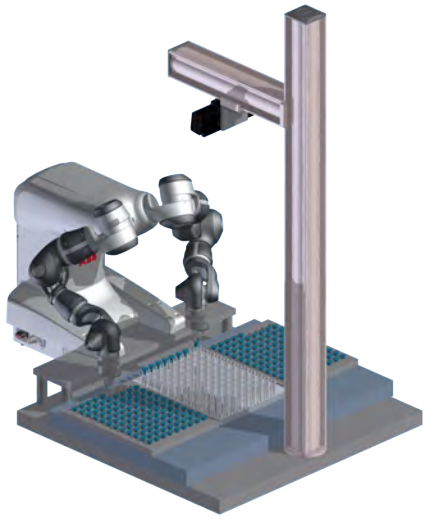

Figure 4.7: A rendered model of the demonstrator setup with pallets and 3D sensor mounted overhead.

scribed in more detail in Section 4.5.2). Based on this computation of appropriate behavior, the command generator triggers the high-level motion commands offered by the motion command interface described in Section 4.6.1. This command stream corresponds to the input stream of motion commands mentioned in the problem statement in Section 1.3. The collision-free motion planning and execution capability offered by the interface is implemented jointly by all the remaining modules present in the system. Some of these modules conceptually belong to the planning part of the architecture. After a motion command is received, a collision-free trajectory is first planned by a geometric path and a trajectory planner (see Section 4.5.3). After the trajectories for single arms have been determined, a coordinated trajectory for the whole system is calculated by the last component in the plan stage, the Coordination Law Generator. In contrast to the other components of the planning stage, this component takes into account information from the Act stage, which is key to reactive planning and execution of tasks. Further details are given in Section 4.6.

Act The architecture is completed by the Act stage, which connects the planning components with the robot control software itself, which runs on a physically distinct robot controller. This connection is handled via the Robot Command Generator, which implements the other end of the bidirectional communication channel connecting the Plan and Act stages. Together, the two components shown in red and their associated data buffers implement the responsive and reactive coordination approach developed in Section 4.6. The Command Generator also handles the communication with the native robot controller, which runs on a physically separate real-time system supplied together with the robot (the robot controller). In the

demonstrator setup described here, the command generator and robot controller are connected by the low-level robot command interface described in Section 4.5.4.

4.4.3 Threading

As already established above, concurrency of task execution and planning are key to implementing reactive systems of any kind. In the architecture proposed in this work, this requirement finds its natural equivalent in the multi-threaded design depicted in Figure 4.6. The frequency at which the three threads run corresponds loosely to the frequency classes of the three-layered architectural framework described in (Gat 1998) – thread no. 1 accommodates the high-level task-oriented and associated path and trajectory planning. Both these tasks run at the relatively low frequency dictated by the computational speed of scene detection and associated behavior and path planning. Thread no. 2 hosts the core coordination law generator shown in red. This component has to run in a different thread because it must be able to communicate with the Act stage at a higher frequency than the higher-level planning tasks. This enables it to obtain up-to-date world states for the dynamic part of the world, i.e., the other robot arm. With this information, it can plan and replan collision-free and efficient coordinated motions. Thread no. 3 supplies joint angle set points to the robot's trajectory following controller at a rate of 20 Hz and must therefore run at an even higher frequency than the coordination planning thread that computed the plan. The necessary synchronization between the threads is provided by the shared buffers shown in green and described in further detail in Section 4.7.

4.5 Components of the demonstrator system

This thesis proposes two main contributions to the state of the art that address the same problem at two different levels of abstraction: On one hand, the proposed dual-arm robot motion interface (Section 4.6.1) aims to provide an easily usable high-level representation of dual-arm robot motion skills. On the other hand, the algorithm for responsive and reactive coordination addresses the low-level problem of arm coordination to enable the proposed interface functionality. To do that, it operates at the level of individual robot trajectories. As shown in Figure 4.6, a range of components are necessary to connect these two layers with each other and empower a working system. While the detailed workings of the components itself are not this thesis' main focus, their integration into a system following the proposed design is a core contribution of this work.

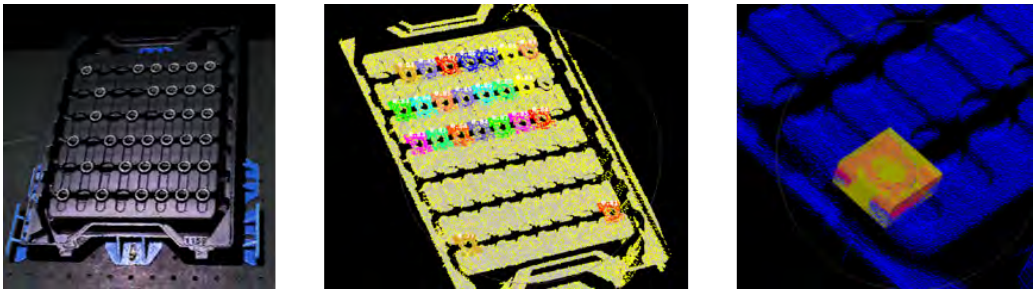

Figure 4.8: Left: 2D image of a palette in the robot's workspace. Center: 3D scan of the pallet with individual recognized parts. Right: Calculated robot gripper pose for removal of a part.

Therefore, the following section provides a detailed description of the choice and application of components used in the implementation of the R&R coordination system. To facilitate a clear distinction between externally developed and internally developed parts of this work, all state-of-the-art components used in the system are covered here. In contrast, the interface as well as the system architecture and the core coordination algorithms are part of the main contributions of this work and therefore described separately in the Sections 4.4 and 4.6.

4.5.1 3D sensor

To enable automatic and intuitive robot programming approaches, the demonstrator system has been equipped with a 3D camera that monitors the robot's workspace and acquires three-dimensional images of the scene. Data processing of these images yields a semantic description and geometric representation of the objects in the robot's workspace. The results of the 3D image-based perception module are illustrated in Figure 4.8. The raw image of the pallet with parts is captured via a stereo camera system which computes a 2D image as well as a 3D point cloud of the workspace (see left image). The geometry information extracted from this data is then analyzed and matched to predefined CAD models of pallets and parts known to the system. As a result of this step, the presence and poses of parts of interest can be determined in workspace coordinates (see center image). Based on a database, suitable grasp poses are then automatically determined and used in subsequent motion commands sent to the robot (see right image). Using this automatically generated description of the world state, the behavior generation module described in the next section can compute meaningful high-level robot behavior, e.g., send motion commands to move parts between detected pallets. Thanks to the underlying automatic planner proposed here, no manual teaching of poses or motions is necessary to execute this robot behavior. Instead, the lower-level motions needed to execute the computed behavior will be triggered via the proposed motion interface. During the planning

process, the exact details of the motions will be planned, such that the actions are executed with both arms in a collision-free manner.

4.5.2 Behavior generation

To solve a given task, the robot must exhibit meaningful behavior. Typically, in today's application of industrial robots, this behavior is hard-coded in the robot motion script as described in Section 3.1.1. In contrast, this work's aim is to provide a flexible robot motion interface for a dual arm robot that can serve as a basis for more autonomous robot behavior. This interface is described in more detail in Section 4.6.1. In a nutshell, it provides a high-level abstraction for the motions of a dual-arm robot that can be leveraged on a symbolical level. This means that it provides appropriate collision-free motions for commands fulfilling the high-level objective of commands such as "*Move the right arm to pose P without colliding with the other arm moving concurrently or with any of the static obstacles in the environment*". These commands provide the necessary abstractions that allow for efficient and simplified programming by a human or by means of other algorithms, such as logical planners[9] (Srivastava et al. 2014).

However, while operating on a much higher level of abstraction than the motion commands provided by robot controllers, the commands provided through the dual-arm robot motion interface still target single motions for single arms. Entire robot behavior such as the one given as an example above still need to be composed from these higher-level motion commands by a human or orchestrated at runtime by an automatic planner. To make possible use cases for the system developed here more intuitively accessible, a component is needed that can combine several single motions into meaningful behavior which the robot can then execute. As mentioned above, the use of logical planners seems to be a promising method to enhance the autonomy of the robot system on top of the interface developed here. The first step towards using such planners is to model the problem in an applicable language, for example PDDL (Fox et al. 2003).

Two actions often needed in industrial tasks are the *Pick* and *Place* actions. To allow for a more intuitive use of the coordination system, these two actions are modeled as an aggregation of lower-level functionality offered by the interface itself. These actions are then parameterized as appropriate by the behavior generator. An example of the modeled skills and their interface usage can be seen in Figure 4.9. A higher-level skill for picking parts from a pallet is composed from a typical sequence of lower-level functions implemented by the interface. Similarly to the PDDL model for state transitions, the skill definition includes three parts. Firstly, it contains

[9]The research leading to this thesis was originally inspired by prior work from this domain (Dornhege et al. 2012).

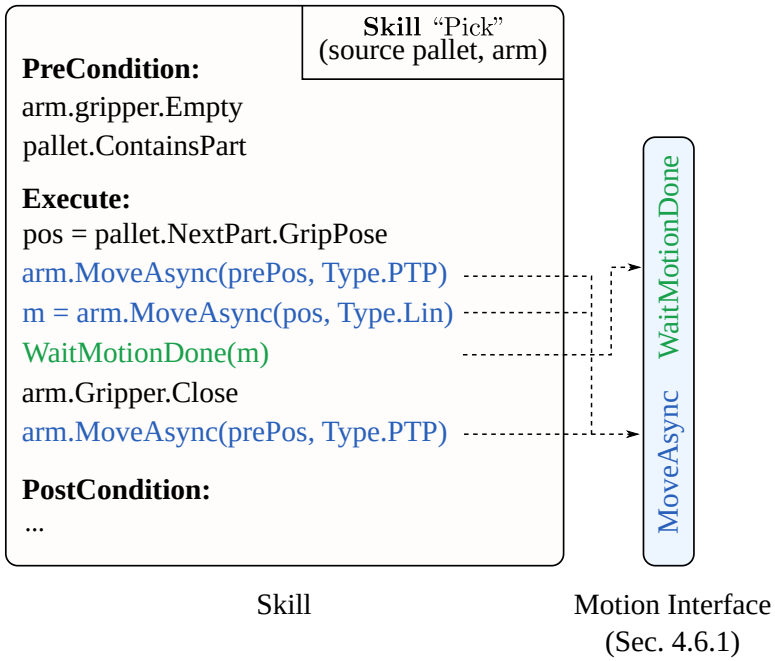

Figure 4.9: Aggregation of interface functions yield a behavior. Here, four interface commands are composed to let the robot pick an object from a pallet.

a precondition which needs to evaluate to true for the skill to be applicable in the current state of the world. Secondly, the body of the skill contains logic and a number of robot motion commands from the interface. Together, this code effectuates the robot's actual interaction with the environment. The third and last part of the skill contains a set of postconditions that become satisfied upon successful execution of the skill.

In the particular case of the grasp skill displayed in Figure 4.9, the pre- and postconditions are simple. The skill can only execute if the gripper is empty and there are parts left in the current pallet. The postcondition code takes care of reflecting the effect of the robot's action on the world by updating the world model. The main body of the skill first computes an auxiliary pre-grasp pose based on the location of the actual part that will be picked. Then, it executes five basic commands, three of which trigger robot motion via the interface designed in this work. First, it commands the targeted arm to the grasp preposition, then, in linear motion, on to the final grasp pose. The program is blocked until successful execution of the second motion command, at which point the robot has reached the final grasp position. The command for closing the gripper is executed and followed by the reverse linear motion that removes the part from the tray. At this point, skill execution is finished and the post conditions are set.

All the triggered motion commands are inherently collision-free with respect to the environment as well as with respect to the other robot arm, which might simultaneously be executing motion

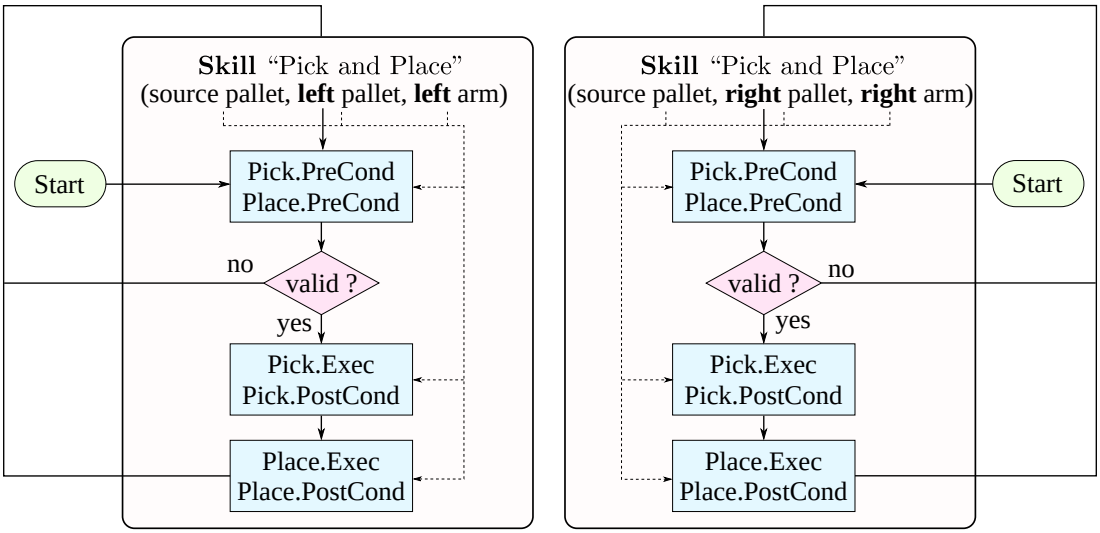

Figure 4.10: The logic of the behavior generator.

commands sent asynchronously, potentially from a different thread or process[10]. Together with a complete domain model and suitable problem definition given in PDDL or another suitable language, these actions could be used by a logical planner to automatically solve problems using both arms of the robot. To keep things appropriately simple for demonstration purposes, the implementation driving the robot in our system does not contain such logical planning. Instead, it implements the workflow shown in Figure 4.10.

The robot's behavior is generated from a composition of the pick skill just described and a place skill that is implemented in an analogous fashion, albeit with different preconditions, postconditions and motion commands. This composite skill is parameterized on all of the parameters of the contained skills. At system runtime, the execution of the composite pick-and-place skill can be switched on and off by user request. Depending on user's choice, the pick-and-place behavior encoded in the composite skill can either execute once or continuously. Regardless of the mode of execution, the skill is called with a new valid parameter set determined based on the semantic world model in every cycle. The same behavior runs in parallel for both the left and the right arm. Therefore, both arms can be independently and asynchronously commanded to start or stop their pick-and-place behavior.

[10]Mechanisms for ensuring consistency of the world model despite this concurrency are in place in the behavior generator to prevent illogical robot behavior (e.g., trying to pick the same part twice). As these are outside of this work's scope, they will not be discussed here further.

4.5.3 Path and Trajectory Planning

As outlined in Section 3.3, approaches to automatic robot coordination can be divided into two subgroups. Coupled approaches on the one hand treat the entire motion planning problem at the same time, while decoupled approaches split it into two or more parts that are solved successively and in a decoupled fashion. In view of the requirements laid out in Section 4.2, coupled coordination approaches are unable to meet the runtime requirements that arise from the dynamic nature of some of the industrial applications targeted by the system. For this reason, our system decouples planning at several points to make the problem more easily tractable computationally. The trajectory planning problem for the whole system is split into separate trajectory planning problems for the individual manipulators. Whenever an asynchronous motion command is triggered via the command interface, an uncoordinated path and trajectory are planned for this motion command. In this context, the word "uncoordinated" signifies that the existence of the other robot arm is not taken into account at all at this stage of planning. In a subsequent planning stage, the preplanned motions are coordinated by the coordination algorithm described later. The following sections discuss the workflow and the algorithms used to solve the first part of the decoupled motion planning problem, before the solution to the core coordination problem in the second planning stage is subsequently described in Section 4.6.

Pose optimizer Motion interfaces of industrial robots typically offer a set of task-space based motion commands. Instead of a fully specified robot configuration, these commands accept a six-dimensional Cartesian pose as goal. Before planning, the corresponding robot joint configuration is determined via computation of the inverse kinematics for the robot. Even in the case of non-task-redundant 6-DOF manipulators, ambiguity exists for many of the possible goal poses in the robot's workspace. Industrial robot motion interfaces require the user to resolve this ambiguity by the specification of extra parameters that fully determine the robot configuration. Often, it is not trivial to find the series of configurations that leads to an optimal sequence of motions. This is even more true when a task-redundant robot is used. To ensure maximum usability of the interface implemented here, task-space based motion commands are also offered (cf. Section 4.6.1). When one of these commands is triggered, the pose parameter does not fully specify the joint configuration. Therefore, the goal pose supplied by the user is converted into a fully specified robot configuration by the methods discussed in Section 2.2.3 before actual path planning starts.

Collision - free path planner Automatically planning a fully coordinated motion for a robot system includes the problem of automatically planning a statically collision-free motion for that system, i.e. a motion which does not cause the robot to collide with any

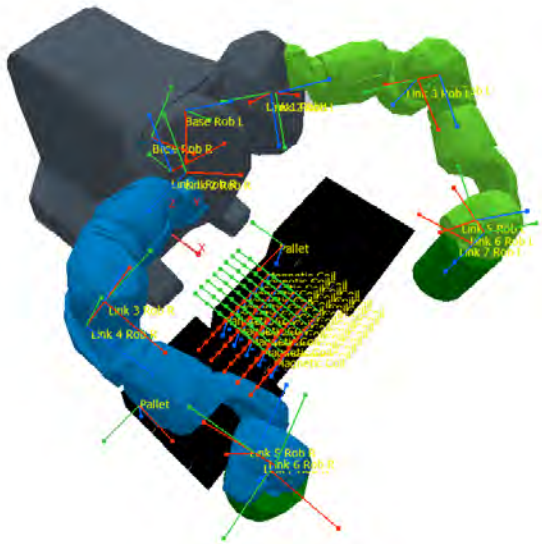

Figure 4.11: 3D geometrical model of the robot and the surrounding scene modeled in the Bullet physics library.

of the static items in its environment. Collision-free path planning algorithms that take into account the geometry of the robot and its environment to plan collision-free paths between given start and goal poses or configurations are not available in any of the state-of-the-art industrial manipulators on the market today. However, in academic research, path planning algorithms exist that can be used to solve the path-planning problem arising in the context of the work presented here. Once a start and goal configuration for the currently processed motion command is known (either through direct specification by the interface user or by computation through pose optimization/inverse kinematics as described above), a path connecting these configurations in a collision-free manner has to be computed. Following a paradigm often applied in the domain of robotic motion planning, we decouple path planning from trajectory planning to make the motion planning problem more easily tractable computationally. In the first step discussed here, a geometric path is determined for the robot to follow. All of the motion dynamics are disregarded at this stage of motion planning. Instead, the problem of computing a dynamically feasible trajectory is deferred to the trajectory planning stage described in the next section. While this does constrain the robot to performing quasi-static motion only, it allows computation times that are acceptable for applications as the ones targeted here.

