Planning and Control for Industrial Manipulation

Philipp Sebastian Schmitt

Technische Fakultät
Albert-Ludwigs-Universität Freiburg

Dissertation zur Erlangung des akademischen Grades
Doktor-Ingenieur

Betreuer: Prof. Dr. Wolfram Burgard

Planning and Control for Industrial Manipulation

Philipp Sebastian Schmitt

Dissertation zur Erlangung des akademischen Grades Doktor-Ingenieur
Technische Fakultät, Albert-Ludwigs-Universität Freiburg

Dekan	Prof. Dr. Rolf Backofen
Erstgutachter	Prof. Dr. Wolfram Burgard
	Albert-Ludwigs-Universität Freiburg
Zweitgutachter	Prof. Dr. Bernhard Nebel
	Albert-Ludwigs-Universität Freiburg
Tag der Disputation	15. 12. 2020

Abstract

Manipulation tasks that are done manually today are likely automated tomorrow using robots. These tasks include industrial applications such as assembly or logistics. Potential applications also include robotic manipulators in households or in healthcare. The state-of-the-art approach to automate manipulation is to hold the manipulated objects in precise fixtures or part-feeders and to explicitly predetermine all motions of the robot. This approach is not viable when tasks or environments change frequently. A manipulation task might even be encountered only once. In these scenarios, the robot must compute its motions autonomously.

Computing motions and actions for manipulation is a challenging task for several reasons. The main difficulty arises due to the sequential interdependence of the motions of robots, the motions of manipulated objects, and the interactions of robots and objects. Consider the problem of putting an object into a box using a robot. To do so, the object must be grasped by the robot and moved into the box. However, the way the object is placed into the box constrains the way the object can be grasped as the robotic gripper should not collide with the box. As a result, the motions of the robot and the object must be computed simultaneously. This problem is called manipulation planning. Manipulation planning involves reasoning about high-dimensional state spaces with both continuous and discrete variables. In some applications it is important to minimize the cycle times of manipulation tasks. This is especially important in the context of industrial automation. For this reason, manipulation planners should, ideally, compute optimal motions for robots and objects. There exists a vast variety of different robots, manipulation tasks, and interactions between robots and objects. It is therefore necessary to employ models, that capture the constraints of this variety of planning problems. Finally, when the environments of robots change dynamically, motions and actions must be adapted online and in real-time.

The contributions of this thesis are algorithmic techniques to address these challenges of manipulation:

(1) We propose an asymptotically optimal, sampling-based manipulation planner. This planner efficiently computes high-quality solutions to realistic manipulation problems. We prove probabilistic completeness and asymptotic optimality under a novel set of robustness conditions.

(2) To address the variety of manipulation tasks, we propose a novel, constraint-based model for manipulation. This model captures properties of manipulation tasks, such as closed kinematic chains and time-variant environments. From this model we automatically

derive constraint-based motion controllers and propose algorithms for feedback planning that utilize these controllers. This enables manipulation where robots react in real-time to estimates of the environment. Examples include reactions to measurements of object poses and detected obstacles.

(3) Finally, we propose models and algorithms to include additional process steps within manipulation. An example for such a process step would be joining two parts during assembly. These processes are modeled as robotic skills. A sequence of skills and their parameters are computed by a novel and asymptotically optimal task and motion planner. This approach enables to automate complex, industrial assembly and is demonstrated with the assembly of an electrical switching cabinet.

We evaluated all proposed approaches in extensive simulated experiments and on real robots. For the optimal planners, the experiments support the optimality guarantees and show significant improvements with respect to state-of-the-art planners.

Zusammenfassung

In vielen industriellen Automatisierungslösungen werden Gegenstände von Robotern manipuliert. Beispiele hierfür sind Montage oder Intralogistik. Die gängige Methode die Bewegungen der Roboter zu programmieren, ist der sogenannte Teach-In. Dabei werden die zu manipulierenden Objekte in präzisen Fixturen oder Teilezuführungen bereitgestellt. Alle Zwischenpunkte der Bewegungen des Roboters werden einmal angefahren, abgespeichert und für die eigentliche Produktion wieder abgespielt. Dieser Ansatz hat sich für große Stückzahlen seit Jahrzehnten bewährt, stößt jedoch bei kleinen Stückzahlen schnell an wirtschaftliche Grenzen. Im Extremfall wird ein Produkt nur einmal hergestellt. In diesem Fall ist es notwendig, dass der Roboter seine Bewegungen selbst berechnet und dabei mit weniger strukturierten Umgebungen ohne Fixturen umgehen kann. Insbesondere kleine und mittelständische Betriebe könnten von einer autonomen Bewegungsgenerierung profitieren, da hier kleine Losgrößen häufiger auftreten. Manipulatoren die ihre Bewegungen autonom erzeugen könnten zudem in vielen nicht-industriellen Anwendungen hilfreich sein. Beispiele hierfür sind Haushaltsroboter, Lieferroboter oder Systeme zur Unterstützung bei Kranken- und Altenpflege. Diese Anwendungen zeichnen sich ebenfalls durch eine hohe Varianz der Aufgaben und wenig strukturierte Umgebungen aus.

Das Ziel dieser Arbeit ist es innovative Modelle und Algorithmen zu entwickeln, die es einem Robotersystem ermöglichen autonom Bewegungen für Manipulationsaufgaben zu erzeugen. Diese sollten die folgenden Randbedingungen erfüllen: (1) Insbesondere für industrielle Anwendungen sind die Zykluszeiten einer Automatisierungslösung entscheidend. Aus diesem Grund sollte eine autonome Bewegungsgenerierung nicht nur eine Manipulationsaufgabe lösen, sondern dies mit optimalen Bewegungen erreichen. (2) Es existiert eine Vielfalt von Robotersystemen, Umgebungen und Manipulationsaufgaben. Daher ist es entscheidend, dass die Modellierung der Problemstellung über die Kapazität verfügt diese Vielfalt abzubilden. (3) In manchen Anwendungen ändert sich die Umgebung des Robotersystems dynamisch. Ein Beispiel hierfür sind Personen die sich gemeinsam mit einem mobilen Manipulator in einem Warenlager bewegen. Für diese Anwendungen ist es unerlässlich, dass der Manipulator seine Bewegungen in Echtzeit an die sich ändernde Umgebung anpasst.

Aus der Zielsetzung und den Randbedingungen dieser Arbeit ergeben sich einige Herausforderungen, die im Folgenden anhand von Beispielen erläutert werden. Betrachten wir einen Manipulator der ein Objekt in einen Behälter legen soll. Dazu ist es nötig, dass der Manipulator das Objekt greift, es in den Behälter transportiert und sich danach wieder in

seine Ursprungskonfiguration zurückbewegt. Zum Ablegen des Objekts ist es nötig, dass das Objekt zuvor mit der richtigen Orientierung gegriffen wurde, da ansonsten der Manipulator mit dem Behältnis kollidieren würde. Schlussendlich muss sowohl das Aufheben des Objekts als auch das Ablegen so erfolgen, dass sich der Manipulator kollisionsfrei aus dem Behältnis zurückziehen kann. Dieses Beispiel illustriert die sequentiellen Abhängigkeiten von Bewegungen und Prozessschritten, die bei Manipulationsaufgaben entstehen. Um diese Abhängigkeiten systematisch aufzulösen ist es nötig die gemeinsamen Bewegungen von Robotern und Objekten über mehrere Schritte hinweg zu planen. Dieses Problem wird als Manipulationsplanung bezeichnet. Manipulationsplanung findet in hoch-dimensionalen Zustandsräumen statt und muss sowohl kontinuierliche Größen (Bewegungen, Greif-Posen) als auch diskrete Variablen (Abfolge von Aktionen) berücksichtigen. Dieses hybride Planungsproblem ist insbesondere dann anspruchsvoll wenn optimale Lösungen gefordert sind oder wenn Bewegungen in Echtzeit erzeugt werden müssen. Dies liegt daran, dass nicht nur eine einzelne Bewegung des Manipulators berechnet werden muss, sondern eine Sequenz von untereinander abhängigen Bewegungen von Roboter und Objekten.

Um die Vielzahl von existierenden Manipulationsaufgaben darstellen zu können werden Modelle benötigt, die die Randbedingungen und die Dynamik dieser Aufgaben abbilden. Typische Randbedingungen sind Achsbegrenzungen oder Kollisionsvermeidung. Je nach Aufgabe können sich die Randbedingungen aber erheblich unterscheiden. Ein Beispiel hierfür ist ein Delta-Roboter, der Objekte von einem Förderband greifen soll. Der Roboter selbst ist hierbei eine geschlossene kinematische Kette, was bei der Planung berücksichtigt werden muss. Zudem ist die Dynamik der Umgebung zeitvariant aufgrund des sich bewegenden Förderbands. Ein weiteres Beispiel ist ein Schweißroboter der Metallteile verbindet. Das Verbinden der Teile mittels eines Schweißgeräts ist dabei ein Bestandteil der Aufgabe, der modelliert und ausgeführt werden muss. Allerdings liegt die Wirkung und auch die Ausführung (Aktivierung des Schweißgeräts) nicht in einer Bewegung des Roboters oder der Objekte. Es ist daher nötig Modelle zu verwenden, die derartige prozessbezogenen Eigenschaften einer Aufgabe abbilden können.