All of the standard state-of-the-art algorithms that solve this problem (see Section 3.2.1) require a geometrical model of the robot and the surrounding planning scene to determine the validity of a given robot state. The robot model used in the implementation for this work is shown in Figure 4.11. The geometrical models for the individual links of the robot arms are generated from CAD models supplied by the robot vendor. According to the derived D-H parameters given in Table 4.1, the kinematic chains for the arms are modeled and parameterized on the

joint angle variables of the individual revolute joints. To speed up computation, the full-detail meshes generated from the CAD models are decomposed into over-approximating convex submeshes using the volumetric hierarchical approximate convex decomposition algorithm (V-HACD) by Mamou and Ghorbel (Mamou et al. 2009). To determine the validity of a given robot configuration, the geometrical model must be positioned according to that configuration and then checked for collision with itself or any obstacles in the environment. The implementation discussed here uses the Bullet physics library (Coumans 2015) to execute this collision check based on the supplied meshes that represent the scene's geometry.

Collision-free path planning in complex environments is still a computationally expensive operation. This is true even more for more recently developed optimal planners which find a path that is asymptotically optimal with respect to some optimization objective (cf. Section 3.2.1). However, typical computation times of available path planning algorithms can vary substantially depending on the features of the problem (complexity of the workspace morphology, typical computation time for a single collision check, etc.). Therefore, it is not straightforward to select a path planner based on evaluations found in research alone. To select the most suitable algorithm for use in the context of the lab demonstrator, a set of different algorithms was evaluated in a master thesis published separately (Müller 2016).

For use in a robust and responsive planning system, the two evaluation criteria of foremost importance for the selection of a suitable algorithm are the computational speed and the success rate of planning. Since the development of robot path-planning algorithms is not the focus of this thesis, only readily available or easily self-implementable algorithms were considered. To ensure comparability of computational times, all algorithms were tested on the same set of randomly generated planning problems with identical industrial planning scenes and based on an identical implementation of the underlying collision checking algorithm. Since not all of the algorithms yield optimal paths, the output of the non-optimal path planners is post-processed to yield paths of better quality. This step is described in the next section. Since non-optimized paths were usually of unacceptable quality, the computation times needed for this step were included in the comparison of planner runtime. The results of the evaluation strongly suggest the use of the RRT-Connect algorithm for path planning, which for our planning scenes consistently yielded acceptable results in short planning times. For further details of the planner evaluation, refer to (Müller 2016). Based on these results, the RRT-connect planner is used as path planning algorithm in the demonstrator system.

In the approach implemented here, the problem of arm coordination is treated separately from the problem of path and trajectory planning. This means that at the stage of path planning, the existence of the other robot arm is entirely neglected. For any given motion, the start configuration of the robot arm as well as the desired goal configuration computed previously

are passed to a standard implementation of the RRT-Connect planner, which computes a solution in configuration space path as detailed in Section 3.2.1. While our algorithmic choice for solving the path planning problem is motivated by the selection criteria discussed above, in principle any path planner or integrated path and trajectory planner could be used to solve this subproblem.

Path optimizer Since RRT-Connect is not an optimizing path planner, it computes a random path between the given start and goal configurations of the robot. This enables the relatively low computation times satisfying the constraints of the proposed planning system. However, the computed paths are of low quality with excessive path lengths. Post-processing is a way to improve path quality at relatively low computational cost. To achieve better plans, we apply an elastic band algorithm as proposed by Billow and Klette (Li et al. 2007). While this path optimizing cannot find paths in a better homotopy class, it does reduce the noise introduced by the randomly sampled via points considerably. At the same time, since it is only searching for a better path in the homotopy already selected, its runtime is much shorter than that of optimizing path planning algorithms that search the entire $\mathcal{C}$-space for better solutions. This approach delivers the best compromise between path quality and computational speed for the use case scenario laid out in this thesis.

Trajectory planner After the determination of a collision-free and kinematically feasible geometric path for the robot arm, a dynamically feasible trajectory has to be planned along this path. This approach follows the path-velocity decomposition already mentioned[11] (Kant et al. 1986). In the subsequent stage of the planning process, this trajectory will be used as input for the core coordination algorithm described in a later part of this work (see Section 4.6).

The optimized paths generated in the previous planning stage serve as a basis for trajectory planning. The general idea of the approach used in our system is illustrated in Figure 4.12: the path function $q(s)$ (shown on the left side) is reparameterized with a motion law $t(s)$ in such a way, that the composite trajectory function $q(s(t))$ (shown on the right side) and its velocity and acceleration derivatives $\dot{q}(t)$ and $\ddot{q}(t)$ do not violate the given maximum and minimum value constraints for the YuMi manipulator for any t.

One common class of approaches for solving this problem are phase-plane trajectory planning techniques as introduced by Bobrow et al. (Bobrow et al. 1985) and recently developed further by Pham (Pham 2014). However, available implementations of these techniques did not prove

[11] While there are approaches that unite the two steps of path and trajectory planning (Ratliff et al. 2009), their success rate is not yet high enough for our envisioned use case in our experience.

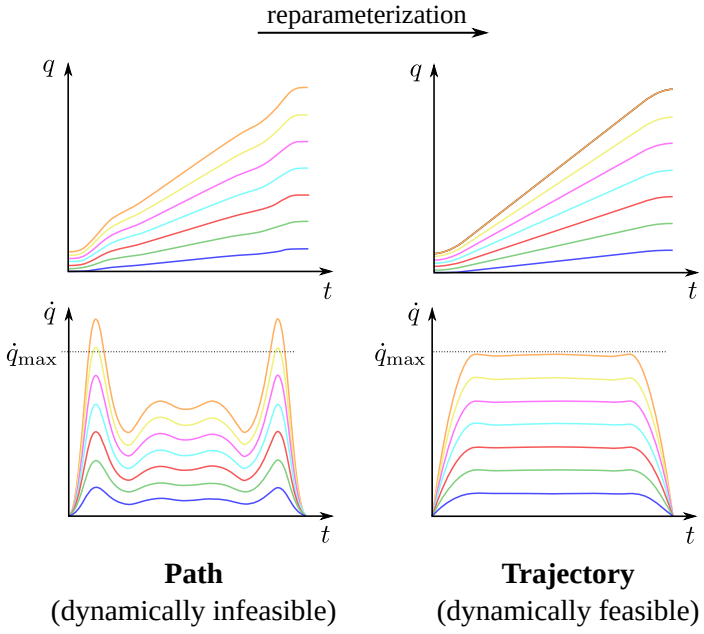

Figure 4.12: Preplanned joint angle paths are reparameterized synchronously, such that maximum joint limit velocities and accelerations are never exceeded.

robust enough for use in the demonstrator system. Therefore, we resorted to using our own implementation of an iterative numerical scaling-based trajectory optimization approach as described by Biagiotti et al. (Biagiotti et al. 2008). In this approach, the planned path is discretized into a number of segments. For each of the segments, a scaling factor is computed based on the ratio of the actual values for $\dot{q}, \ddot{q}$ and the permissible ones. Then, the respective segments are stretched in time such that the maximum values for velocity and acceleration are not exceeded locally. The rest of the trajectory is recomputed to smoothly connect to the reparameterized the segments. These steps are repeated until convergence. For further details on the theory and numerics of the implemented approach, the reader is referred to pages 175 through 198 of the book by Biagiotti et al. (Biagiotti et al. 2008).

4.5.4 Robotic demonstrator setup

The previous part of this chapter has introduced the algorithmic side of the planning pipeline used for the uncoordinated part. In contrast, the following section will describe the hardware side of the setup. Figure 4.7 shows an illustration of this demonstrator and its control infrastructure from a high-level perspective.

In the setup, the dual-arm industrial robot ABB YuMi is mounted in front of a table accommodating different trays with industrial parts. A 3D camera is mounted above the station and connected to the central computer which is running the planning pipeline shown in Figure 4.6

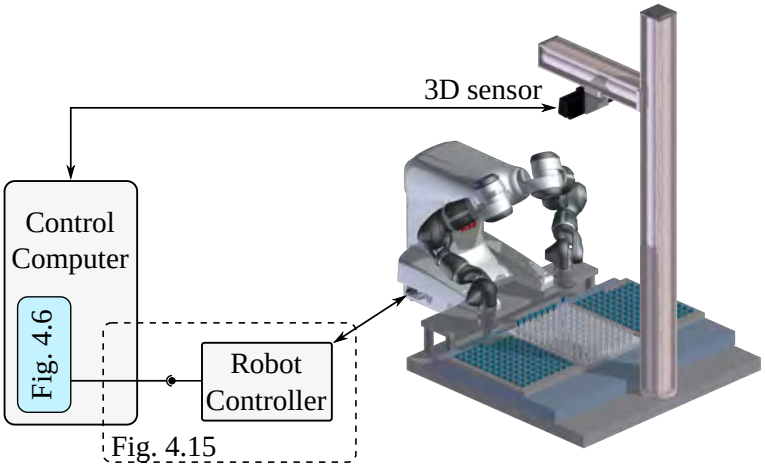

Figure 4.13: A rendered model of the demonstrator setup with its control infrastructure.

and discussed previously. This computer is also connected to the robot controller via a custom-built interface based on TCP/IP socket messaging, which will be described in a subsequent section.

Robot Hardware The core hardware used in the test setup is a YuMi dual-arm industrial robot by ABB Automation. This device combines two 7-DOF revolute-joint kinematic chains mounted on a common body. The arms are positioned in a way that mimics the configuration of human arms. The distance of the shoulder joints is therefore very small. This leads to the fact that the overlapping parts of the workspace constitute a very large part of the total workspaces of the arms. Each of the arms can carry a rated load of 0.5 kg and the geometrical dimensions are similar to those of a human arm (see Figure 4.14). However, the arms are not symmetric to a central plane. Instead, the same kinematic chain is mounted on the body twice in the position of the left and right arms. Other than that, the positioning and sequence of the joints itself is kinematically similar to the mechanics of the human arms. The low arm power and the soft padding around the links allow the robot to work in the vicinity of humans without being fenced off by a safety cage. This is a big advantage in the development of a lab setup that requires frequent and direct access to the robot.

As explained in Section 2.1, kinematic calculations require a mathematical description of the kinematic chain of the robot manipulator. The appropriate DH parameters (see Section 2.2.1) were derived based on the 2D and 3D robot models in Figures 4.14 and 4.7. The values are given in Table 4.1.

Low-Level Robot Control Interface Industrial robots are usually controlled by a dedicated computer, i.e., the robot controller, which forms an integral component of any

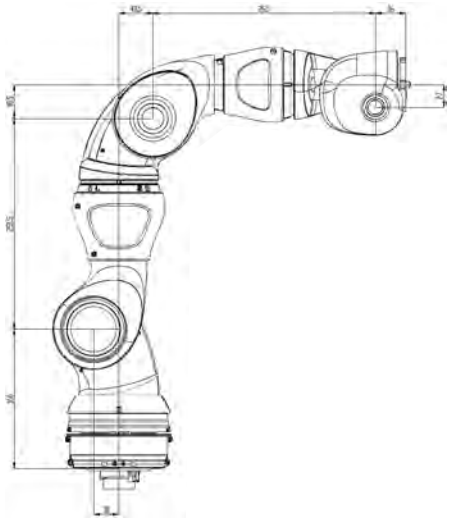

Figure 4.14: Mechanical design and dimensions of the YuMi manipulator.

Link	r	d	α	θ
1	-0.03	0.166	$-\frac{\pi}{2}$	0
2	0.03	0	$\frac{\pi}{2}$	0
3	0.0405	0.2515	$-\frac{\pi}{2}$	0
4	-0.0405	0	$\frac{\pi}{2}$	$\frac{\pi}{2}$
5	0.027	0.265	$-\frac{\pi}{2}$	0
6	-0.027	0	$\frac{\pi}{2}$	0
7	0	0.036	0	0

Table 4.1: D-H parameters of the YuMi manipulator.

robot. The controller contains the robot program, i.e., a script written in the applicable robot programming language which depends on the manufacturer of the device (see also Section 3.1.1). Besides the possibility of uploading scripts to the robot controller and executing these, industrial robots do not usually offer any other interface to robot motion functionality[12]. For this reason, custom online control of the ABB YuMi can only be achieved via a custom interface that directly targets the motion commands natively offered by the robot controller. This interface has been implemented as part of a separate work pertaining to this thesis (Scheitenberger 2015). It connects the control computer with the robot controller (see Figure 4.7) and will be described in the following section.

The objective of the interface is to enable direct access to a subset of the motion commands natively offered by the robot controller and to allow the tracking of robot motion execution. For this, the interface offers two core functionalities, shown here by way of example with the functions

1. MotionID MoveGoal(Config L, Config R) and

2. WaitMotionDone(MotionID id)

The first function accepts a robot configuration for the left and right arms respectively and moves the robot arms to the commanded position. Since the motion command is executed

[12]For this reason, all of the offline programming tools discussed in Section 3.1.3 generate script code and upload it to the robot controller. This can possibly be done in a matter of seconds. However, this programming mode is essentially still an offline mode, because several seconds are a very long reaction time when compared to typical robot motion durations. Also note that some manufacturers like Stäubli do offer low-level interfaces to their robots. Devices of this type were not available for evaluation in this thesis.

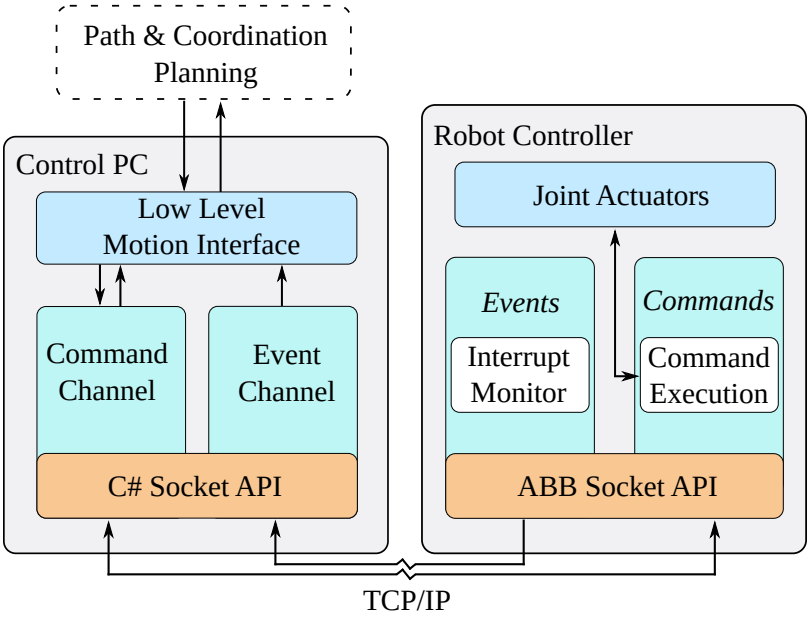

Figure 4.15: Low-level command interface architecture

asynchronously, the function return an identifier for the commanded motion. This can be used, for example, to block program execution until successful completion of the motion via the supplied function WaitMotionDone.

The software implementing this low-level robot command interface consists of two components. One of these runs on the control computer and the other one runs on the robot controller. Both components and the associated communication channels are shown in Figure 4.15. The client component of the interface is implemented in C# and encodes the received commands into strings that are then sent via a TCP/IP connection to the component running on the robot controller. The robot controller side of the interface implements a server in ABB's native RAPID on top of the socket API supplied by the robot controller. This server listens to string messages sent to it by the client. These messages are decoded and mapped to an interface that executes native motion commands on the robot system. To do this, the server side software parses the input string and triggers the appropriate command on the robot controller. In the use case shown in this work, only fully synchronous absolute joint motion commands are needed, since all higher-level logic is computed by other parts of the control software. Therefore, the only native motion command targeted by the interface is RAPID's MoveAbsJ (cf. Section 3.1.1). To ensure temporal synchronization of the executed commands across the separate robot arms, ABB's synchronous motion functionality is used via SyncMove (for more detail, see Section 3.1.1). The execution state of the motion is tracked and successful execution is communicated back to the client by a string message sent via a separate asynchronous event channel.

4.6 Achieving Responsive and Reactive coordination

After the overall system setup and the pre-coordination planning pipeline have been introduced in the previous sections, the following section will describe the proposed R&R coordination approach, which is at the heart of the system architecture for computing and executing dynamically collision-free motions as described in 4.4. The discussion is split into three logically separate parts. The first section sheds light on the proposed motion interface that enables automatically coordinated use of the two robot arms and thereby provides a device-level abstraction for the dual-arm robot. The following two sections detail the approach used in the implementation of the offered functionality. In the first of these sections, entitled *Offline Coordination*, the core techniques used to achieve automatic coordination for the arms given an offline environment with a known motion sequence are discussed. The subsequent section, entitled *Online Coordination* then demonstrates how these techniques can be applied to a real robot system to achieve online-capable collision-free motion execution without a motion sequence that is known a priori. With these two ideas combined, our approach enables online use of the dual-arm motion planning system. With this functionality in place, a dual-arm system can be used in the same way as two single arms.

4.6.1 Motion Command Interface

The motion command interface is the glue between high-level reasoning and abstract motion planning on the one hand and low-level, concrete motion planning and execution on the other hand. Of all the functionality described in this section, this is the only part that is exposed to an outside user. As such, it represents all the functionality proposed in this thesis in a nutshell. Referring back to the overview of the system design shown in Figure 4.6 at the beginning of this chapter, Figure 4.16 illustrates the conceptual role of the motion interface within the proposed system. This interface offers an API encapsulating motion functionality for the robot system as a whole. On the abstract side, the interface can be used either directly in a high-level programming language (currently C++ or C#), or can itself be the low-level component in some higher-level abstraction of the robot system[13].

The motion interface contains two types of commands: firstly, a set of commands for triggering motions and secondly, one command for monitoring the execution state of the triggered motions.

[13] A rudimentary higher-level abstraction on top of this motion interface has been implemented using the skill approach described in Section 4.5.2.

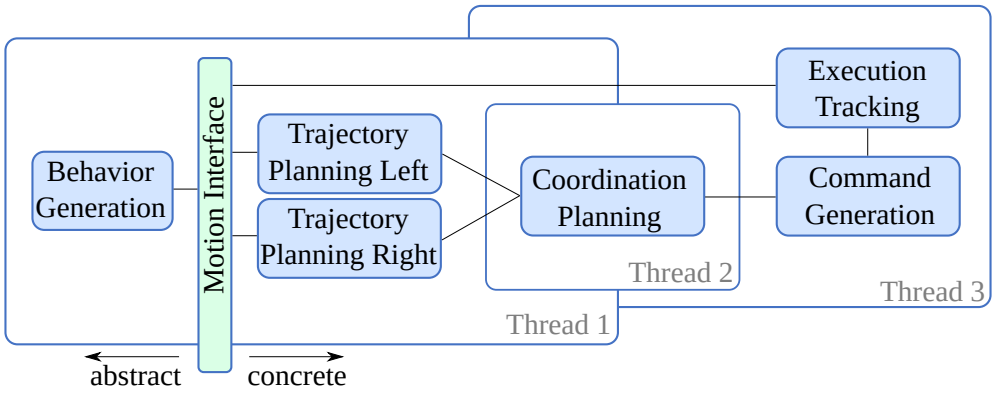

Figure 4.16: Position of the motion interface in the system design.