Der Beitrag dieser Arbeit sind Modelle und Methoden die autonome Bewegungsgenerierung für Manipulation ermöglichen und die zuvor genannten Randbedingungen und Herausforderungen adressieren. Hierzu stellen wir folgende Ansätze vor:

1. Asymptotisch optimaler Manipulationsplaner: Dieser Planer erweitert einen asymptotisch optimalen Bewegungsplaner, sodass sequentielle Manipulationsaufgaben optimal gelöst werden können. Unter Annahme von Robustheitseigenschaften beweisen wir asymptotische Optimalität. Der Planer berechnet in weniger als einer Sekunde hochqualitative Bewegungen für ein reales Manipulationsproblem.

2. Generische Modelle für Manipulation und reaktive Planung: Um Manipulation in dynamischen Umgebungen abzubilden stellen wir ein innovatives Model für Manipulationsprobleme vor. Dieses Modell ist insbesondere dafür geeignet automatisch

Regler zu erstellen die zielgerichtete, reaktive Bewegungen für Manipulation erzeugen. Ein Beispiel hierfür ist ein Regler der einen Roboter ein Objekt greifen lässt. Dies gilt auch für Systeme mit komplexen Randbedingungen wie geschlossenen kinematischen Ketten und zeitvarianter Dynamik. Basierend auf dem vorgestellten Modell und den davon abgeleiteten Reglern stellen wir zwei Planungsansätze vor, die eine reaktive Manipulationsplanung ermöglichen.

3. Kombinierte Aufgaben- und Bewegungsplanung in einem Skill Framework: Um Prozess-Schritte abzubilden die nicht ausschließlich aus einer Bewegung bestehen wird ein innovatives Model von Roboter-Fähigkeiten, sogenannte Skills, vorgestellt. Ein wesentlicher Aspekt dieser Skills ist es, dass ihre Formulierung es ermöglicht in einem kombinierten Aufgaben- und Bewegungsplaner verwendet zu werden. Hierbei werden Skills von einem optimalen Planer aneinander gereiht und parametriert. Dies geschieht in einer Regelschleife in der kontinuierlich neu geplant wird. Der Ansatz wird an einer industriellen Schaltschrank-Montage validiert.

Diese Arbeit ist in sechs Kapitel unterteilt. Im ersten Kapitel werden die Problemstellungen anhand von einigen Beispielen motiviert. Zudem werden die Forschungsfragen und Beiträge der Arbeit erläutert.

Das zweite Kapitel gibt einen Überblick über die Grundlagen auf denen diese Arbeit aufbaut. Dies beinhaltet die Beschreibung kinematischer Bäume in generalisierten Koordinaten und die in dieser Arbeit verwendete Notation. Außerdem werden die Sampling-basierte Bewegungsplanung sowie Manipulationsplanung kurz diskutiert.

Idealerweise sollten Manipulationsaufgaben nicht nur gelöst werden, sondern optimale Bewegungen erzeugt werden. Hierzu stellt Kapitel drei einen optimalen, Sampling-basierten Manipulationsplaner vor. Dieser Planer erweitert einen optimalen Bewegungsplaner, sodass der gemeinsame Konfigurationsraum von Roboter und Objekten effizient und im Grenzwert optimal durchsucht wird. Es wird zudem ein Satz von Robustheitsannahmen vorgestellt unter denen die asymptotische Optimalität des Planers bewiesen wird. Um die Planungszeiten zu reduzieren, wird eine generische Strategie vorgestellt, die es ermöglicht einmal durchgeführte Berechnungen mehrfach zu verwenden. Mit diesem Ansatz werden in einem realistischen Manipulationsszenario in weniger als einer Sekunde hochqualitative Bewegungen berechnet.

Wie zuvor diskutiert, existiert eine Vielzahl von Robotersystemen und Manipulationsaufgaben. Wenn sich die Umgebung des Roboters dynamisch ändert, müssen die Bewegungen des Roboters in Echtzeit angepasst werden. Kapitel vier stellt eine generische Modellierung für Manipulationsaufgaben vor, mit der diese beiden Herausforderungen gelöst werden. Hierbei ist die wesentliche Idee, automatisch einen Satz von Reglern aus der Modellierung abzuleiten. Diese Regler erzeugen dabei einzelne, zielgerichtete Bewegungen innerhalb einer Manipulationssequenz. Beispiele sind Regler, die Objekte aufheben oder ablegen oder eine Tür öffnen oder schließen. Um die Manipulationsaufgabe zu lösen ist es ausreichend zwischen diesen Reglern umzuschalten. Hierfür stellt Kapitel vier zwei Ansätze vor. Im ersten Ansatz werden die Regler von einem kinodynamischen Planer in eine lineare Sequenz gereit. Das Ergebnis dieses Planers ist keine Trajektorie, sondern eine Sequenz von Reglern, die bei der Ausführung auf Störungen reagieren. Beispiele für solche Störungen sind Kollisionskörper, die zur Laufzeit entdeckt werden, oder verschobene Objekte. Ein wesentlicher Aspekt hierbei ist es, dass die Reaktion der Regler die ursprüngliche Aufgabenmodellierung berücksichtigt. Der zweite Ansatz basiert darauf das Umschalten der Regler als ein Reinforcement Learning Problem zu modellieren. In Simulation kann damit ein Agent trainiert werden, der in Echtzeit zwischen den einzelnen Reglern umschaltet. Der Vorteil dieses zweiten Ansatzes ist eine globale Abbildung von Zuständen auf aktive Regler. Hiermit kann auf größere Störungen reagiert werden. Ein Beispiel ist ein Objekt das von einem Roboter unbeabsichtigt fallen gelassen wird. Mit diesem Ansatz wird das Objekt direkt wieder aufgehoben und die Manipulationsaufgabe fortgesetzt.