The structure of the motion commands is illustrated here using the example of the command for point-to-point motion of one arm in the C++ API [14]:

```
MotionResult MoveAsync
(
  RobotArm     arm,
  Pose      targetPose,
  double       velocity,
  double       acceleration,
  Termination tType
);
```

Listing 4.1: Structure of the MoveAsync command.

The command MoveAsync triggers an asynchronous point-to-point motion of the specified robot arm to a given goal position. It takes a parameter list that specifies the details of the motion and returns an object of Type MotionResult, which can be used in subsequent API calls. To get a better understanding of what exactly this command does, here is a closer look at its parameters:

RobotArm This parameter specifies the arm to be used in the motion. For the dual-arm system presented here, this can evaluate to either the left or the right arm. For systems with more than two kinematic chains, any single one of these chains can be selected via this parameter.

Pose The 6D Cartesian pose in the robot's workspace that the tool center point should reach at the end of the motion. This value is used as the basis for pose optimization and then converted to a robot goal configuration via inverse kinematics computation.

[14] This part of the work covers only the important concepts which an interface user should be familiar with. A complete documentation exists in a separate document.

velocity A scaling value in the interval 0 . . . 1. The maximum permissible joint velocities are reduced by this factor. This allows the specification of slower motion speeds.

acceleration A scaling value in the interval 0 . . . 1. The maximum permissible joint accelerations are reduced by this factor. This allows the specification of motions with reduced accelerations.

TerminationType Specifies whether the robot should come to a full stop at the end of this motion or smoothly transition to the subsequent motion command (if that command is received in a timely manner).

When comparing the structure of this motion command to the motion commands offered by industrial robots today, the similarities are obvious. This is no coincidence, but the embodiment of the very goal of our interface design, which is to allow the use of a dual-arm robot system as if it were one single arm and encapsulate all the complexity introduced by the close positioning of two manipulators.

However, despite the structural similarity there are a number of notable differences in functionality which enable much easier use of the dual-arm robot system as compared to traditional interfaces.

1. The motions are collision-free with respect to the geometric model of the robot and the objects in the environment. Collision-free motion in obstructed environments can be executed by specifying no more than the desired goal position of the motion. In contrast, a corresponding traditional motion command would lead to collision with objects in the workspace if not carefully designed to avoid these collisions.

2. The motion command can be triggered for any arm asynchronously, i.e., without explicit synchronization with commands sent to the other arm. The executed motion will be collision-free with respect any motions concurrently executed by the other robot arm. This is the core coordination functionality enabled by the algorithmic approaches laid out in the subsequent parts of this chapter. When compared to the classical robot motion command, the command offered by the interface described here offers much more flexibility to the user. To achieve temporal motion coordination in classical robot programming, a command to one arm always has to be accompanied by a corresponding command to the other arm. Both commands will execute concurrently and end at the same time. This obliges the interface user to specify an explicit synchronization, i.e. an arm motion can only be specified by also specifying the exact motion of the other arm at the same time. For many applications, this is impossible, because there is no inherent synchronization of the arms which arises from the task definition. Furthermore, if the commanded paths

happen to intersect, the robot arms will collide with each other even though the motions are temporally synchronized. Therefore, any combination of motion commands has to be designed such that the resulting motion is collision-free under all circumstances. Since the start positions for these motions are the current positions of the arms, this task is far from trivial.

In contrast, the motion command MoveAsync described above requires neither a corresponding concurrent motion command for the other arm, nor does the interface user have to worry about possible collisions. All this complexity of the motion calculation is handled internally by the described algorithms.

3. Traditional-style motion commands require a complete specification of the robot goal pose. Even for 6-DOF-robots, this means that for any given goal pose used in a motion command, the user has to also select the wanted configuration from among the up to eight possible ones to remove the ambiguity from the inverse kinematics calculation. In the case of a 7-DOF robot as considered here, the set of configurations to choose from is infinite. This puts the burden on the user to select such a sequence of configurations that enables best motion performance. Sometimes, a wrong combination of configurations can even lead to a sudden stop in robot motion. In contrast, the family of motion commands described here allow the user to specify just the desired Cartesian goal position and requires no configuration specification, which further simplifies the use of the robot system.

In combination, these three major differences in functionality allow a dual-arm robot to be programmed and used much more intuitively and with less expert knowledge, thereby enabling more robust robot programs, shorter application times and thus higher flexibility of the device. Besides the MoveAsync command, the motion interface offers a number of additional commands for executing robot motion. These are similar to the described command in functionality but offer a slightly different user interface. For example, some commands accept an arm configuration instead of a Cartesian pose, so that the user can specify the exact robot position if desired. Others compute linear instead of point-to-point motion. However, since this thesis' main focus is arm coordination, the underlying functionality will not be described here in detail.

The only interface command which does not belong to the group of motion commands is the command

```
void WaitMotionDone(MotionResult mResult);
```

This command does not trigger the execution of a robot motion. Instead, it will block execution of the caller until the motion command call which created mResult has been successfully planned and executed on the robot. This functionality is an essential component of the interface, which allows to synchronize arm movement with other logic of the task, e.g. closing a gripper when the robot has reached a gripping pose. To illustrate the necessity of this functionality, consider this code snippet from the implementation of the grasp skill described in Section 4.5.2.

After calling the motion command which positions the robot to grasp the desired object, execution is blocked by the command WaitMotionDone until the corresponding motion is completely executed. Only then, program execution continues and the signal for closing the gripper is sent. If no such command were present, it would not be possible to synchronize task logic with motion execution[15].

4.6.2 Offline Coordination

Whenever a computational problem is too complex to be solved all at once, one possible approach to finding a solution is characterized by the divide-and-conquer paradigm – break up the problem into smaller and less complex subproblems until the resulting problems can be solved, then reconstruct the solution to the complete problem from the partial solutions to the subproblems. This strategy is the foundation of a number of solutions to problems discussed here already (e.g., in task vs. motion planning and path vs. trajectory planning)). Although it comes with the problems of narrowing down the space of possible solutions, it is applied successfully in all of these instances. As far as the problem of coordinated motion planning for multiple manipulators is concerned, our experiments discussed in Section 4.3 have shown, that a treatment of the integral problem is computationally infeasible for the domain of industrial manipulators. Therefore, the complexity of the problem needs to be reduced. To do that, our approach to dual-arm robot coordination follows the divide-and-conquer principle. It does not treat and solve the proposition of robot motion planning as one integral problem, as this approach was shown to be inadequate from a viewpoint of computational complexity. Instead, the composite planning problem is broken down into several subproblems, which are treated subsequently.

More specifically, in our approach, a solution to the arm coordination problem is found at the trajectory level. This means, that most of the subproblems in the motion planning problem

[15] As an extension, it would also be possible to block the calling thread until motion execution has concluded to some specified degree instead of completely. This is a feature that many industrial robots offer to allow more fine-grained synchronization of task logic and robot motion.

chain (i.e., behavior planning, path planning, trajectory planning) have already been solved when the proposed arm coordination algorithms begins its work. The motivation for the chosen way of breaking down the problem and the solutions implemented to address the upstream subproblems have been briefly discussed in the previous sections. The result of these planning algorithms is a set of two trajectory functions $q_L(t_L)$ and $q_R(t_R)$ which fully determine the arm motion. As these trajectory functions are the result of the upstream collision-free planning pipeline, the corresponding motions are already free of collisions with respect to any static obstacles that can be found in the scene. Up to this point, the motion commands make the system behave like an ordinary industrial robot, although with the added functionality of automatically avoiding static obstacles defined in the geometric model used by the planner. However, the computed motions do not yet take into account the presence and dynamic motions of the other arm. Because of the close geometric proximity of the arms, a naive execution of the independently computed robot plans will almost certainly lead to collision between the two arms.

To stop this from happening, we propose to re-parameterize the trajectory functions q with two functions $t_L = c_L(t)$ and $t_R = c_R(t)$, such that the resulting coordinated trajectories

$$q_L(c_L(t)),\ q_R(c_R(t)) \tag{4.1}$$

are collision-free. We call the functions $c_L(t)$ and $c_R(t)$ *coordination laws* for the left and right arms respectively. In its mathematical description, this reparameterization is equal to the reparameterization of the path function carried out during trajectory planning for a given path (as discussed in Section 4.5.3). However, finding an optimal coordination law is conceptually different from finding an optimal motion law, because the constraints on the reparameterization functions are different. In the following section, we will show how we conduct the search for the coordination law functions.

The trajectory functions for the left and right arm depend on the scalar parameters t_L and t_R respectively. These parameters can be understood as a measure of trajectory completion. Since the robot motions are planned independently of each other, these parameters are independent as well. However, not all possible combinations of trajectory parameters are valid – some combinations place the robot arms at such points along their predefined trajectories that lead to a collision between the arms. It is therefore necessary to remove the independence of the parameters and find a valid coupling of them. Such a valid coupling is given by a function $t' = c(t)$ with an image that contains only valid states (t_L, t_R). To find such a function, we consider the coordination space sketched out in Figure 4.17. First, let us revisit the definition of some concepts that are important for the understanding of this approach:

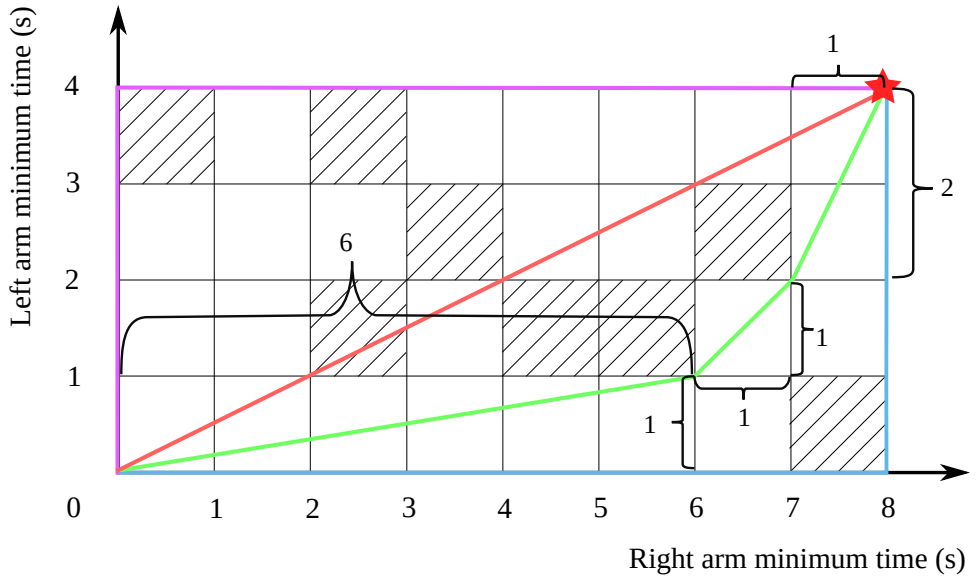

Figure 4.17: An example of the coordination space for a dual-arm robot. **Purple:** left arm moves first, then right arm. **Blue:** right arm moves first, then left arm. **Red:** both arms move with collision (shaded areas). **Green:** coordinated simultaneous motion.

Formal definitions *Configuration space* (C-Space or $\mathcal{C}$) is the space of robot joint values, i.e., $\mathcal{C} = \mathbb{R}^n$, where n is the space dimensionality and equal to the number of robot DOFs.

A *configuration* q is a point in configuration space, i.e., an n-dimensional vector with joint angle entries for each of the joints. The dimensionality is equal to the numb

A *path* ρ is a continuous function that maps a path parameter $s \in [0, 1]$ to a point in $\mathcal{C}$, i.e., $\rho : [0, 1] \rightarrow \mathcal{C}$.

A *motion law* μ is a mapping of a time from the domain $[0, T]$ to a path parameter $s \in [0, 1]$, i.e., $\mu : [0, T] \rightarrow [0, 1]$, where T is the motion duration. A composite function of robot path and motion law μ is called *trajectory* τ. It describes a motion that can be executed on a real system, i.e., $\tau(t) = \rho(\mu(t))$, where $t \in [0, T]$.

A *coordination law* σ is a continuous mapping of a time value t_r from the interval $[0, T_r]$ to the parameters of a tuple of m independent trajectories $(\tau_1, ..., \tau_m)$, i.e., $\sigma : [0, T_r] \rightarrow (t_1, ..., t_m)$, where $t_i \in [0, T_i]$, $i = 1...m$ and T_i is the duration of the i-th uncoordinated trajectory and T_r is the duration of the coordinated motion. The coordination law couples the otherwise independent trajectories into a synchronous motion depending on one single parameter, i.e., real time.

The coordination problem is to find a coordination law that leads to a collision-free trajectory for the whole system. When the coordination law is found, it is applied to the precomputed trajectory in order to obtain a *coordinated trajectory* τ_c.

The coordinated trajectory is a composition of a tuple of trajectories $(\tau_1, ..., \tau_m)$ and a coordination law σ:

$$\tau_c = (\tau_1(\sigma_1(t_r)), \ldots, \tau_m(\sigma_m(t_r))). \tag{4.2}$$

Since $\tau(t) = \rho(\mu(t))$, Equation (4.2) can be rewritten in the following way:

$$\tau_c = (\rho_1(\mu_1(\sigma_1(t_r))), \ldots, \rho_m(\mu_m(\sigma_m(t_r)))). \tag{4.3}$$

Therefore, a coordinated trajectory defines a collision-free motion for the whole robot system, i.e., $\tau_c : [0, T_r] \rightarrow \mathcal{C}^{m \times n}$.

The coordination search is done in the *coordination space* $\mathcal{S}_c$ of m robots, which is the Cartesian product of the m trajectory domains, i.e., $\mathcal{S}_c = [0, T_1] \times \cdots \times [0, T_m]$. For the special case of the dual-arm problem treated here, $m = 2$. In general, a coordination space can be formed with different parameters, e.g. path length parameters or trajectory parameters. In our proposed approach, the trajectory parameters t_L and t_R form the dimensions of the space. Our motivation for making this particular choice over other possible choices such as path length (see (Kimmel et al. 2016)) will become clear later. Since the robot motion is encoded in the axis through the trajectory parameter, any given state in the coordination space represents a snapshot along the preplanned trajectories of both robots. As such, it can either be collision-free and therefore valid, or cause a collision and therefore be invalid. As in the path planning problem, the free and obstructed subspaces are known only implicitly through the collision-checking function, see (3.1). For reasons of visual simplicity, the subspaces are represented in Figure 4.17 by white and shaded squares respectively – their actual shape differs and depends on the specific trajectories encoded in the axes.

Describing coordination plans Assuming the start state is given by the current robot state and the desired goal state lies at the end of the currently planned trajectories for the arms, a valid coupling of the parameters can be described by a curve which connects start and goal states and lies entirely in free space. Some valid and non-valid curves are shown in Figure 4.17. Each of the curves represents a hypothetical coordination plan. When following the purple plan, the left arm will execute its trajectory first and reach its goal. Only then will the right arm start to move until it also reaches its goal and the system reaches the desired

overall goal state. When following the blue plan, the order of arm motion is reversed: the right arm will move first while the left arm stands still, and vice versa. These plans are trivial in so far as the robot's arm motions are not coordinated in a narrow sense, as only one arm is moving at any given moment. The plans can only succeed if the arms happen to not block each other's way as one of them remains motionless in its start pose. The remaining two plans are different in so far as they represent concurrent arm motion. If the red plan were executed, both arms would move simultaneously. However, this plan is invalid as it crosses collision areas. The green plan causes concurrent arm motion as well. Moreover, it connects the start and goal poses passing through the collision-free subspace only. Therefore, it doesn't cause a collision between the arms and represents a valid coordination plan. When executing this plan, the right arm would move faster than the left arm at first, then both arms would move at the same speed, while eventually the left arm would move faster so that both arms would reach the desired system goal state at the same time. Of course, real coordination plans should not be only C^0-continuous as the ones shown here, as this would lead to infinite accelerations and forces that cannot be exerted by physical actuators. In a smooth coordination plan as the one shown in Figure 4.18, execution speeds will change gradually, such that the robot actuators can actually follow the computed plan.

Optimality From the considerations in the previous paragraph, it is intuitively obvious that some plans are better than others. For example, letting one arm wait until the other one finishes its motion might lead to a valid plan. It also potentially wastes a lot of time. Since industrial robotics is often concerned with cycle times and execution speeds of a given automation task, an avoidable waste of time is unacceptable. T It would be considered much better in this case to let both arms move simultaneously and therefore finish more quickly, as encoded by the green plan. To allow an algorithm to take this into account, the concept of execution time has to be incorporated into the cost function considered during the search for a solution. This is why we choose to form the coordination space by use of the arms' individual trajectory parameters. This immediately allows us to incorporate execution time into the cost function. Consider for example the coordination plan shown in purple: Its execution would take twelve seconds, i.e., four for the left arm's motion and another eight for the following motion of the right arm (note that this assumes robots reach their target execution velocity instantaneously, with no time spent accelerating or decelerating). In contrast, executing the green plan under that same assumption would take only 9 seconds. To arrive at this value, let us consider the three linear segments of that path. During the first segment, the robot's right arm covers six seconds worth of trajectory. Since the planned trajectory is already time-optimal for the given piece of underlying path, the robot will need at least that amount of time to reach the end of the first segment. Following the same logic, the other two segments will last at least one and

two seconds, respectively. Summing up the execution times of the segments yields the mentioned total duration of 9 seconds. As in the previous case, this assumes that the associated accelerations and decelerations are instantaneous.

Generalizing these considerations, a suitable cost function measuring the quality of a coordination plan can be defined: when optimal trajectories are used as parameters of the coordination space axes, shorter path yield shorter execution times and therefore better plans. As noted above, other authors propose the use of other variables to form coordination space (e.g., path length as proposed by Kimmel et al. (Kimmel et al. 2016)). However, the problem with such a formulation is that the concept of execution time is missing from the problem statement, and the assumption that path length corresponds to execution time does not universally apply. It is not necessarily true that similar path lengths can be covered by different manipulators within similar execution times. Consider for example the case, where two arms of different size and with different actuation systems need to be coordinated. In this situation, it is very likely that execution times for similar path lengths would differ considerably. Thus, search in a path length-based coordination space would fail to find a coordination plan with low execution time, because information about execution times is absent from that space in the first place.

From coupling to coordination law Until this point, when describing coordination curves, we talked about robot motion in terms of being executed *faster than* or *slower than* the other arm's motion. This is because the coordination curves shown in 4.17 describe only one aspect of the arms' motions: only the relative execution speeds of the robots can be determined directly from the slope of the coordination curve. For example, consider a straight line in coordination space which runs at an angle of 45 degrees between the axes. When following this line, both robot arms would execute their trajectories at the same speed. If the line's slope is 30 degrees counting from the axis of the right robot arm, the left arm's execution speed would be

$$\frac{\mathrm{d}t'_L}{\mathrm{d}t'_R} = \tan(30) \Leftrightarrow \mathrm{d}t'_L = \tan(30)\mathrm{d}t'_R = \frac{\sqrt{3}}{3}\mathrm{d}t'_R \approx 0.58\mathrm{d}t'_R, \tag{4.4}$$

meaning the arm would follow its trajectory at only about half the execution speed of the right arm. The opposite situation is characterized by a slope of 60 degrees, along which the ratio of execution speed would be

$$\frac{\mathrm{d}t'_R}{\mathrm{d}t'_L} = \cot(60) \Leftrightarrow \mathrm{d}t'_L = \frac{\mathrm{d}t'_R}{\cot(60)} \approx 1.73\mathrm{d}t'_R, \tag{4.5}$$

leading to a left arm execution speed of 1.73 times the right arm's speed.