Für manche Manipulationsaufgaben sind zusätzliche Prozessschritte nötig, die sich nicht sinnvoll als Planungsproblem über Bewegungen von Robotern und Objekten darstellen lassen. Ein Anwendungsbeispiel bei dem dies auftritt ist die Montage von elektrischen Komponenten in einem Schaltschrank. Hierbei werden die Komponenten zwar manipuliert, müssen aber über eine bestimmte Bewegungsfolge mit dem Schaltschrank oder untereinander verbunden werden. Für die praktische Anwendung von Manipulationsplanung ist es hilfreich dem Planer dieses Vorwissen bereitzustellen. Zu diesem Zweck wird in Kapitel fünf ein neues Modell für Fähigkeiten des Roboters, sogenannte Skills, vorgestellt. Dieses Modell ist so gestaltet, dass eine Sequenz von parametrierten Skills von einem kombinierten Aufgaben- und Bewegungsplaner erstellt werden kann. Der ebenfalls neu vorgestellte Planer berechnet asymptotisch optimale Sequenzen von Roboterbewegungen und Skills. Mittels einer informierten Suchstrategie kann diese Planung zur Laufzeit regelmäßig wiederholt werden um den Plan an neue Sensorinformationen anzupassen. In Experimenten auf einem realen Multi-Roboter-System wurde mit diesem Ansatz ein Teil einer industriellen Schaltschrank-Montage realisiert.

Im letzten Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und diskutiert. Zusätzlich wird ein Ausblick über mögliche zukünftige Forschungsfragen gegeben. Die Beiträge dieser Arbeit sind die Entwicklung, Analyse und experimentelle Validierung innovativer Modelle und Planungsmethoden für Manipulation. Dies wird erreicht, indem Methoden aus den Bereichen Manipulationsplanung, optimale Bewegungsplanung und Constraint-basierte Aufgabenspezifikation und Regelung integriert und erweitert werden. Alle entwickelten Ansätze wurden in ausführlichen Experimenten in Simulation und auf realen Robotern validiert. Für die optimalen Planungsmethoden zeigen diese Experimente eine signifikante Verbesserung der berechneten Bewegungen gegenüber vorherigen Ansätzen. Die Regler-basierten Ansätze dieser Arbeit ermöglichen eine reaktive Manipulation, die auch in dynamischen Umgebungen eingesetzt werden kann. Hierdurch leistet diese Arbeit einen Beitrag autonome Manipulation in weiteren Gebieten zu ermöglichen.

Acknowledgements

This work would not have been possible without the support of my colleagues at Siemens and of Professor Wolfram Burgard at the University of Freiburg. Professor Burgard supervised this thesis and provided valuable advice that shaped my understanding of developing algorithms. Georg von Wichert made this work possible and strengthened my understanding of model-based autonomous systems. I would like to thank my advisor, Kai Wurm, for his support, especially by shaping and focusing my scientific writing. The dual-robot setup used throughout this thesis was constructed and maintained together with my colleagues and friends Vincent Dietrich, Bernd Kast, and Florian Wirnshofer.

I would like to thank the following people for proof-reading earlier versions of this thesis or the preceding publications: Sebastian Albrecht, Wolfram Burgard, Vincent Dietrich, Robert Eidenberger, Wendelin Feiten, Bernd Kast, Daniel Meyer-Delius, Werner Neubauer, Georg von Wichert, Florian Wirnshofer, and Kai Wurm.

The work presented in this thesis was funded and supported by the Siemens AG. Siemens provided a great work environment with wonderful colleagues and inspiring, industrial use cases for robotics. Furthermore, Siemens granted access to an excellent robotics laboratory that enabled the real world experiments presented in this thesis.

Parts of the presented work were funded by the TransFit project. TransFit is funded by the German Federal Ministry of Economics and Technology (BMWi) according to a resolution of the German Bundestag, grant no. 50RA1701, 50RA1702, and 50RA1703.

Finally, I would like to thank my family for the love and support throughout the years.