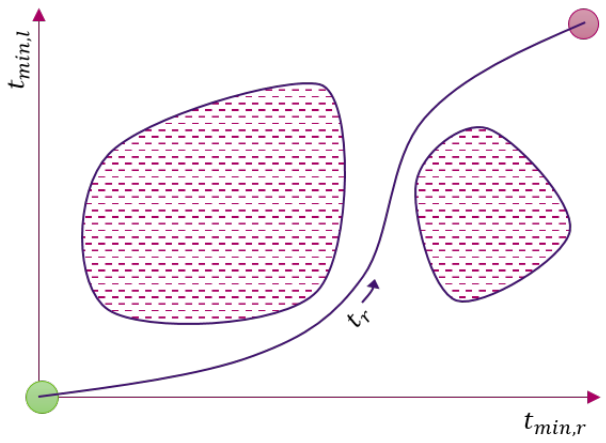

Figure 4.18: Change of relative execution speeds along a smooth coordination curve.

However, to obtain a fully defined system motion, it is not sufficient to know the relative execution speeds $\frac{dt'_L}{dt'_R}$ at any given point in the plan. Rather, to execute a coordinated motion, we are interested in the execution speed of each single arm with respect to time, i.e., the values for $\frac{dt'_L}{dt}$ and $\frac{dt'_R}{dt}$ respectively. To obtain these values, let us consider again a coordination curve such as the one shown in Figure 4.17. Since both t'_L and T'_R are free parameters, the two values cannot be described in terms of a function of each other. Rather, the coordination law can be described as a parametric curve with the parametric equations

$$\begin{aligned} t'_L &= \sigma_L(t) \text{ and} \\ t'_R &= \sigma_R(t), \end{aligned} \tag{4.6}$$

with a common parameter t which corresponds to real-world execution time. This parameter runs along the coordination curve as shown in Figure 4.18. The functions σ that define this parameterization correspond to the coordination laws as introduced in (4.3). When a valid parameterization defined by (4.6) is known, the composite coordination function given in (4.3) can be evaluated for any valid t and therefore, a coordinated motion is fully defined.

Obtaining the coordination law Since the coordination laws σ are part of the composite trajectory function, the resulting joint velocities, accelerations and jerks depend on σ. To obtain a valid overall trajectory, the composite derivatives may not exceed the physical limits of the robot actuators. This leads to constraints on the functions σ which need to be considered when searching for valid coordination laws. In principle, the full dynamics of the manipulator system would have to be considered to define these constraints for each robot state. However, manufacturers of industrial robot systems usually supply actuation bounds for the joints that are known to not exceed the physical limits of the arm. These values can be used

in obtaining constraints for finding a valid parameterization of the coordination law. The joint velocity is given by the first derivative of the trajectory function with respect to time, i.e.

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}[\tau \circ \sigma]' = \tau'(\sigma)\sigma' \leq \dot{q}_{\max}. \tag{4.7}$$

Analogously, for the maximum acceleration constraint, we have

$$\frac{\mathrm{d}^2q}{\mathrm{d}t^2} = \cdots \leq \ddot{q}_{\max}. \tag{4.8}$$

Mathematically, the problem of finding a valid parameterization of the coordination curve is similar to the trajectory planning problem. Here as there, an arbitrarily parameterized vector-valued function needs to be parameterized such that the given maximum values for the derivatives of the composite function aren't exceeded at any point.

As we have seen in the previous paragraphs, the *offline arm coordination problem* corresponds to the problem of finding a curve in the free subspace of coordination space which connects the given start and goal states in an optimally parameterizable manner. Since the shape of the curve determines the way in which it can be parameterized, the overall optimal solution can only be found if the complete problem is treated at once, such that the constraints given in equations 4.7 and 4.8 can guide the search for an optimal shape of the coordination curve. However, searching a solution to the coupled problem is forbiddingly expensive from a computational point of view. However, since the problem is mathematically similar to the path planning problem, it can be decomposed in a similar way. Applying this concept, we subdivide the search for a valid coordination law into the following three steps:

1. **Coordination law search**: Coordination space is searched for a sequence of via points, that connect start and goal states in a collision free manner, when interpolated linearly.

2. **Coordination law interpolation**: The points in this sequence are interpolated with a function exhibiting a higher degree of continuity at the interpolation points.

3. **Coordination law parameterization**: The interpolation defined in the second step is parameterized in such a way, that the derivative constraints from equations 4.7 and 4.8 are satisfied.

The following paragraph will give an overview of how each step can be solved.

Coordination law search In the first step, a sequence of states must be found through which start and goal states can be connected in a collision-free way. As discussed above, shorter paths in configuration space enable a better parameterization in the subsequent step.

This corresponds to the path planning problem in so far as shorter paths found in configuration space also enable quicker robot motion. The difference in this step lies mainly in the different dimensionality of the spaces – while seven or even fourteen dimensions have to be considered in the path planning problem, the coordination space search is constricted to two dimensions in the case of a dual-arm robot manipulator. Another difference is the geometric complexity of the space. While the complexity of configuration space strongly depends on the model complexity of the objects surrounding the robot, the complexity of the obstacles in coordination space does not. Because of the lower dimensionality and potentially lower geometrical complexity, search methods other than those discussed in Section 4.5.3 could be applicable to the coordination space search. However, the influence of this special feature of coordination space diminishes with a growing number of kinematics, as the dimensionality of coordination space is equal to the number of independent kinematic chains in the system. To not limit the generality of the implemented approach to two robots, we also employ RRT-Connect and the described path smoothing to find a coordination curve in $\mathcal{S}$. When the number of robot arms in the system is fixed at two, the computational speed of the overall coordination algorithm could be considerably increased by employing a specialized 2D search algorithm instead.

Coordination law interpolation Once a sequence of states is found that connects start and goal states, a continuous curve has to be defined by interpolating the via points found in the previous stage. As discussed in the above section on optimality, shorter curves in coordination space tend to yield shorter motion durations. This suggests that a piecewise linear interpolation between the via points leads to an optimal coordination curve. However, as noted above, this is only true under the assumption that the robots can accelerate and decelerate infinitely. When a real system as the one used here is considered, this assumption is invalid as illustrated by the dynamic constraints given in the equations 4.7 and 4.8. These penalize the high velocities and accelerations caused by the discontinuities of the linear interpolation's first and higher derivatives. Instead, smoother curves with continuous derivatives of lower amplitude yield more optimally parameterizable coordination laws. Therefore, disregarding the dynamic constraints at this stage does lead to a loss of optimality. However, the influence of the dynamic constraints on the duration of the coordinated motion is negligible in comparison to the path length objective. The reason for this is that in most motions, the robot arms spend most of their time in joint saturation, i.e. acceleration is zero. On the contrary, acceleration phases are short in comparison. Therefore, we neglect the influence of dynamic constraints at this stage and interpolate the found via points from the previous stage with piecewise linear interpolation.

Coordination law parameterization Now that the interpolation of the coordination curve is known, a valid parameterization needs to be found as the last step of coordination

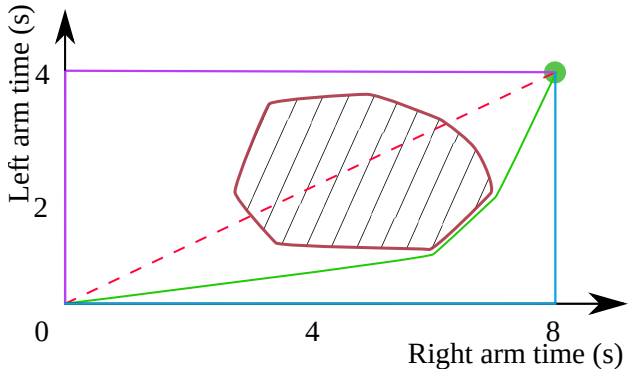

Figure 4.19: Piecewise linear coordination curve and its parameterization.

law generation. To do this, we use our a priori knowledge of the trajectories encoded in the axes of coordination space. Consider again the piecewise linear coordination curve shown in Figure 4.19. It consists of three segments with different slopes which govern the relative execution speeds of the robot's arms as discussed above. In finding an initial parameterization for this curve, we use the fact that at any given point along the coordination curve, one of the robots should be executing its trajectory as quickly as possible. Neglecting for the moment the effects introduced by the dynamic constraints, this means that either

$$\frac{\mathrm{d}t'_L}{\mathrm{d}t} = 1 \quad \text{or} \quad \frac{\mathrm{d}t'_R}{\mathrm{d}t} = 1 \quad \text{for all } t. \tag{4.9}$$

This property lets us find a parameterization that is locally valid for any linear piece of the coordination law. Following this logic, the parameterization given in (4.6) evaluates to

$$t'_L = \begin{cases} \frac{1}{6}t & 0 \le t \le 6 \\ t - 5 & 6 \le t \le 7 \\ 2(t-6) & 7 \le t \le 9 \end{cases} \tag{4.10}$$

for the left arm and

$$t'_R = \begin{cases} t & 0 \le t \le 6 \\ t & 6 \le t \le 7 \\ \frac{1}{2}(t+7) & 7 \le t \le 9 \end{cases} \tag{4.11}$$

for the right arm.

This yields the coordination laws shown in Figure 4.20. When following this coordination plan, the robot arms would have to accelerate infinitely at the joint points, i.e., where $t = 6$ and $t = 7$. In other words, the term $\frac{\mathrm{d}\sigma}{\mathrm{d}t} = \infty$. This obviously violates the dynamic constraint inequalities given in (4.8). Therefore, the found motion law is locally invalid around these points. For

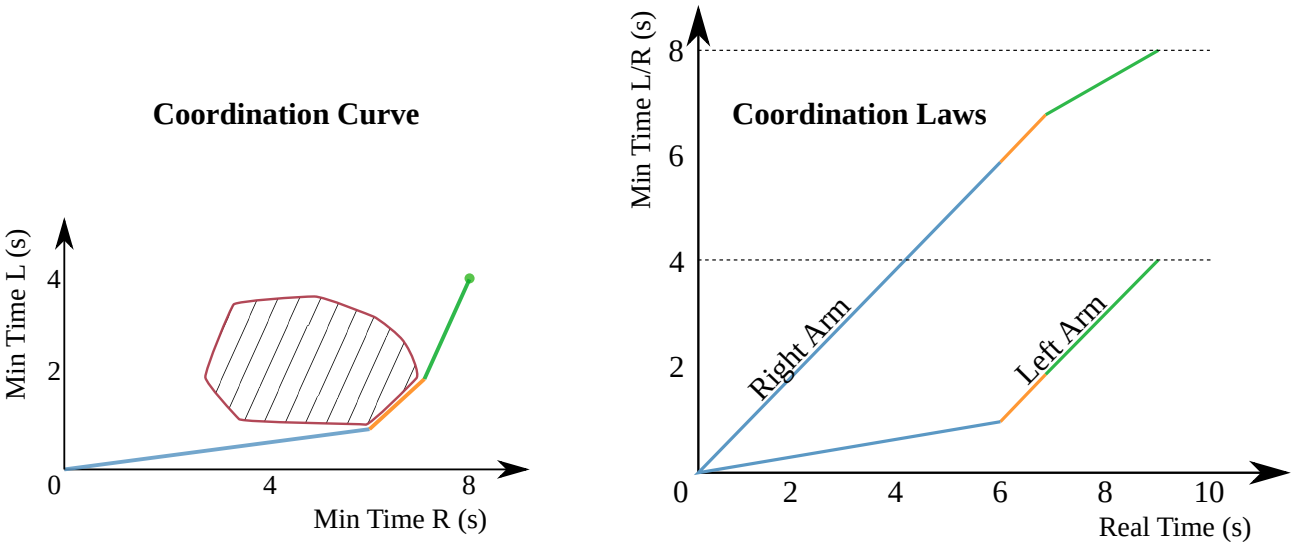

Figure 4.20: Decomposition of the coordination curve into two parametric coordination laws.

the application in the lab demonstrator discussed here, this is not a major problem, because the robot controller will reduce accelerations to feasible values when executing the trajectory anyway.

However, when directly controlling the robot without the native robot controller as a proxy, there are techniques to remedy this problem. As discussed above, the reparameterization problem occurring in the context of coordination law search is mathematically equivalent to the reparameterization problem in decoupled trajectory planning. Therefore, the same algorithms are applicable. Consequently, to make the invalid coordination law dynamically valid, one could apply the same technique we use in the trajectory planning problem described in Section 4.5.3. Starting with the locally invalid trajectory found in step two, this approach yields a dynamically valid coordination law leading to an overall valid trajectory satisfying the dynamic constraints.

Summary The previous sections in this chapter looked at the offline coordination problem, i.e., given a set of statically collision-free independent motions for the left and the right arms, compute reparameterizations of these motions that can transfer the robot system from a given start state (the left boundary values of the planned trajectories) to a given goal state (the right boundary values of the planned trajectories). We first looked at how to model this problem mathematically. We then went on to present how this problem can be decoupled into three subproblems that can be solved efficiently. Finally, we presented solutions to each of these problems and showed how these can be assembled into a solution of the original problem in case a solution exists to all the subproblems. If the decoupling approach constrains the solution space in such a way, that no decoupled solution can be found, additional measures

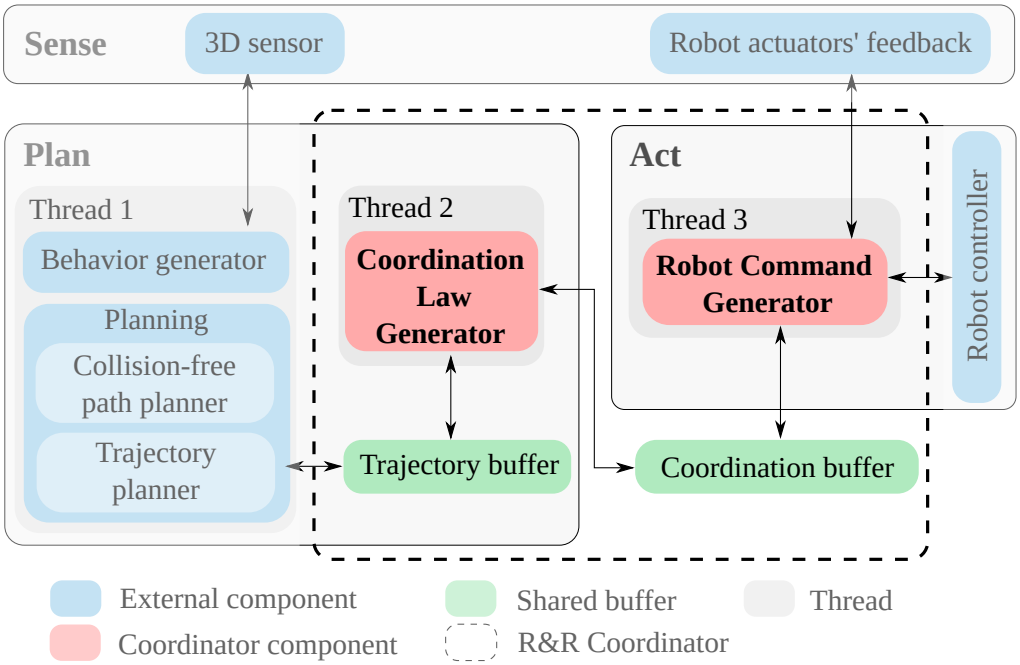

Figure 4.21: Core coordination part of the R&R system architecture.

have to be considered. These will be discussed in a later part of this work (see Section 4.9). Before we turn to this problem, however, the next section will look at how to integrate the core coordination approach described here into a live system that computes and drives coordinated robot motion at system runtime.

4.6.3 Online Coordination

The previous section discussed the problem of offline coordination, in which a known sequence of independent robot trajectories is coordinated to yield a collision-free motion of the overall system. The techniques described there can be applied to offline planning, which is applicable in situations when the complete logic of a robot motion tasks is known beforehand and no reactive capabilities are needed in the system. However, as recognized early on in the scientific community (Li et al. 1995) and as discussed in section Section 4.2, this is rarely the case for an industrial dual-arm robot system. Instead, to allow for the execution of truly general tasks, the robot must be able to plan and execute motions as is necessary given the current context and state of the environment. For this reason, it must be a dynamic system controlled by online-capable algorithms. The section will present how the described offline coordination approach can be integrated into a live planning environment to enable reactive and responsive arm coordination at system runtime. More specifically, it will discuss the interplay of the two components of the proposed system that are highlighted in Figure 4.6: the coordination

law generator and the robot command generator. Together, we name these components the Responsive and Reactive Coordinator (R&R coordinator).

The R&R coordinator implements the parts of the plan and act stages outlined with a dashed line in Figure 4.21. The two main components are the coordination law generator (CLG) and the robot command generator (RCG). The CLG implements the coordination part of the plan stage, i.e., its main responsibility lies in planning a coordinated collision-free motion using the techniques discussed above. The RCG's main responsibility is on the acting side of the system, i.e., it uses the plan for the generation of discrete motion data for the robot to execute and monitors the motion execution state. To enable responsive and reactive online planning, these two components frequently have to exchange data and work together tightly. The coupling between the components is facilitated by the buffers shown in green in Figure 4.21. These will be described in more detail in Section 4.7.

However, before we turn to the detailed description of each of the components, we define some concepts of time which unite the components and are used throughout the descriptions. These definitions of time help to understand the following parts and should be used for further reference:

- t_c – committed time: the part of the coordination plan before this time is committed for execution and not allowed to be replanned by the CLG.
- t_r – current robot time: denotes the point of the plan that is currently being executed by the real robot.
- t_{mc} – maximum readable coordinated time: used to prevent the RCG from reading and executing parts of the trajectories that are currently being (re)planned.
- t_m^L and t_m^R – maximum trajectory times: denote the end of the calculated trajectories.

4.6.4 Coordination Law Generator

The coordination technique discussed in the previous Section 4.6.2 operates on a given set of trajectories and finds a collision-free transition between the trajectories' start and end states. It operates under the assumption that the desired motions are fully known at planning time. For a reactive online planning system, this assumption does not hold. On the contrary, it is expected that a motion command for one of the arms might be triggered by the behavior generation system at some time t_1, at which point no motion has been assigned to the other arm. At some point during the first arm's motion, a motion command might be sent to the other arm at time $t_2 > t_1$. In this situation, it is undesirable for the second arm's motion to

be delayed until after the originally computed plan for the first arm's motion has been fully executed. If this were the case, execution of the second motion command would be potentially delayed by several seconds. More importantly, this would always lead to the case in which one arm waits motionlessly for the other arm's motion to finish. This is obviously not an acceptable use case for a dual-arm robot system.

To remedy this problem, the described approach for offline coordination has to be used as the core algorithm of a replanning loop which continuously replans the system trajectory as new motion commands are received during system runtime. The CLG's tasks is to control the continuous replanning. To do that, it has to determine (i) when to replan and (ii) which start and goal states are to be considered inside the core coordination algorithm. To understand how exactly this is done, consider the logic of the CLG as presented in Algorithm 4.

Algorithm 4: Coordination Law Generator

Initialization: TB, CB

while *no exit command received* **do**
  τ_L, $\tau_R \leftarrow$ TB_ReadTrajectories
  if $(\tau_L \neq \varnothing)$ *or* $(\tau_R \neq \varnothing)$ **then**
    $t_c \leftarrow$ CB_GetCommittedTime
    $t_{plan} \leftarrow$ SetPlanningTime
    $t_{sc} \leftarrow t_c + t_{plan}$
    $t_{ts}^L, t_{ts}^R \leftarrow \sigma(t_{sc})$
    $s_{start} \leftarrow [t_{ts}^L, t_{ts}^R]$
    CB_SetMC(t_{sc})
    t_{mt}^L, t_{mt}^R, $\leftarrow$ GetEndTime(τ_L, τ_R)
    $s_{goal} \leftarrow [t_{mt}^L, t_{mt}^R]$
    if s_{goal} *is in collision* **then**
      $s_{goal} \leftarrow$ BacktrackGoalState
    end
    if s_{goal} *is in collision* **then**
      continue
    else
      $\sigma \leftarrow$ FixedPathCoord(s_{start}, s_{goal})
    end
    if $(\sigma \neq \varnothing)$ **then**
      $\tau_c \leftarrow$ Compose((τ_L, τ_R), σ)
      CB_AddCoordinatedTrajectory(τ_c)
      $t_{mc} \leftarrow$ CB_GetMC()
      $t_{mc} \leftarrow t_{mc}$+GetDuration(σ)
      CB_SetMC(t_{mc})
    end
  end
end

Coordination law replanning starts when new trajectories are made available, see lines 2, 3. The CLG reads the committed time from the CB and estimates how much time will be required for coordination planning, see lines 4, 5. Our implementation uses a constant time value which is an upper threshold to computation time determined from experience. Then, start and goal states for the core fixed-path coordination solver are computed.

It is impossible to recompute any parts of the plan before t_c, as the robot is already executing the corresponding trajectory segment. It is also impossible to let the new plan start right at t_c, because the robot motion could pass by this point before an updated plan is ready. Therefore, we set the start state time t_{sc} for replanning as the sum of committed time t_c and planning time t_{plan}, see lines 6–8 in Algorithm 4. To inform the RCG of the invalidated section of the previously computed plan, the maximum readable coordinated time is set to the start time of the new plan with function CB_SetMC, see line 9 in Algorithm 4. To yield good plans, the segments considered for replanning should be as long as possible in order to incorporate all available information. Therefore, the maximum available trajectory times t_m^L and t_m^R are used as goal state for the local coordination algorithm, see line 11.

Algorithm 5: BacktrackGoalState

Initialization: CB, RC, t_l, r

```
1 while no exit command received do
2 |   τ_c ← ReadCoordTrajectoryFromCB
3 |   if τ_c ≠ ∅ then
4 |   |   t_c ← CB_GetCommittedTime
5 |   |   t_r ← RC_GetCurrentRobotTime
6 |   end
7 end
```

As arbitrary motion commands can be sent to the CLG, the goal state might cause a collision and therefore be invalid. In these cases, it cannot be used as a goal for the coordination algorithm. Therefore, an intermediate goal state is determined by the routine BacktrackGoalState, which follows the logic presented in Algorithm 5. If the goal is valid, the local coordination space is searched for a solution by a call to the function FixedPathCoord, see line 15. It searches for a valid coordination law between the current start and goal states in coordination space according to the method described in Section 4.6.2. If a coordination law σ is found, a coordinated trajectory is composed using Equation (4.2) within function Compose, see line 18. Then, the coordinated trajectory is added to the CB, see line 19. Finally, a new maximum readable coordinated time is calculated and saved to the CB as well, see lines 20–22. This allows the RCG to read and execute the new plan.

4.6.5 Robot Command Generator

Algorithm 6: Robot Command Generator

Initialization: CB, RC, t_l, r

```
1  while no exit command received do
2      τ_c ← ReadCoordTrajectoryFromCB
3      if τ_c ≠ ∅ then
4          t_c ← CB_GetCommittedTime
5          t_r ← RC_GetCurrentRobotTime
6          if t_c − t_r ≤ t_l then
7              t_c ← t_c + r
8              CB_SetCommittedTime(t_c)
9              RC ← CB_ReadConfAtTime(t_c)
10             InformMotionsDone
11         end
12     end
13 end
```

The RCG acts as a proxy between planning module and real-time execution. The logic of the RCG is presented in Algorithm 6. The command generator is initialized with coordination buffer CB, robot controller RC and constant values for the lookahead time t_l and discretization resolution r. The value t_l denotes the length of the configuration buffer that is stored inside the RC, while r denotes the temporal resolution for the discretization of the computed trajectories.

Once a new coordinated trajectory becomes available in the coordination buffer (lines 2, 3 in Algorithm 6), the RCG retrieves the committed time from the CB and subtracts from it the current robot time t_r supplied by the RC, see lines 4–6. If the difference is smaller than the lookahead time, the native buffer in the RC is running empty and a new section of the coordinated plan must be read to refill it. The committed time t_c is incremented by the discretization resolution r and the coordination buffer is updated with the new value for the committed time. Joint values for the new committed time are read from the coordinated trajectory and pushed to the robot controller for execution, see line 9. In our implementation, the command generator is parameterized such that new joint values are pushed to the robot controller at a frequency of 20 Hz as long as coordinated plans are available for execution, i.e., r equals 0.05 s. If a motion request generated by the behavior generator (see Fig. 4.6) has been completely committed for execution in the current cycle, the RCG informs the behavior generator about the successful execution of the respective motion by allowing the function WaitMotionDone to return, see line 10.

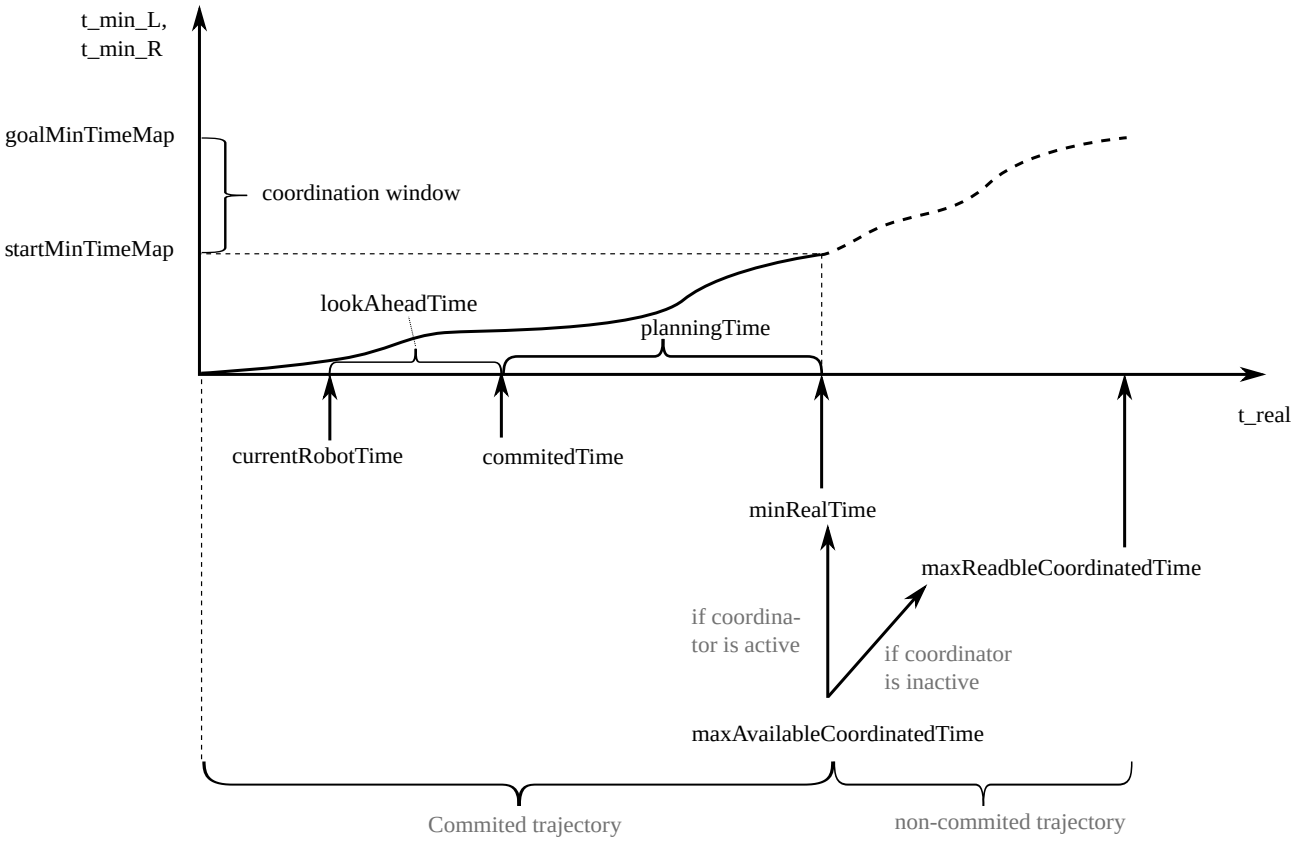

Figure 4.22: Coordination buffer with coordination law and variables used for temporal synchronization.

4.7 Inter-Thread Communication

To enable the reactive online capability of the proposed planning system, all the described components have to operate together on a shared state, which for example contains the current coordination plan of the system. Since different modules run asynchronously in different threads, operations on shared state data have to be synchronized in order to ensure data consistency across the components. This concept is represented by the two data buffers described below.

Trajectory buffer The trajectory buffer is the link between the one-shot trajectory planning carried out in thread 1 and the trajectory replanning carried out in thread 2 (refer to Figure 4.21 for the numbering of the different threads). After a motion command has been triggered via the motion interface, the trajectory planning pipeline described in Section 4.5.3 computes an uncoordinated trajectory and saves it to the trajectory buffer. When the buffer receives newly planned trajectory data, it wakes up the Coordination Law Generator thread, where a new replanning cycle is triggered. In this case, thread synchronization is trivial, because

the interaction of thread 1 and 2 can be described as a straightforward producer-consumer logic. Therefore, it is sufficient to ensure synchronized read and write access to the trajectory data, as no further state is maintained in this buffer.

Coordination buffer Interaction between threads number 2 and 3 on the other hand is more complex, because this is where the interplay between planning and acting stages unfolds. Figure 4.22 contains a graphical representation of the data stored in the coordination buffer and can serve to illustrate the implemented synchronization functionality. The horizontal axis shows the real time as it elapses in the real-world application. The vertical axis shows the elapsed minimum times $t_{min,L}$ and $t_{min,R}$ respectively. Once a valid motion law has been computed, it is saved to this buffer. As the robot starts executing this coordinated motion, the pointer for *currentRobotTime* begins to move forward along coordination law. With it, the values for the elapsed minimum times also start to increase according to the coordination laws. To make sure that the robot can always continue its coordinated motion, the coordination buffer must never be empty as long as the robot has not yet reached its final target state. At the same time, the coordination plan held in the buffer needs to change whenever a new motion command is triggered in the high-level motion interface. These two conflicting requirements are balanced by the coordination buffer. To ensure that enough joint values are always available for reading, a part of the coordination plan is committed to the robot, meaning that it cannot be changed by replanning the coordination laws. The scope of this lookahead is determined by the variable *lookAheadTime*, which can be tuned to fit the system's execution speed. When the end of the preplanned coordinated trajectory falls into this zone, the robot decelerates and comes to a stop smoothly.

The situation described in the previous paragraph is valid during phases of normal operation, i.e. when no replanning of the coordination law is under way. When a new motion command is received and replanning kicks in, a starting point for the replanning has to be determined as the very first step. Replanning the coordination law takes a noticeable amount of computation time, which must be added to the committed section of the trajectory to avoid an inconsistent buffer state. This guarantees that replanning can conclude before motion execution reaches the beginning of the replanned section. The parameter responsible for setting the maximum planning duration is *planningTime* (see Figure 4.22). Adding the values for *lookAheadTime* and *planningTime* to the *currentRobotTime* will yield the time until which the motion plan is committed to the robot during replanning phases. It is also the start point for replanning the coordination laws (These also correspond to the branching points in Figure 4.23).

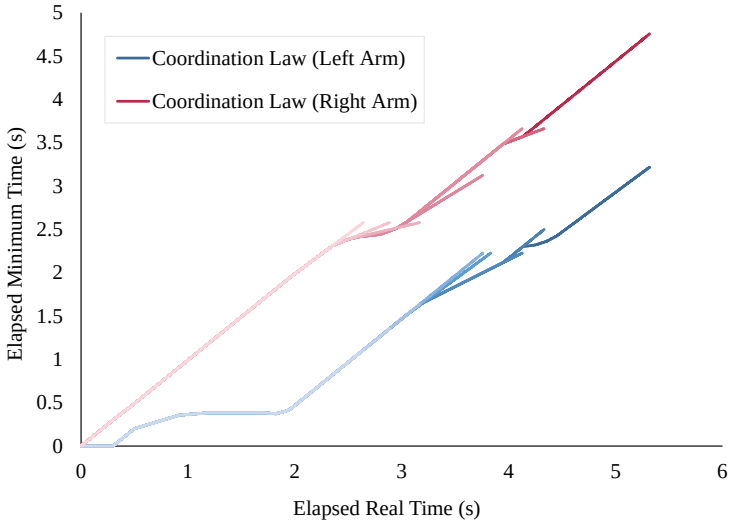

Figure 4.23: Planned and replanned coordination laws for left and right robot arms. Lighter colors denote an earlier planning time. As new motion commands arrive and are processed, some parts of the coordination laws become invalid and the new coordination law branches off.

4.8 Replanning

The proposed coordination algorithm can be implemented such that typical planning times for a given motion command do not exceed the order of several hundred milliseconds (Beuke et al. 2018). This makes it possible to replan the coordination law $c(t)$ as described in the previous sections whenever a new motion command is received. As a result, the coordination law can be continuously adapted to yield a good solution for the current set of motions to be executed. This leads to the situation shown in Figure 4.23. It shows an excerpt of the planned and replanned coordination laws for the left and right robot arms from a test run of the real robot system. The motion covers a stretch of a little over five seconds' worth of real time. In this time, the right arm executes roughly 5 s of its preplanned minimum time trajectory and the left arm executes about 3.5 s of its respective trajectory. The time at which the respective parts of the coordination law were planned are color encoded, with darker shades of blue signaling a later planning time. As can be seen, parts of the originally planned coordination laws are regularly discarded to take into account new motion commands that arrived during the execution of previous commands. When this happens, the new coordination laws branch off the originally computed coordination laws. This gives the proposed motion planning software the ability to compute and recompute coordinated motion plans on the fly. Although it does not occur in the example motions shown here, it may sometimes be necessary to allow the robot arms to backtrack along parts of the trajectory already executed to avoid deadlocks (see Section 4.9).

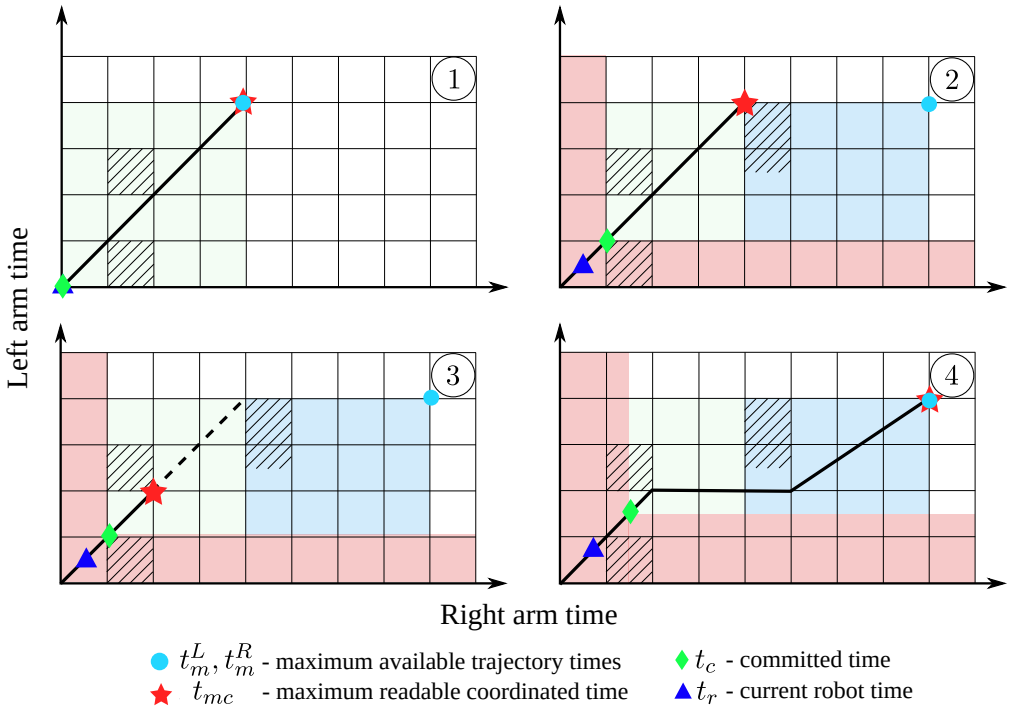

Figure 4.24: Workflow of the R&R coordinator replanning. Coordination space is discretized for visual simplicity, in real-life it is continuous with non-convex obstacles. The times t_c, t_r and t_{mc} are parameters for the curve, while $\left(t_m^R, t_m^L\right)$ is a state in coordination space.

4.8.1 R&R coordinator thread-synchronized workflow example

An example of the resulting R&R coordinator workflow is shown in Figure 4.24. It consists of four stages:

Stage 1 The CLG has computed a coordination law for the left and right arm trajectories, see coordination space shown with green area. The RCG has not yet started to feed the robot controller, thus t_c and t_r are zero. It can now start to read and execute the whole trajectory, as the time t_{mc} is set to the end of the coordination plan, see overlay of the red star and cyan dot.

Stage 2 A new motion command for the right arm is received by the CLG as depicted by the cyan dot. The coordination space is extended with the new trajectory, see blue area. The RCG is feeding motion commands to the robot and the current robot time t_r is moving along the planned coordination law, see blue triangle. The RCG reserves part of the solution for robot execution by setting the new t_c according to the current robot execution state. This way, the CLG cannot access this part of the coordination space, see red area. Thus, the part that is about to be executed is protected from being replanned.

Stage 3 The CLG starts its replanning loop as a new trajectory is added. It estimates the planning time required to reach the new goal and adds it to the committed time t_c. This way, a new maximum readable coordinated time is obtained, see the new position of the red star. By doing this, the coordinator commits to finishing the planning before the RCG would require a new plan, i.e., before t_r would reach the time t_{mc}. The old plan is discarded, see dotted line.

Stage 4 The CLG computes a new collision-free coordination law inside the green and blue areas and sets t_{mc} back to the end of the new plan, see overlay of the cyan dot and red star. This allows the RCG to read the new plan for further execution and concludes the coordination cycle.

This process is repeated when a new command is received from the behavior generator. Using the described algorithm, it is possible to plan and execute simultaneously.