Contents

Chapter 1

Introduction

Autonomous robotic manipulators have the potential to automate or assist a rich variety of activities in several application domains. Robots that manipulate objects have proven useful in industrial manufacturing. Examples include assembly, packaging, and logistics. Also, many non-industrial domains could benefit from robotic manipulation. Robots in households or care-facilities could support the elderly in manipulation tasks such as preparing meals. Another field of application lies in areas that are inaccessible to humans. This includes disaster recovery or space applications. Techniques developed for real world manipulators could also be used in virtual environments, e. g., to animate non-player characters in video games.

Most of these applications are out of reach for state-of-the-art methods to program and deploy robots. Today, the main use of robots is in repetitive tasks in industrial mass production. To automate these tasks the environments of robots are structured for the task at hand and the motions of robots are predetermined for each application. Typically, the manipulated objects are held in precise fixtures or part-feeders. Cages prevent humans from approaching and thus render the robot's environment predictable. When the task changes, the environment of the robot must be restructured and the motions re-programmed. For small lot sizes this approach is not economically viable. A product could even be manufactured only once. In environments with less structure, e. g., households, this approach is not feasible at all. These use cases can only be automated with robots that autonomously compute their motions to fulfill a manipulation task.

In this thesis we propose and assess novel algorithms for autonomous manipulation. We aim for robots that effectively and efficiently manipulate objects in a rich variety of tasks and environments without detailed programming. This is a challenging problem for the following reasons: In order to manipulate objects, robots must reason about a sequence of motions and interactions with the manipulated objects. For efficient manipulation, this reasoning should compute optimal motions. When the environment of a robot changes dynamically, motions and actions must be adapted in real-time. There is an enormous variety of robots, end-effector tools, environments, and tasks. Hence, the autonomous motion generation must be grounded in models that capture this variety.

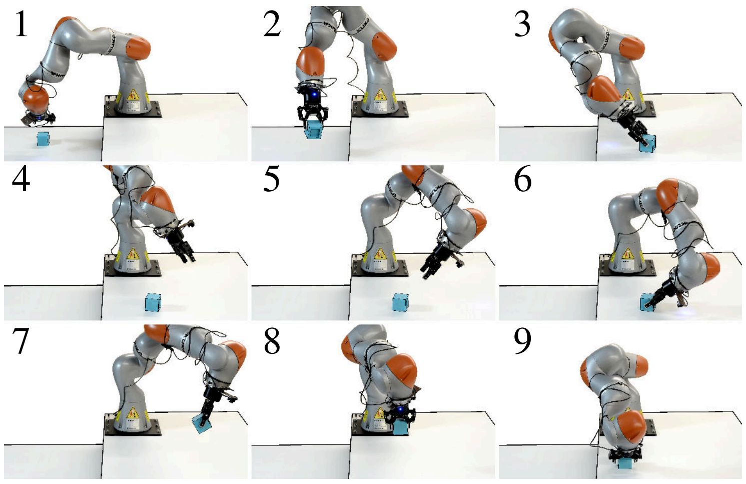


Figure 1.1: A manipulation problem: The task for the robot is to place a cube upside down on the right table. To reorient the cube it must be (re-)grasped at least twice. The sequence of grasp poses, placement poses, and the motions of robot and object are interdependent. For example, if the robot grasps the object with a wrong orientation it cannot be placed correctly with the next motion.

1.1 Research Questions

In the following we motivate our research questions and illustrate their relevance with a set of examples. Let us consider the exemplary manipulation task depicted in Figure 1.1. In this example a robot must transport a cube onto a table and place it upside down. A naive approach to manipulation would be to treat grasping and placing the cube separately. One could compute a grasp pose, compute a robot configuration that solves the inverse kinematics problem of positioning the gripper, and then plan a collision free motion towards that configuration. Once the object is grasped, similar steps are repeated for placing it again. This naive approach is likely to fail as the sequential robot motions, grasps, and placements are interdependent. In the example at least two grasps and placements are necessary to transport and turn the object. To solve this manipulation problem reliably it is necessary to reason about a sequence of motions of the robot and the object simultaneously. This is known as manipulation planning.

The interdependencies in a manipulation task do not only have consequences for the reliability of manipulation but also for its efficiency. For example, if the sequence of grasps and placements is not chosen well the robot may have to perform large intermediate motions in order to avoid collisions or axis limits. Cycle times of robotic manipulation are important, especially in industrial applications. For this reason it would be beneficial if manipulation planning provided optimal sequences of motions for robots and objects. This leads to the first research question of this thesis:

Research Question 1: How can a robot compute optimal sequences of motions and interactions with objects?