4.9 Resolving Deadlocks

As discussed previously, the approach proposed in the previous sections of this chapter sacrifices completeness for computational efficiency. Since the path planner for the single arms is oblivious of the other arm's motion and the coordination algorithm is constrained to the preplanned paths, a situation can arise in which the robots paths intersect and no solution can be extracted from coordination space. To illustrate this case, picture the situation where the robot arms are crossed and the arms are commanded to exchange their end effector positions as shown in Figure 4.25. With a very high probability, this situation would lead to such a deadlock – no matter how fast or slow the robots move on the precomputed paths, they would always collide at some point. However, since responsibility of arm coordination has been taken on by the motion planning system described here, a failure is only acceptable if the goal poses lie outside of the robot's reach (see Section 4.2). Therefore, a backup strategy has to be implemented for cases in which the decoupled planning strategy leads the robot into a deadlock situation.

In some cases, a coordinated motion can still be computed by stretching the lower and upper bounds of the local coordination space and thus essentially allowing the robot arms to back up on the paths they came by. However, for several reasons, this method is not guaranteed to succeed. Firstly, there might not be a long enough previous motion to move out of the deadlock zone. Secondly and more importantly, backing up along the trajectory might not be possible for mechanical reasons. This could be the case, for example, when a part has been fed to a processing station and a safety door is now blocking the previously unobstructed access to the

Figure 4.25: Dual-arm robot YuMi in crossed-arm configuration with motion commands causing deadlock indicated by blue arrows.

machine. In this and similar cases, the robot cannot simply move back along their trajectories until a way around the deadlock is found.

Since the existence of a deadlock arises from the problem decoupling and the consequential fixed-path coordination approach, the only reliable way is to remove this constraint and allow at least one of the robots to digress from the preplanned path. Given that enough physical maneuvering space exists for the robot arms, such an approach will find a way out of the deadlock. However, there was good reason for decoupling the problem in the first place, namely the computational complexity of the motion planning problem for high-DOF single arm manipulators. Luckily, the nature of the path planning problem arising in the context of deadlock resolution is different from the global path planning problem discussed in previous sections of this work. While algorithms solving the former need to find paths connecting any two given robot configurations in a workspace, the path replanning problem is more local in nature, because start and goal configurations are not arbitrarily far away from each other.

Consider again the situation shown in Figure 4.25. Here, the robots arms start in a crossover position and are commanded to move to the goal states indicated by the goal pose coordinate systems. The direct preplanned paths to these goal states indicated by the blue arrows cause a deadlock. However, the problem is not geometrically confined to a narrow space, and a slight modification in robot paths would resolve the deadlock. Because of the local scope of this planning problem, local path planning methods offer themselves as a promising and less computationally heavy approach in this situation. In selecting a suitable approach to deadlock resolution, computation time is a key factor, as this algorithm would always run in

addition to any previous global path planning algorithm used to determine the original path. One such computationally efficient method for dual-arm redundant industrial manipulators has been proposed by Petric and Zlajpah. It works through a Cartesian space-based controller and calculates a series of incremental joint motions using task-space distances and inverse kinematics (Petrič et al. 2011). In the context of this thesis, this and similar approaches have been evaluated with respect to its applicability in the deadlock resolution problem in a separate work (Koo 2017).

The basic idea of using this approach is as follows: A deadlock can be characterized by the absence of a solution in coordination space. Whenever such a deadlock is detected, the colliding subsection of the path can be identified and a set of locally alternative via points can be determined, taking into account all degrees of freedom at the same time. This new set of via points can be interpolated and transformed into a trajectory piece using the methods already described earlier in this thesis. It then replaces the piece of trajectory between the deadlock-causing states and therefore locally modifies the previously blocked subspace in coordination space. After this modification, the decoupled coordination approach can continue to work as described until the next deadlock occurs and the process is repeated. Since this path planning method does not explore the search space but exploits a predefined motion strategy based on the workspace situation, it is potentially faster. As a detailed evaluation shows, it also succeeds with a high probability for typical local path replanning problems (Koo 2017).

5 Evaluation

The performance of the proposed approach was evaluated on the industrial use-case shown in Fig. 4.1. The core coordination algorithm is designed to satisfy two objectives: (i) responsiveness, i.e., computation time must be less than execution time to allow for continuous movements and (ii) reactiveness, i.e., new motion commands must be coordinated and executed as they become available. In the following, we present two sets of experiments to show how the proposed approach performs with respect to each objective. All experiments were run on an Intel Core i7-6700K 4.0 GHz processor with 32 GB RAM controlling an ABB YuMi dual-arm robot. Synchronous motion commands for all joints were triggered on the robot via a custom interface implemented on top of socket messages. In the experiments, the robot has to grasp parts from the central pallet and put them into the side pallets. An excerpt of the motions generated by the coordination algorithm is shown in Fig. 5.1.

5.1 Responsiveness

In order to evaluate the responsiveness, we compare the time needed to compute a plan for a given motion command with the time needed to execute the coordinated trajectory for that command. Based on the palletizing scenario described in Fig. 4.1, we let each arm perform 100 motion commands while picking and placing parts from the central to the sink pallets. Computation times and the respective execution times for the corresponding motions are depicted in Fig. 5.2. To test a range of different path lengths, the behavior generator was set up to command either short, intermediate or long motions during each pick-and-place cycle. Therefore, execution times show only slight variations around three different levels.

For the left arm, the ratio of computational vs. execution times ranges from 2.6% to 35.3%, with an average of 7.8% and a standard deviation of 5%. For the right arm, the ratios range from 3.9% to 36.9 % with an average of 12.4% and a standard deviation of 6.3%. Computation times were always smaller than execution times, regardless of what type of motion was commanded. Therefore, the responsiveness objective is met and the robot can move without stopping while receiving new commands. Note that the shown execution times are a lower

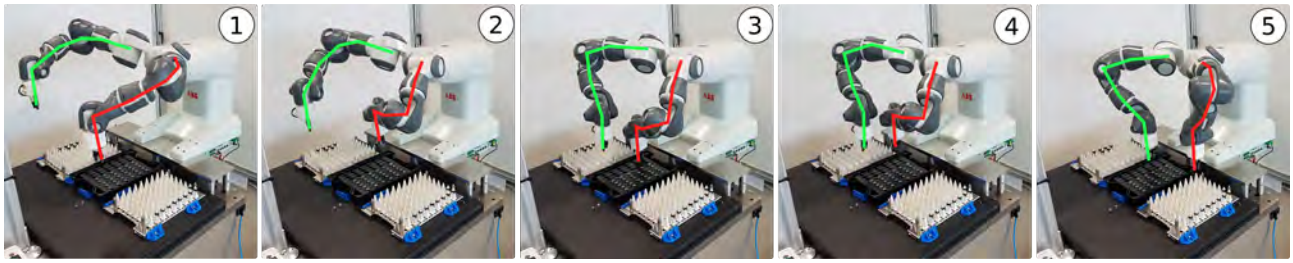

Figure 5.1: Real robot's workflow of the palletizing use case. At the beginning of the movement, both arms are in their outward rest position (1). In this run, the motion command for the left arm was triggered first by the behavior generator. Therefore, that arm starts its motion toward the source pallet first (2). After the motion command for the right arm is generated and coordinated with the already moving left arm, the right arm also starts the movement toward its current picking position (3). As soon as the left arm picks the part and clears the central collision area (4), the right arm can complete its pick motion (5).

bound to actual execution duration on the robot. This is because we use the duration of uncoordinated trajectories in this part of the evaluation to cover the case of naturally collision-free motions. Any necessary arm coordination would slow down these times and therefore increase their duration. This effect makes the actual system more responsive as shown here.

5.2 Reactiveness

To show the reactiveness of our approach, we compare the results of our implementation to an identical one which lacks the replanning feature. To achieve this, we alter line 8 of Algorithm 4 such that the start state for the local coordination is equal to the goal state of the previous

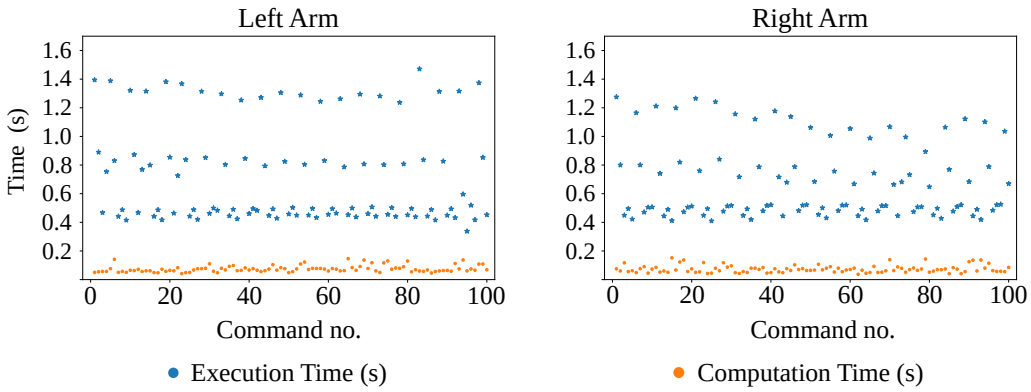

Figure 5.2: Computation versus motion execution times. Computational time is always smaller than execution time which allows for continuous robot movement by calculating the coordinated trajectory online.

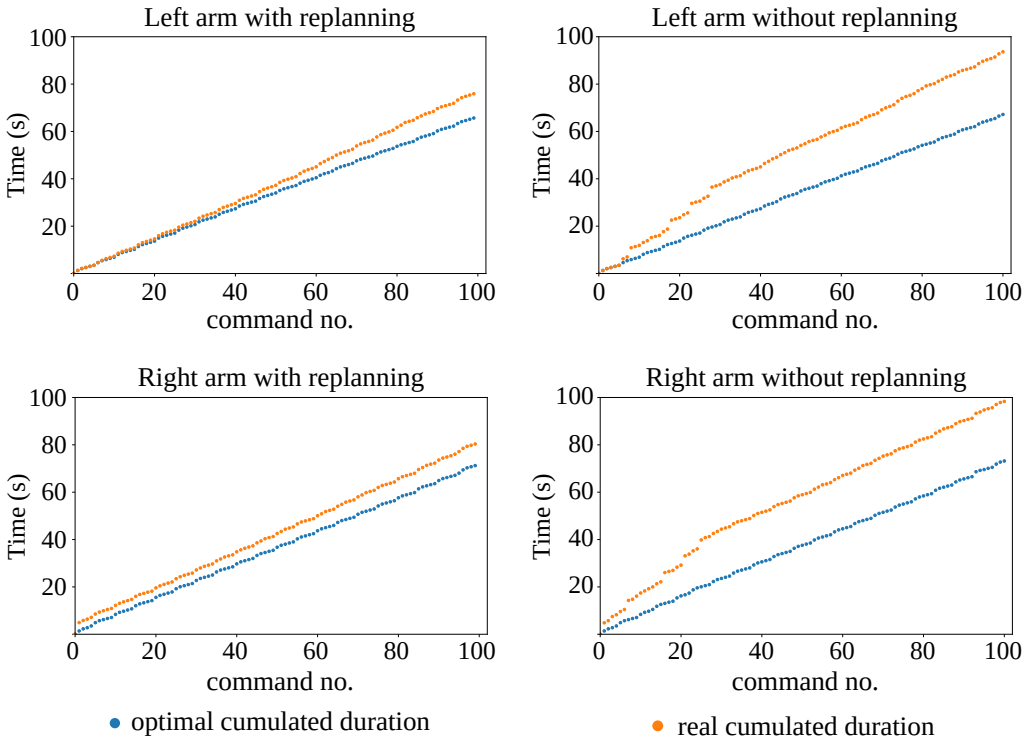

Figure 5.3: Real versus theoretically optimal execution times. The coordination algorithm with replanning stays closer to optimum after 100 commands being executed in contrast to algorithm without replanning.

coordination cycle, i.e. no replanning occurs and reactiveness is lost. The results of both replanning and non-replanning approaches are presented in Fig. 5.3. The theoretically reachable optimal time is computed by assuming the ideal situation in which both arms perform their mission and no collision occurs. In that case, the optimal time to finish all tasks equals the duration of the slowest arm's motion.

The replanned motion takes 80.8 s to execute, i.e., it is 12.7% slower than the theoretical optimum at 71.7 s. The non-replanned motion takes 98.3 s, making it 34.4% slower than the optimum at 73.1 s. The small difference in optimal times is due to the nondeterministic behavior of the RRT-Connect path planner. Since the data was gathered in two separate experiments, the paths planned by the probabilistic planner can be slightly longer or shorter even though the same goal positions were requested in the received motion commands. When only the first 27 motion commands are taken into account, the difference between execution times is even greater. The replanning approach generates trajectories that are only 21.1% longer than optimum, while the trajectories computed by the non-replanning approach are 98.9% longer than optimum. This is because a phase shift for the motion is found after command no. 27 that makes it naturally collision-free. In our setup, this happens when one arm places a part at the same instant the other arm picks one up. Any deviation from the optimum after command

no. 27 is caused by a slight geometrical asymmetry in the setup. The reactive version of the algorithm finds the same shift nearly instantaneously, see jump of the orange curve in the lower left diagram at command no. 1. These results demonstrate a clear advantage of the proposed solution over the non-replanning algorithms.

Our results show that non-reactive coordination planning might be acceptable for repetitive tasks that are static in execution time and completely known beforehand. However, in the face of any kind of uncertainty (e.g., in execution duration, task ordering, etc.), reactive planners as the one presented in this paper are needed to apply the idea of automatic arm coordination to real-world domains. To the best of our knowledge, there is no state-of-the-art coordination algorithm that is capable of reactively improving the motion coordination plan of the two arms as new motion commands become available during plan execution.

5.3 Comparison with task-space based zones

The proposed automatic arm coordination method allows the robot to be programmed at a higher level of abstraction. However, it can only be considered for practical application in real use case scenarios when this benefit doesn't come at a higher runtime cost and/or execution speed. To evaluate how the proposed approach compares to conventional arm coordination approaches, we let the robot work on the industrial palletizing task shown in Figure 5.4. In this test setup, the robot has to use both hands to move parts from the central source pallet to the outer sink pallets. Motion commands are determined by a higher-level planner and are triggered via the interface functionality described in 4.6.1. A total of two experiments were conducted: in one experiment, the robot is driven by the implemented automatic coordination. In the second experiment, the robot is driven by a manual coordination approach as described below.

5.3.1 Manual coordination approach

When manually coordinating the motions of arms working in close proximity, the geometrical layout of the particular task has to be considered. For manual coordination, we define a rectangular shape in the robot's workspace that designates a zone of mutually exclusive end effector access for the arms, see blue box in Figure 5.4. Access to these zones is provided exclusively through two access points located on the edges toward either side. As long as the end effector of one arm is inside this zone, the other arm is barred from planning and executing motion commands that would lead into the zone. To allow for comparison of the two approaches,

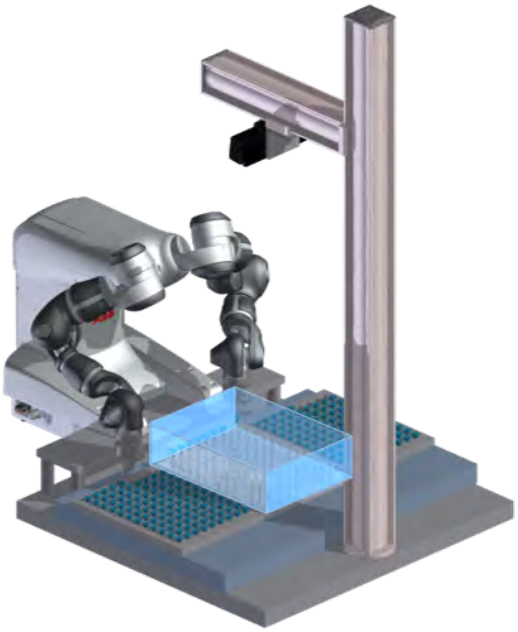

Figure 5.4: Dual-arm robot working on a palletizing task: both arms pick parts from the central pallet and place them in the respective outer pallets. The light blue box is a zone of mutually exclusive end effector access used in the manual coordination approach.

the same implementation is used in comparing the automated and manual coordination schemes. Apart from the given modifications, the same series of motion commands is executed via the implemented motion planning and control interface. The described automatic arm coordination algorithm executes in the manual coordination experiment – however, since all arm motions are already collision free by virtue of the previous manual coordination, no trajectory modification takes place at this stage.

5.3.2 Experimental results

The results of the experiments are shown in Figure 5.5. In both the manual and automatic scenario, the robot arms execute 60 s worth of non-coordinated (minimum time) trajectories in a collision-free manner. For each arms, a total of four hypothetical and actual motion execution times are shown on the y-axis in the respective diagrams. The dashed green lines represent the hypothetical situation in which the arms can execute all of their trajectories at the preplanned maximum velocity without causing a collision. In geometrically tight collaboration situations like in our test setup, this case is very unlikely to occur. For reference, the dashed red lines represent the hypothetical situation in which the preplanned velocities are executed at half the maximum possible speed by both arms. As expected, the real execution times for the automatic and manual coordination cases lie in between these two extremes. They are shown in blue for

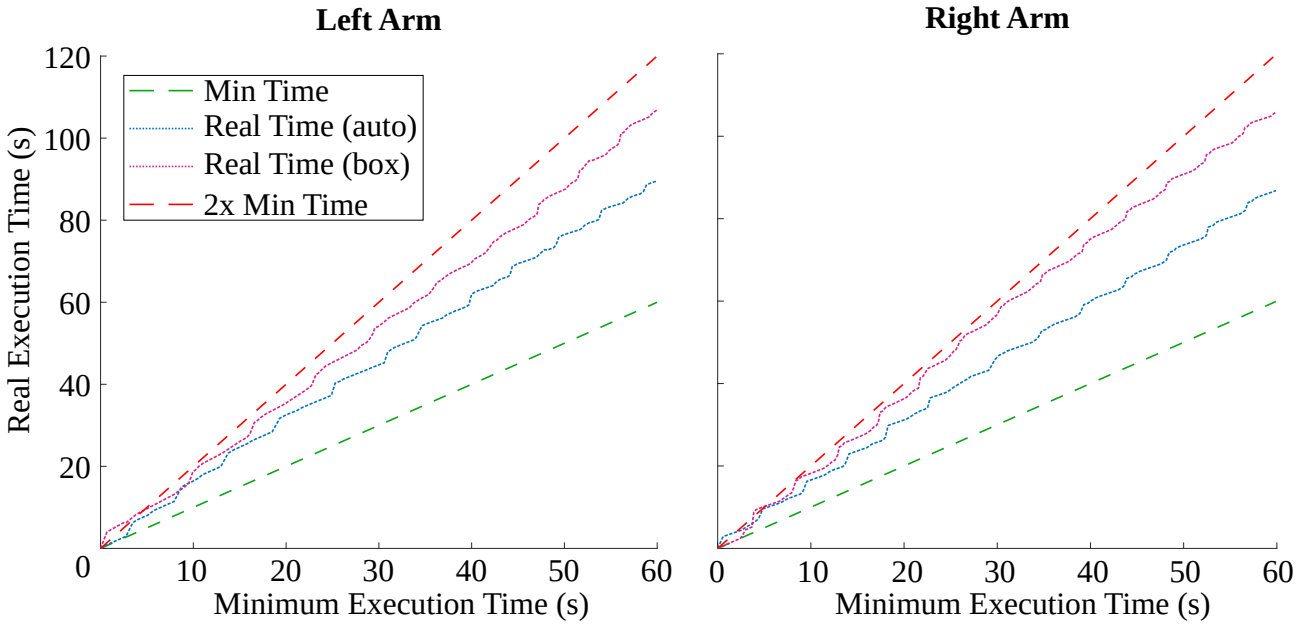

Figure 5.5: Motion execution times for manual and automatic arm coordination.

the automatic coordination case and in brown for the manual approach based on a task space exclusive zone as described above. Execution of the manually coordinated motion plan takes roughly 107 s (104.79 s for the right arm and 106.85 s for the left arm). When executed via the automatic arm coordination algorithms, execution of the same motion sequence takes roughly 90 s (86.8 s for the right arm and 89.53 s for the left arm). Therefore, in the example use case considered here, automatic arm coordination leads to roughly 16.2% shorter execution time compared to manual coordination. In the manual coordination case, both robot arms always have to wait before entering the mutually exclusive zone, because the duration of motions outside this zone is shorter than that of the motions inside the zone. Therefore, the arm inside the zone can't clear it before the arm outside return in the next cycle. In contrast, in the automatic coordination case the outside arm can already enter the zone while the other arm is still clearing it, leading to shorter wait times.

The manual coordination approach used here as a baseline for comparison is arguably naive and could certainly be improved upon. A more careful manual design could match the performance of the automatic approach or even outperform it slightly. However, this improvement comes at the cost of using a considerable amount of expert time and knowledge, and the found solution is dependent on the details of a single use case and its exact geometry. The proposed automatic solution is dependent only on the geometrical model, which can be changed and adapted much more easily than an implicit representation of workspace geometry spread out over many lines of robot motion code. Our results show that the proposed general solution

for arm coordination can make this investment unnecessary, yielding satisfying solutions while considerably decreasing the complexity of programming dual-arm industrial robots.

5.4 Deadlock avoidance

Since integration into the planning system is still ongoing, the functionality of the deadlock avoidance and local path replanning mechanism was evaluated separately. For this part of the evaluation, a total of 2000 deadlock problems were randomly generated and solved by the deadlock solver developed as part of this work in (Koo 2017) and briefly discussed in Section 4.9. Test cases were generated from a total of four different categories with each category contributing 500 randomly generated test cases. For each test case, virtual boxes were placed in the robot's workspace, representing start and goal areas for the end effector poses. Start and goal poses for the test cases were generated from a uniform random distribution over these zones. Depending on the specifics of the test case, the workspace was either free of obstacles or contains obstacles placed in a fixed location suitable for the robot and test case geometry. The different categories represent collision situations that can occur during motion execution as follows:

Fixed Obstacle A fixed obstacle is placed at a random location along the robot arm, start and goal poses are randomly sampled from two boxes on each side of the obstacle.

Narrow passage Two obstacles are statically placed to create a narrow passage for the robot end effector. Start and goal poses are drawn from two boxes on each side, forcing the arm to go through the passage.

End effector arm collision Both arms are assigned start and goal poses from random samples leading their end effectors into collision (a situation typically occurring in this test case is illustrated in Figure 4.25).

Joint arm collision Both arms are assigned start and goal poses randomly sampled from boxes that are placed such that a collision would occur along the robot arms if they move along the shortest path to their respective goals.

For an in-depth description of the test cases including illustration examples, the reader is referred to (Koo 2017).

An overview of the evaluation results for these scenarios is given in Table 5.1. Out of the 500 test cases per category, a maximum of 9 test runs exceeded the cutoff number of steps ($n = 200$) and were therefore considered failed. All other tests passed successfully within the

	Runs	Failed	Success rate	Average steps
Fixed obstacle	500	4	0.974	72.1
Narrow Passage	500	0	1	94.0
End effector collision	500	1	0.996	67.3
Joint collision	500	9	0.968	87.8

Table 5.1: Task-space based local deadlock resolution in different situations.

limit of allowed computation steps. The most commonly appearing situation of end effector collision between the two robot arms could be resolved in all but one instance, and the average number of computation steps necessary to guide the robot arms from their initial poses to the goal poses never exceeded $n = 94$.

These data suggest two main conclusions. Firstly, the evaluated approach for task-space based deadlock resolution is robust for local path replanning. Almost all randomly generated test cases could be solved. Moderate failure rates were observed for the more challenging case of joint-joint collisions. These would not be acceptable in a production environment. However, the test cases where the approach failed to find a solution corresponded to extreme arm configurations generated randomly. In a practical applications of a dual-arm industrial robot handling well-defined tasks, these situations are very unlikely to occur.

Secondly, the computational performance of the approach is acceptable for use inside a replanning loop. The average number of computational steps taken to resolve each situation varies from roughly 70 to roughly 100. Further experiments suggest that this number can be reduced by about half with an adaptive parameterization of the approach. The collision checker is called only once per iteration. Apart from this call, the Jacobian inversion is the only other computationally heavy step for this algorithm. Although computational performance couldn't be evaluated as part of the planning system yet, these considerations suggest that the deadlock resolution algorithm would increase the replanning loop's computation time only marginally.

6 Conclusion and Outlook

Conclusion This dissertation has introduced a motion planning and control system for dual-arm industrial robot manipulators. After discussing the necessary prerequisites from the fields of robotic motion planning and the state of the art in dual-arm industrial robots, it has identified arm coordination in dual-arm robots as one of the challenges hampering the easy application of these devices today. It then argues that an automation of the manual programming tasks associated with the close arm proximity has the potential to remove these application obstacles. Taking real workstations from manufacturing sites into view, this thesis has then set forth a set of requirements that an automated arm coordination and motion planner must satisfy in order to be feasibly applicable in the domain of industrial robotics. One of the most important of these requirements has turned out to be online-capability of the motion planning pipeline.

With these requirements at hand, this thesis has then identified algorithmic approaches suitable for automatic arm coordination and built on top of these approaches to obtain an online-capable coordinated motion planner architecture for industrial dual-arm robots. A real test system has been designed and implemented around the core coordination algorithms to help evaluate the ideas developed in the theoretical part of the work. Using example use cases inspired by real-life applications from industrial production, an evaluation of the developed motion planner has then been carried out on the real system. This evaluation has focused on the applicability in industrial use case scenarios. Using the test system as a demonstrator, the work has also shown how the proposed motion planning system can improve the state of the art in dual-arm robot teaching, either directly or as a concise low-level interface enabling higher-level abstract behavior planners for industrial automation.

The dual-arm motion planner proposed here offers an intuitive and minimal interface, which completely encapsulates the complexity of arm coordination. At the same time, it allows the user to manually handle synchronization between arms or one arm and other task logic via the WaitMotionDone functionality. Using the proposed interface, the human dual-arm robot programmer can concentrate on task logic alone and only specify manual coordination when so desired.

Outlook The number of different scenarios in which dual-arm robots could be applied in manufacturing automation is virtually endless. To date, only a fraction of these scenarios has already been automated with dual-arm robots. The core motion planning algorithms discussed in this thesis have focused on the subset of scenarios that require both arms to work mechanically independently in the same workspace. The planner implemented here can support only scenarios following this paradigm. However, the architecture of the system has been designed such that it could support planners for other types of scenarios. For example, one very intuitive use of a dual-arm system would be to let both arms handle a single object simultaneously. Other planners suitable for closed kinematic chains could be integrated to allow for richer functionality in the high-level motion interface. Like this, functionality for dual-arm handling and other dual-arm specific motion types could be supported.

As this work has briefly discussed before, the motion planning system proposed here assumes responsibility for planning a coordinated motion in all possible circumstances. Although the human programmer still retains full control over arm coordination if desired, the system still has to work robustly in order to be adopted in reality. Preliminary tests have been carried out, but thorough testing of a robustified implementation under real-life circumstances has yet to prove that the motion planner can reach the required robustness as is actually usable in real applications. Further real-life testing could also reveal if other assumptions made in the development of this work are justified given real-life scenarios.

Finally, it would be interesting to explore new programming paradigms for dual arm robots that could be enabled by the motion planning approach discussed in this thesis. This work has briefly explored this direction with a behavior generation mechanism based on an aggregation of interface and logic functionality in skills and tasks. However, given the limited complexity of the motion planning interface, the application of more high-level behavior planners seems possible. For example, the dual-arm motion planner could be extended to report back a suitably defined cost metric to its caller. With this metric in place, a higher-level approach could use the abstracted motion interface for self-optimizing motion plans. Along the same lines, it also seems conceivable to use this metric for learning new motion plans from scratch.

Bibliography

Afaghani et al. 2015
Afaghani, Ahmad Yasser; Aiyama, Yasumichi, 2015.
On-line collision avoidance of two command-based industrial robotic arms using advanced collision map.
In: *IEEE International Conference on Advanced Robotics (ICAR)*, pp. 15–20.
New York: IEEE.
ISBN 978-1-4673-7509-2.
DOI: 10.1109/ICAR.2015.7251490

Akella et al. 2004
Akella, Srinivas; Peng, Jufeng, 2004.
Time-scaled coordination of multiple manipulators.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3337–3344.
New York: IEEE.
ISBN 0-7803-8232-3.
DOI: 10.1109/ROBOT.2004.1308769

Amato et al. 1996
Amato, Nancy M.; Wu, Yan, 1996.
A Randomized Roadmap Method for Path and Manipulation Planning.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 113–120.
New York: IEEE.
ISBN 0-7803-2988-0.
DOI: 10.1109/ROBOT.1996.503582

Arkin 2008
Arkin, Ronald C., 2008.
Governing lethal behavior: Embedding ethics in a hybrid deliberative/reactive robot architecture part I: Motivation and philosophy.
In: *Proceedings of the 3rd ACM/IEEE International Conference on Human-Robot Interaction*, pp. 121–128.
New York: Association for Computing Machinery.
ISBN 978-1-60558-017-3.
DOI: 10.1145/1349822.1349839

Arkin et al. 1997
Arkin, Ronald C.; Balch, Tucker, 1997.
AuRA: principles and practice in review.
Journal of Experimental & Theoretical Artificial Intelligence, **9** (2-3), pp. 175–189.
DOI: 10.1080/095281397147068

Arslan et al. 2013
Arslan, Oktay; Tsiotras, Panagiotis, 2013.
Use of relaxation methods in sampling-based algorithms for optimal motion planning.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2421–2428.
New York: IEEE.
ISBN 978-1-4673-5643-5.
DOI: 10.1109/ICRA.2013.6630906

Bennewitz et al. 2001
Bennewitz, Maren; Burgard, Wolfram; Thrun, Sebastian, 2001.
Optimizing schedules for prioritized path planning of multi-robot systems.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 271–276.
New York: IEEE.
ISBN 0-7803-6576-3.
DOI: 10.1109/ROBOT.2001.932565

Beuke et al. 2018
Beuke, Felix; Alatartsev, Sergey; Jessen, Simon; Verl, Alexander, 2018.
Responsive and Reactive Dual-Arm Robot Coordination.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 316–322.
New York: IEEE.
ISBN 978-1-5386-3081-5.
DOI: 10.1109/ICRA.2018.8462868

Bhattacharjee et al. 2011
Bhattacharjee, Preetha; Rakshit, Pratyusha; Goswami, Indrani; Konar, Amit; Nagar, Atulya K., 2011.
Multi-robot path-planning using artificial bee colony optimization algorithm.
In: *3rd World Congress on Nature and Biologically Inspired Computing*, pp. 219–224.
New York: IEEE.
ISBN 978-1-4577-1124-4.
DOI: 10.1109/NaBIC.2011.6089601

Biagiotti et al. 2008
Biagiotti, Luigi; Melchiorri, Claudio, 2008.
Trajectory Planning for Automatic Machines and Robots.
Berlin, Heidelberg: Springer Verlag, 2008.
ISBN 978-3-540-85628-3.
DOI: 10.1007/978-3-540-85629-0

Bobrow et al. 1985
Bobrow, James E.; Dubowsky, Steven; Gibson, John S., 1985.
Time-Optimal Control of Robotic Manipulators Along Specified Paths.
The International Journal of Robotics Research, **4** (3), pp. 3–17.
DOI: 10.1177/027836498500400301

Boor et al. 1999
Boor, Valérie; Overmars, Mark H.; van der Stappen, A. Frank, 1999.
The Gaussian sampling strategy for probabilistic roadmap planners.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1018–1023.
New York: IEEE.
ISBN 0-7803-5180-0.
DOI: 10.1109/ROBOT.1999.772447

Brooks 1986
Brooks, Rodney A., 1986.
A Robust Layered Control System for a Mobile Robot.
IEEE Journal on Robotics and Automation, **2** (1), pp. 14–23.
DOI: 10.1109/JRA.1986.1087032

Bruce et al. 2002
Bruce, James; Veloso, Manuela, 2002.
Real-time randomized path planning for robot navigation.
In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2383–2388.
New York: IEEE.
ISBN 0-7803-7398-7.
DOI: 10.1109/IRDS.2002.1041624

Byrne et al. 2014
Byrne, Steven; Naeem, Wasif; Ferguson, Stuart, 2014.
Improved APF strategies for dual-arm local motion planning.
Transactions of the Institute of Measurement and Control, **37** (1), pp. 73–90.
DOI: 10.1177/0142331214532002

Canny 1988
Canny, John F., 1988.
The Complexity of Robot Motion Planning.
Cambridge, London: The MIT Press, 1988.
ISBN 0-262-03136-1

Choi et al. 2017
Choi, Younsung; Kim, Donghyung; Hwang, Soonwoong; Kim, Hyeonguk; Kim, Namwun; Han, Changsoo, 2017.
Dual-arm robot motion planning for collision avoidance using B-spline curve.
International Journal of Precision Engineering and Manufacturing, **18** (6), pp. 835–843.
DOI: 10.1007/s12541-017-0099-z

Choset et al. 2005
Choset, Howie; Lynch, Kevin M.; Hutchinson, Seth; Kantor, George A.; Burgard, Wolfram; Kavraki, Lydia E.; Thrun, Sebastian, 2005.
Principles of Robot Motion: Theory, Algorithms, and Implementations.
Cambridge, London: The MIT Press, 2005.
ISBN 0-262-03327-5

Correll et al. 2016
Correll, Nikolaus et al., 2016.
Analysis and Observations from the First Amazon Picking Challenge.
IEEE Transactions on Automation Science and Engineering, **15** (1), pp. 172–188.
DOI: 10.1109/TASE.2016.2600527

Coumans 2015
Coumans, Erwin, 2015.
Bullet 2.83 Physics SDK Manual.
Available from: https://github.com/bulletphysics/bullet3/blob/master/docs/Bullet_User_Manual.pdf
Visited on: 03/25/2018

Ćurković et al. 2010
Ćurković, Petar; Jerbić, Bojan, 2010.
Dual-arm robot motion planning based on cooperative coevolution.
In: *2010 Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS)*, pp. 169–178.
Berlin, Heidelberg: Springer, 2010.
ISBN 978-3-642-11628-5.
DOI: 10.1007/978-3-642-11628-5_18

Denavit et al. 1955
Denavit, Jacques; Hartenberg, Richard Scheunemann, 1955.
A kinematic notation for lower-pair mechanisms based on matrices.
Transactions of the ASME Journal of Applied Mechanics, **23**, pp. 215–221

Dornhege et al. 2012
Dornhege, Christian; Eyerich, Patrick; Keller, Thomas; Trüg, Sebastian; Brenner, Michael; Nebel, Bernhard, 2012.
Semantic Attachments for Domain-Independent Planning Systems.
In: Prassler, Erwin et al. (Ed.): *Towards Service Robots for Everyday Environments*.
Berlin, Heidelberg: Springer, 2012, pp. 99–115.
ISBN 978-3-642-25116-0.
DOI: 10.1007/978-3-642-25116-0

Duffy 1980
Duffy, Joseph, 1980.
Analysis of Mechanisms and Robot Manipulators.
New York: John Wiley & Sons, 1980.
ISBN 0-470-27002-0

Elbanhawi et al. 2014
Elbanhawi, Mohamed; Simić, Milan N., 2014.
Sampling-Based Robot Motion Planning: A Review.
IEEE Access, **2**, pp. 56–77.
DOI: 10.1109/ACCESS.2014.2302442

Fox et al. 2003
Fox, Maria; Long, Derek, 2003.
PDDL2. 1: An extension to PDDL for expressing temporal planning domains.
AAAI Journal of Artificial Intelligence Research, **20**, pp. 61–124

Gammell et al. 2015
Gammell, Jonathan D.; Srinivasa, Siddhartha S.; Barfoot, Timothy D., 2015.
Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3067–3074.
New York: IEEE.
ISBN 978-1-4799-6923-4.
DOI: 10.1109/ICRA.2015.7139620

Gat 1998

Gat, Erann, 1998.
On Three-Layer Architectures.
In: Kortenkamp, David; Bonasso, R. Peter; Murphy, Robin (Ed.): *Artificial Intelligence and Mobile Robots*, pp. 195–210.
Stanford: AAAI Press, 1998.
ISBN 0-262-61137-6

Harada et al. 2012

Harada, Kensuke; Foissotte, Torea; Tsuji, Tokuo; Nagata, Kazuyuki; Yamanobe, Natsuki; Nakamura, Akira; Kawai, Yoshihiro, 2012.
Pick and place planning for dual-arm manipulators.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2281–2286.
New York: IEEE.
ISBN 978-1-4673-1405-3.
DOI: 10.1109/ICRA.2012.6224780

Heer 2017

Heer, Carsten, 2017.
World Robot Report 2017: Industrial Robotics.
Frankfurt a. Main: IFR International Federation of Robotics, 2017.
Available from: https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf

Hoppe et al. 2017

Hoppe, Sabrina; Lou, Zhongyu; Hennes, Daniel; Toussaint, Marc, 2017.
Deep Learning for Manipulation with Visual and Haptic Feedback.
In: *IROS 2017 Workshop on the Frontiers of Contact-Rich Manipulation*.
Available from: https://ipvs.informatik.uni-stuttgart.de/mlr/papers/17-hoppe-IROSws.pdf

Hsu et al. 2004

Hsu, David; Sun, Zheng, 2004.
Adaptively combining multiple sampling strategies for probabilistic roadmap planning.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 774–779.
New York: IEEE.
ISBN 0-7803-8645-0.
DOI: 10.1109/RAMECH.2004.1438016

Kant et al. 1986
Kant, Kamal; Zucker, Steven W., 1986.
Toward Efficient Trajectory Planning: The Path-Velocity Decomposition.
The International Journal of Robotics Research, **5** (3), pp. 72–89.
DOI: 10.1177/027836498600500304

Karaman et al. 2011
Karaman, Sertac; Frazzoli, Emilio, 2011.
Sampling-based algorithms for optimal motion planning.
The International Journal of Robotics Research, **30** (7), pp. 846–894.
DOI: 10.1177/0278364911406761

Kimmel et al. 2016
Kimmel, Andrew; Bekris, Kostas E., 2016.
Scheduling Pick-and-Place Tasks for Dual-arm Manipulators using Incremental Search on Coordination Diagrams.
ICAPS workshop on planning and robotics (PlanRob), visited on: 04/17/2018.
Available from: https://www.cs.rutgers.edu/~kb572/pubs/kimmel_schedule.pdf