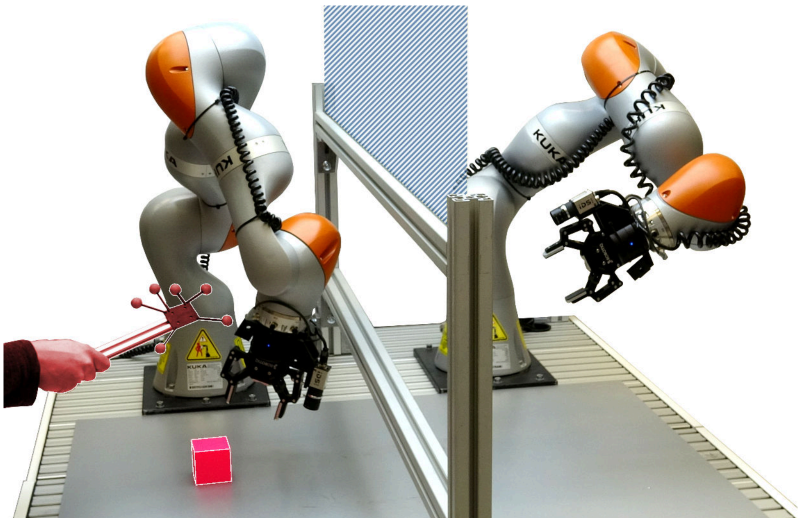


Figure 1.2: Reactive manipulation in a dynamic environment: Two robots must transport a cube to the right side of the setup. A human enters the workspace and is detected via markers. The robots must avoid collisions in real-time. Avoiding the human must not result in self-collisions of the robots. If possible, redundant degrees of freedom should be used to continue with the task.

Robots and manipulated objects form dynamic systems. Furthermore, models of these systems and estimates of their current state are inherently inaccurate. This results in deviations between the execution of a planned sequence of motions and the original plan. An open-loop execution of planned trajectories is therefore likely to fail eventually. For this reason autonomous manipulation benefits from closed-loop, real-time reactions to disturbances and estimates of a robot's environment. In some applications the environments of robots change dynamically as well. An example would be humans and robots that operate in a shared workspace as shown in Figure 1.2. In these scenarios robots must adapt their motions to the changing environment in real-time. To be safe and useful a reactive manipulation must address several challenges. Reacting to external disturbances must not violate the constraints of the manipulation task. For example, a robot must not violate its axis limits while avoiding collisions with a human worker. When safety constraints are fulfilled a robot should use remaining degrees of freedom to continue the execution of its task. In some cases reactions must include changing the sequence of motions and interactions with manipulated objects. For example, if an object is accidentally dropped it must be picked up again before continuing with the task. These challenges are subsumed in the second research question:

Research Question 2: How can a robot reason about sequences of dynamic motions and interactions with objects in real-time?

There exists a rich variety of robot kinematics and geometries, end-effectors, environments, and manipulation tasks. The combination of hardware setup and task defines the constraints on safe and goal-directed motions for a manipulator. For this reason it

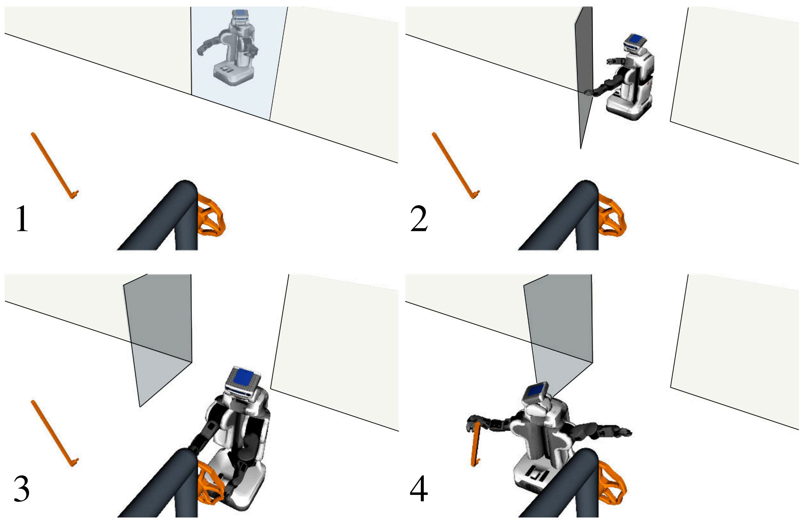


Figure 1.3: Constrained manipulation task: A mobile manipulator has to operate a valve and a lever. It must first open a door that blocks the path. As additional constraint, the head-mounted camera must point at the lever, door, or valve respectively while they are being manipulated.

is necessary to employ models that can capture these constraints. In many applications these constraints go beyond what can be modeled as collision free motion planning. An example can be seen in Figure 1.3. In this example a mobile manipulator has to operate a door, a valve, and a lever. When the robot manipulates these objects closed kinematic chains are formed that constrain the motion of the robot and the objects. Additionally, the head-mounted camera of the robot must point at any object that is currently manipulated. The choice of a suitable model for these constraints is a challenge for describing tasks, planning motions, and controlling motions during execution.