Kleiner et al. 2011
Kleiner, Alexander; Sun, Dali; Meyer-Delius, Daniel, 2011.
ARMO: Adaptive road map optimization for large robot teams.
In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3276–3282.
New York: IEEE.
ISBN 978-1-61284-456-5.
DOI: 10.1109/IROS.2011.6094734

Koo 2017
Koo, Chee Hung, 2017.
Untersuchungen zur situationsabhängigen und kollisionsfreien Bahnsteuerung für zweiarmige Knickarmroboter.
Dresden, Univ., Master's thesis, 2017

Korpela et al. 2014
Korpela, Christopher; Orsag, Matko; Oh, Paul, 2014.
Towards valve turning using a dual-arm aerial manipulator.
In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3411–3416.
New York: IEEE.
ISBN 978-1-4799-6934-0.
DOI: 10.1109/IROS.2014.6943037

Kuffner et al. 2000
Kuffner, James J.; LaValle, Steven M., 2000.
RRT-Connect: An efficient approach to single-query path planning.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 995–1001.
New York: IEEE.
ISBN 0-7803-5886-4.
DOI: 10.1109/ROBOT.2000.844730

KUKA Roboter GmbH 2013
KUKA Roboter GmbH, 2013.
KUKA System Software 8.3 Operating and Programming Instructions for System Integrators

Kunz et al. 2010
Kunz, Tobias; Reiser, Ulrich; Stilman, Mike; Verl, Alexander, 2010.
Real-time path planning for a robot arm in changing environments.
In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5906–5911.
New York: IEEE.
ISBN 978-1-4244-6676-4.
DOI: 10.1109/IROS.2010.5653275

Latombe 1991
Latombe, Jean-Claude, 1991.
Robot Motion Planning.
Second printing.
New York: Springer Science+Business Media, 1991.
ISBN 978-0-7923-9206-4

LaValle 2006
LaValle, Steven M., 2006.
Planning Algorithms.
Cambridge: Cambridge University Press, 2006.
ISBN 978-0-5115-4687-7.
DOI: 10.1017/CBO9780511546877

LaValle et al. 1998
LaValle, Steven M.; Hutchinson, Seth A., 1998.
Optimal motion planning for multiple robots having independent goals.
IEEE Transactions on Robotics and Automation, **14** (6), pp. 912–925.
DOI: 10.1109/70.736775

Lee et al. 1987
Lee, Beom H.; Lee, C. S. George, 1987.
Collision-Free Motion Planning of Two Robots.
IEEE Transactions on Systems, Man, and Cybernetics, **17** (1), pp. 21–32.
DOI: 10.1109/TSMC.1987.289330

Lee et al. 2014
Lee, Duck-Hyun; Kim, Dong-Hyun; Lee, Ji Yeong; Han, Chang-Soo, 2014.
Collision-free coordination of two dual-arm robots with assembly precedence constraint.
In: *14th International Conference on Control, Automation and Systems (ICCAS)*, pp. 515–520.
New York: IEEE.
ISBN 978-8-9932-1507-6.
DOI: 10.1109/ICCAS.2014.6988044

Lee et al. 2001
Lee, Sukhan; Moradi, Hadi, 2001.
A real-time dual-arm collision avoidance algorithm for assembly.
Journal of Robotic Systems, **18** (8), pp. 477–486.
DOI: 10.1002/rob.1038

Levine et al. 2016
Levine, Sergey; Finn, Chelsea; Darrell, Trevor; Abbeel, Pieter, 2016.
End-to-End Training of Deep Visuomotor Policies.
Journal of Machine Learning Research, **17** (39), pp. 1–40, visited on: 01/13/2018.
Available from: http://jmlr.org/papers/v17/15-522.html

Li et al. 2007
Li, Fajie; Klette, Reinhard, 2007.
Rubberband Algorithms for Solving Various 2D or 3D Shortest Path Problems.
In: *International Conference on Computing: Theory and Applications (ICCTA)*.
New York: IEEE.
ISBN 0-7695-2770-1.
DOI: 10.1109/ICCTA.2007.113

Li et al. 1995

Li, Tsai-Yen; Latombe, Jean-Claude, 1995.
On-line manipulation planning for two robot arms in a dynamic environment.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1048–1055.
New York: IEEE.
ISBN 0-7803-1965-6.
DOI: 10.1109/ROBOT.1995.525420

Likhachev et al. 2008

Likhachev, Maxim; Ferguson, Dave; Gordon, Geoff; Stentz, Anthony; Thrun, Sebastian, 2008.
Anytime search in dynamic graphs.
Artificial Intelligence, **172** (14), pp. 1613–1643.
DOI: 10.1016/j.artint.2007.11.009

Liu et al. 2010

Liu, Hong; Wan, Weiwei; Zha, Hongbin, 2010.
A dynamic subgoal path planner for unpredictable environments.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 994–1001.
New York: IEEE.
ISBN 978-1-4244-5038-1.
DOI: 10.1109/ROBOT.2010.5509324

Maciejewski et al. 1988

Maciejewski, Anthony A.; Klein, Charles A., 1988.
Numerical filtering for the operation of robotic manipulators through kinematically singular configurations.
Journal of Robotic Systems, **5** (6), pp. 527–552.
DOI: 10.1002/rob.4620050603

Makris et al. 2012

Makris, Sotiris; Michalos, George; Eytan, Amit A.; Chryssolouris, George, 2012.
Cooperating Robots for Reconfigurable Assembly Operations: Review and Challenges.
Procedia CIRP, **3**, pp. 346–351.
DOI: 10.1016/j.procir.2012.07.060

Makris et al. 2017
Makris, Sotiris; Tsarouchi, Panagiota; Matthaiakis, Aleksandros-Stereos; Athanasatos, Athanasios; Chatzigeorgiou, Xenofon; Stefos, Michael; Giavridis, Konstantinos; Aivaliotis, Sotiris, 2017.
Dual arm robot in cooperation with humans for flexible assembly.
CIRP Annals, **66** (1), pp. 13–16.
DOI: 10.1016/j.cirp.2017.04.097

Mamou et al. 2009
Mamou, Khaled; Ghorbel, Faouzi, 2009.
A simple and efficient approach for 3D mesh approximate convex decomposition.
In: *16th IEEE International Conference on Image Processing (ICIP)*, pp. 3501–3504.
New York: IEEE.
ISBN 978-1-4244-5654-3.
DOI: 10.1109/ICIP.2009.5414068

Mason 2001
Mason, Matthew T., 2001.
Mechanics of Robotic Manipulation.
New York: The MIT press, 2001.
ISBN 978-0-2622-5662-9.
DOI: 10.7551/mitpress/4527.001.0001

Müller 2016
Müller, Sebastian, 2016.
Bahnplanung und Kollisionserkennung für Zweiarmroboter im Bereich Montageautomatisierung.
Braunschweig, Univ., Master's thesis, 2016

Nakamura et al. 1986
Nakamura, Yoshihiko; Hanafusa, Hideo, 1986.
Inverse kinematic solutions with singularity robustness for robot manipulator control.
Journal of Dynamic Systems, Measurement, and Control, **108** (3), pp. 163–171.
DOI: 10.1115/1.3143764

Nilsson 1969
Nilsson, Nils J., 1969.
A Mobius Automaton: an Application of Artificial Intelligence Techniques.
In: *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pp. 1–26.
DOI: 10.21236/ada459660

O'Donnell et al. 1989
O'Donnell, Patrick; Lozano-Pérez, Tomás, 1989.
Deadlock-free and collision-free coordination of two robot manipulators.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 484–489.
New York: IEEE.
ISBN 0-8186-1938-4.
DOI: 10.1109/ROBOT.1989.100033

Ögren et al. 2012
Ögren, Petter; Smith, Christian; Karayiannidis, Yiannis; Kragic, Danica, 2012.
A Multi Objective Control approach to Online Dual Arm Manipulation.
10th IFAC Symposium on Robot Control, **45** (22), pp. 747–752.
DOI: 10.3182/20120905-3-HR-2030.00032

Pan et al. 2012
Pan, Zengxi; Polden, Joseph; Larkin, Nathan; Duin, Stephen Van; Norrish, John, 2012.
Recent progress on programming methods for industrial robots.
Robotics and Computer-Integrated Manufacturing, **28** (2), pp. 87–94.
DOI: 10.1016/j.rcim.2011.08.004

Peng et al. 2005
Peng, Jufeng; Akella, Srinivas, 2005.
Coordinating Multiple Robots with Kinodynamic Constraints Along Specified Paths.
The International Journal of Robotics Research, **24** (4), pp. 295–310.
DOI: 10.1177/0278364905051974

Petrič et al. 2011
Petrič, Tadej; Žlajpah, Leon, 2011.
Smooth transition between tasks on a kinematic control level: Application to self collision avoidance for two Kuka LWR robots.
In: *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 162–167.
New York: IEEE.
ISBN 978-1-4577-2138-0.
DOI: 10.1109/ROBIO.2011.6181279

Pham 2014
Pham, Quang-Cuong, 2014.
A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm.
IEEE Transactions on Robotics, **30** (6), pp. 1533–1540.
DOI: 10.1109/TRO.2014.2351113

Pierson et al. 2017
Pierson, Harry A.; Gashler, Michael S., 2017.
Deep learning in robotics: a review of recent research.
Advanced Robotics, **31** (16), pp. 821–835.
DOI: 10.1080/01691864.2017.1365009

Ratliff et al. 2009
Ratliff, Nathan; Zucker, Matthew; Bagnell, J. Andrew; Srinivasa, Siddhartha, 2009.
CHOMP: Gradient Optimization Techniques for Efficient Motion Planning.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 489–494.
New York: IEEE.
ISBN 978-1-4244-2788-8.
DOI: 10.1109/ROBOT.2009.5152817

Regele et al. 2006
Regele, Ralf; Levi, Paul, 2006.
Cooperative Multi-Robot Path Planning by Heuristic Priority Adjustment.
In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5954–5959.
New York: IEEE.
ISBN 1-4244-0258-1.
DOI: 10.1109/IROS.2006.282480

Rethink Robotics 2015
Rethink Robotics (ed.), 2015.
Baxter Research Software Developers Kit (SDK) [online]
Visited on: 01/25/2018.
Available from: http://sdk.rethinkrobotics.com/wiki/Main_Page

Saha et al. 2006
Saha, Mitul; Isto, Pekka, 2006.
Multi-Robot Motion Planning by Incremental Coordination.
In: *IEE0/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5960–5963.
New York: IEEE.
ISBN 1-4244-0258-1.
DOI: 10.1109/IROS.2006.282536

Salehian et al. 2017
Salehian, Seyed S.; Figueroa, Nadia; Billard, Aude, 2017.
Dynamical System-Based Motion Planning for Multi-Arm Systems: Reaching for Moving Objects.
In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pp. 4914–4918.
DOI: 10.24963/ijcai.2017/693

Sánchez et al. 2001
Sánchez, Gildardo; Latombe, Jean-Claude, 2001.
Using a PRM planner to compare centralized and decoupled planning for multi-robot systems.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2112–2119.
4914-4918.
ISBN 0-7803-7272-7.
DOI: 10.1109/ROBOT.2002.1014852

Sánchez et al. 2002
Sánchez, Gildardo; Latombe, Jean-Claude, 2002.
On Delaying Collision Checking in PRM Planning: Application to Multi-Robot Coordination.
The International Journal of Robotics Research, **21** (1), pp. 5–26.
DOI: 10.1177/027836402320556458

Scheitenberger 2015
Scheitenberger, Lars, 2015.
Implementierung eines Kommunikationsinterface zur Ansteuerung eines Zweiarmroboters im Bereich Montageautomatisierung.
Ulm, Univ., Master's thesis, 2015

Siciliano et al. 2016
Siciliano, Bruno; Khatib, Oussama (eds.), 2016.
Springer Handbook of Robotics.
2nd ed.
Berlin, Heidelberg: Springer, 2016.
ISBN 978-3-319-32552-1.
DOI: 10.1007/978-3-319-32552-1

Smith et al. 2012
Smith, Christian; Karayiannidis, Yiannis; Nalpantidis, Lazaros; Gratal, Xavi; Qi, Peng; Dimarogonas, Dimos; Kragic, Danica, 2012.
Dual arm manipulation – A survey.
Robotics and Autonomous Systems, **60** (10), pp. 1340–1353.
DOI: 10.1016/j.robot.2012.07.005

Spensieri et al. 2013
Spensieri, Domenico; Bohlin, Robert; Carlson, Johan S., 2013.
Coordination of robot paths for cycle time minimization.
In: *IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 522–527.
New York: IEEE.
ISBN 978-1-4799-1515-6.
DOI: 10.1109/CoASE.2013.6654032

Srivastava et al. 2014
Srivastava, Siddarth; Fang, Eugene; Riano, Lorenzo; Chitnis, Rohan; Russell, Stuart; Abbeel, Pieter, 2014.
Combined task and motion planning through an extensible planner-independent interface layer.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 639–646.
New York: IEEE.
ISBN 978-1-4799-3685-4.
DOI: 10.1109/ICRA.2014.6906922

Stenmark et al. 2016
Stenmark, Maj; Stolt, Andreas; Topp, Elin A.; Haage, Mathias; Robertsson, Anders; Nilsson, Klas; Johansson, Rolf, 2016.
The GiftWrapper: Programming a Dual-Arm Robot With Lead-through.
In: *ICRA Workshop on Human-Robot Interfaces for Enhanced Physical Interactions, Stockholm, 16.-21.05.2016.*
Available from: http://fileadmin.cs.lth.se/cs/Personal/Elin_Anna_Topp/pdf/stenmarkGiftWrapper.pdf

Sucan et al. 2012
Sucan, Ioan A.; Moll, Mark; Kavraki, Lydia E., 2012.
The Open Motion Planning Library.
IEEE Robotics & Automation Magazine, **19** (4), pp. 72–82.
DOI: 10.1109/MRA.2012.2205651.
http://ompl.kavrakilab.org

Sun et al. 2005
Sun, Zheng; Hsu, David; Jiang, Tingting; Kurniawati, Hanna; Reif, John. H., 2005.
Narrow passage sampling for probabilistic roadmap planning.
IEEE Transactions on Robotics, **21** (6), pp. 1105–1115.
DOI: 10.1109/TRO.2005.853485

Surynek 2010
Surynek, Pavel, 2010.
An Optimization Variant of Multi-robot Path Planning is Intractable.
In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)*, pp. 1261–1263.
Cambridge: AAAI Press, 2010.

Todt et al. 2000
Todt, Eduardo; Rausch, Gustavo; Suárez, Raúl, 2000.
Analysis and classification of multiple robot coordination methods.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3158–3163.
New York: IEEE.
ISBN 0-7803-5886-4.
DOI: 10.1109/ROBOT.2000.845149

Tsai et al. 2009
Tsai, Yi-Chih; Huang, Han-Pang, 2009.
Motion Planning of a Dual-Arm Mobile Robot in the Configuration-Time Space.
In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2458–2463.
New York: IEEE.
ISBN 978-1-4244-3803-7.
DOI: 10.1109/IROS.2009.5354154

Ulusoy et al. 2013
Ulusoy, Alphan; Smith, Stephen L.; Ding, Xu Chu; Belta, Calin; Rus, Daniela, 2013.
Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints.
The International Journal of Robotics Research, **32** (8), pp. 889–911.
DOI: 10.1177/0278364913487931

van den Berg et al. 2006
Van den Berg, Jur; Ferguson, Dave; Kuffner, James, 2006.
Anytime path planning and replanning in dynamic environments.
In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2366–2371.
New York: IEEE.
ISBN 0-7803-9505-0.
DOI: 10.1109/ROBOT.2006.1642056

van den Berg et al. 2011
Van den Berg, Jur; Guy, Stephen J.; Lin, Ming; Manocha, Dinesh, 2011.
Reciprocal n-Body Collision Avoidance.
In: Pradalier, Cédric; Siegwart, Roland; Hirzinger, Gerhard (Ed.): *Robotics Research*.
Berlin, Heidelberg: Springer, 2011, pp. 3–19.
ISBN 978-3-642-19457-3.
DOI: 10.1007/978-3-642-19457-3

Wagner et al. 2015
Wagner, Glenn; Choset, Howie, 2015.
Subdimensional expansion for multirobot path planning.
Artificial Intelligence, **219**, pp. 1–24.
DOI: 10.1016/j.artint.2014.11.001

Wampler 1986
Wampler, Charles W., 1986.
Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods.
IEEE Transactions on Systems, Man, and Cybernetics, **16** (1), pp. 93–101.
DOI: 10.1109/TSMC.1986.289285

Wang et al. 2017
Wang, Xinyu; Yang, Chenguang; Ju, Zhaojie; Ma, Hongbin; Fu, Mengyin, 2017.
Robot manipulator self-identification for surrounding obstacle detection.
Multimedia Tools and Applications, **76** (5), pp. 6495–6520.
DOI: 10.1007/s11042-016-3275-8

Yan et al. 2013
Yan, Zhi; Jouandeau, Nicolas; Cherif, Arab Ali, 2013.
A survey and analysis of multi-robot coordination.
International Journal of Advanced Robotic Systems, **10** (2), pp. 1–18.
DOI: 10.5772/57313

Yaskawa 2006
Yaskawa, Inc. (ed.), 2006.
Motoman NX 100 Controller: Arm Intereference Instructions Manual [online].
Rev. 1.
Kitakyushu, Japan: Yaskawa Inc., 2006 visited on: 02/23/2018.
Available from: https://www.motoman.com/getmedia/4321384B-6D45-4005-8745-D5E579D8897C/149648-2.pdf.aspx?ext=.pdf

Zhang et al. 2013 Zhang, Youmin; Mehrjerdi, Hasan, 2013.
A survey on multiple unmanned vehicles formation control and coordination: Normal and fault situations.
In: *International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1087–1096.
New York: IEEE.
ISBN 978-1-4799-0817-2.
DOI: 10.1109/ICUAS.2013.6564798

Der Trend zu einer immer weitergehenden Individualisierung von Endkundenprodukten stellt die Domäne der Fertigungsautomatisierung vor neue Herausforderungen. Zweiarmige und kollaborativ einsetzbare Industrieroboter bieten auf diesem Gebiet in vielen Fällen Potential für die Automatisierung von bisher manuell durchgeführten Arbeiten. Allerdings ist die Programmierung dieser Geräte durch die enge räumliche Anordnung der Arme nach wie vor komplex und zeitaufwendig. Insbesondere die Vermeidung von Kollisionen der sich asynchronen bewegenden Arme stellt den Nutzer dabei vor Herausforderungen. Der Einsatz eines spezialisierten und teuren Roboterprogrammierers zur Lösung dieses Problems verhindert einen wirtschaftlichen und flexiblen Einsatz von Geräten dieser Art häufig. Der vorliegende Band widmet sich deshalb der Frage, wie die Programmierung zweiarmiger Industrieroboter durch Algorithmen zur automatischen, kollisionsfreien Bewegungsplanung vereinfacht werden kann, um die Einstiegshürde für die potentiellen Nutzer möglichst weit zu senken.

ISBN 978-3-8396-1561-4

9 783839 615614

FRAUNHOFER VERLAG