Manipulation may also include process steps that cannot be meaningfully modeled as motions of robots or objects. An example would be an arc welding task in which a robot has to join parts by moving and activating an arc welding torch. Another example is shown in Figure 1.4. In this example a dual-arm robot must assemble an electrical component on a top-hat rail. The robot has to perform a force-controlled assembly that is specific to this scenario. Both for planning and execution the underlying models must capture this process related knowledge. This leads to the third research question:

Research Question 3: What models are suitable to describe, plan, and control manipulation?

1.2 Key Contributions

The contributions of this thesis are novel models and algorithms to plan and control robotic manipulation. Our work integrates and extends the research on manipulation planning,

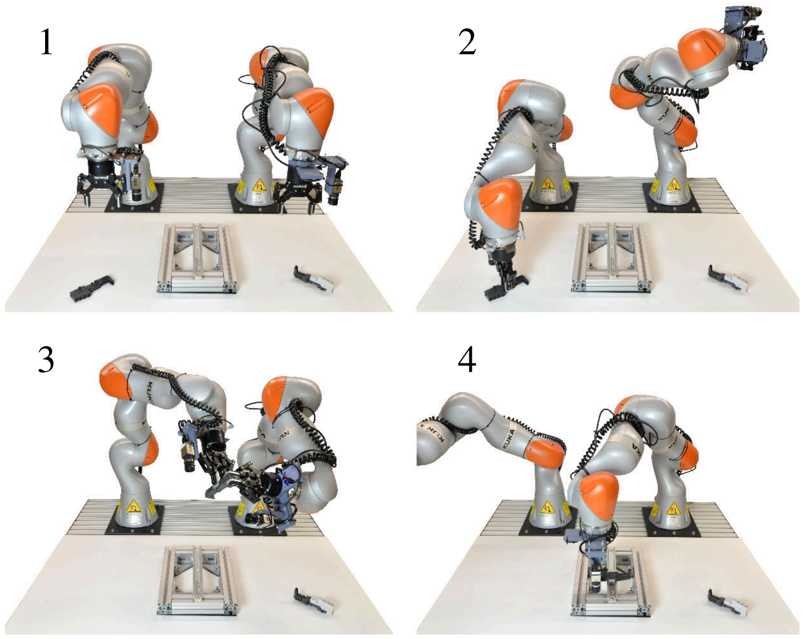


Figure 1.4: Manipulation with additional process steps: A dual-arm robot must mount an electrical component onto a top-hat rail. The component must be re-oriented via a handover before the actual assembly. The force-controlled assembly strategy is a process step that must be included in the planning domain.

optimal motion planning, and constraint-based task specification and control. The goal is to make robotic manipulation more flexible, efficient, and reliable. This thesis proposes:

- an asymptotically optimal manipulation planner (Chapter 3),
- a novel and highly flexible domain model for constrained manipulation tasks in dynamic environments (Chapter 4),
- algorithms to create feedback plans that enable reactive manipulation (Chapter 4),
- an asymptotically optimal task and motion planner (Chapter 5), and
- a control architecture for integrated task and motion planning for industrial manipulation (Chapter 5).

1.3 Outline

The remainder of this thesis is structured as follows. In Chapter 2 we introduce the background for this thesis. This includes modeling the state of motion of robots and objects and a brief introduction to sampling-based motion planning and manipulation planning.

When robots perform manipulation tasks autonomously, they need to determine their own movement, as well as how to grasp and release objects. Reasoning about motions of robots and objects simultaneously leads to a multi-modal planning problem in a high-dimensional configuration space. In Chapter 3 we propose an asymptotically optimal manipulation planner. This approach extends an optimal sampling-based motion planner to efficiently and optimally explore the configuration space of robots and objects. We introduce a novel set of robustness conditions under which we prove probabilistic completeness and global, asymptotic optimality. Furthermore, we introduce generic strategies for the preprocessing of planning problems. This preprocessing allows to re-use computations, such as collision checks, across different modes of the manipulation problem. Our planner efficiently computes high quality solutions to realistic manipulation problems. We implemented this approach for a set of typical pick and place scenarios in simulation and on a real robot. Extensive simulations show that this approach significantly outperforms a state-of-the-art planner.

In Chapter 4 we propose methods to control motions and interactions with objects in real-time. This is necessary whenever a robot's environment changes dynamically. An important insight of this chapter is that planning complex, constrained manipulation and reactive execution are both enabled by suitable underlying models. To this end we propose an innovative model for manipulation problems, the dynamic constraint graph. In this model the different modes or contact states of manipulation are associated with a set of constraint functions. These modes form a graph that encodes potential changes in contact states, e. g., for grasping or releasing objects. This model extends a state-of-the-art model by incorporating second-order dynamics and time-variance. From this model we automatically derive a set of constraint-based controllers. Reactive manipulation can then be implemented by switching between these reactive, low-level controllers. We present two methods that perform this switching of controllers: one is based on a kinodynamic planner, the other formulates the selection of the active controller as a reinforcement learning problem. A key advantage of this approach is that the controllers adhere to the constraints of the planning domain. For example, when a robot executes a plan and simultaneously avoids collision with a human worker, it will not cause self-collisions as a result. We implemented these approaches for several distinct robots, environments, and tasks. This includes time-variant environments and robots with closed kinematic chains. In real world experiments the proposed approaches enable sequential manipulation with real-time reactions to detected obstacles and measurements of object poses.

A key aspect of manipulation is that it often involves process steps that cannot be meaningfully modeled as motions of robots and rigid objects. Industrial examples are joining processes such as welding, gluing, or snap fit assemblies. In Chapter 5 we propose models and planning algorithms to address these processes in industrial manipulation. We introduce an innovative model of robotic skills, that encapsulates simulation and execution of motions and process steps. This skill model is designed to be used within a combined task and motion planner to create a sequence of parametrized skills as plans. For this purpose we also propose an asymptotically optimal combined task and motion planner. Under a set of robustness conditions we proof probabilistic completeness and asymptotic optimality of the planner. Furthermore, we introduce an informed search strategy that speeds up planning and convergence towards optimal plans. We implement our skill model and planner for an industrial assembly use case. In this use case a real robot assembles electric components on a top-hat rail as typically used in switching cabinets. The efficiency of the proposed planning approach enables to continuously plan and re-plan sequences of skills during execution.

We summarize and discuss the results of this thesis in Chapter 6. Additionally, we provide an outlook of potential avenues for future research.

1.4 Publications

Intermediate results of this thesis have been published in proceedings of peer reviewed conferences. The contents of these publications were developed in pursuit of this thesis with the author of this thesis as main contributing author. For this reason contents of these publications, including text passages and images, are used without further citation throughout this thesis.

- **Planning Reactive Manipulation in Dynamic Environments**

  P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G.v. Wichert, W. Burgard

  2019 IEEE/RSJ International Conference on Intelligent Robots and Systems

  Best Conference Paper Award - Winner

  © IEEE 2019

- **Modeling and Planning Manipulation in Dynamic Environments**

  P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G.v. Wichert, W. Burgard

  2019 IEEE International Conference on Robotics and Automation

  © IEEE 2019

- **Optimal, Sampling-Based Manipulation Planning**

  P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G.v. Wichert, W. Burgard

  2017 IEEE International Conference on Robotics and Automation

  Best Robotic Manipulation Paper Award - Finalist

  © IEEE 2017

The following list shows publications that resulted from collaborations. Their contents are not the main focus of this thesis. These publications are cited in this thesis where the respective contents are used.

- **Controlling Contact-Rich Manipulation under Partial Observability**

  F. Wirnshofer, P. S. Schmitt, G.v. Wichert, W. Burgard

  2020 Robotics: Science and Systems

- **Hierarchical Planner with Composable Action Models for Asynchronous Parallelization of Tasks and Motions**

  B. Kast, P. S. Schmitt, S. Albrecht, W. Feiten, J. Zhang

  2020 IEEE: International Conference on Robotic Computing

- **Robust, Compliant Assembly with Elastic Parts and Model Uncertainty**

  F. Wirnshofer, P. S. Schmitt, P. Meister, G.v. Wichert, W. Burgard

  2019 IEEE/RSJ International Conference on Intelligent Robots and Systems

- **State Estimation in Contact-Rich Manipulation**

  F. Wirnshofer, P. S. Schmitt, P. Meister, G.v. Wichert, W. Burgard

  2019 IEEE International Conference on Robotics and Automation

- **Robust, Compliant Assembly via Optimal Belief Space Planning**

  F. Wirnshofer, P. S. Schmitt, W. Feiten, G.v. Wichert, W. Burgard

  2018 IEEE International Conference on Robotics and Automation

- **Configuration of Perception Systems via Planning over Factor Graphs**

  V. Dietrich, B. Kast, P. S. Schmitt, S. Albrecht, M. Fiegert, W. Feiten, M. Beetz

  2018 IEEE International Conference on Robotics and Automation

1.5 Notation, Recurring Symbols, and Abbreviations

Notation for mathematical expressions:

Notation	Meaning
x	scalar
$\mathbf{x}$	column vector
$\mathbf{x}^\top$	row vector
A	matrix
$\mathbf{0}_k$	column vector of k zeros
I_k	$k \times k$ identity matrix
$\frac{\partial f}{\partial x}$	partial derivative of f with respect to x
$\{\ldots\}$	set
$(\cdot)_\mathrm{r}$	subscript r denotes a quantity related to one or multiple robots
$(\cdot)_\mathrm{o}$	subscript o denotes a quantity related to one or multiple objects

Recurring Symbols:

Symbol	Domain	Meaning
n_r	$\mathbb{N}$	dimensionality of the robot configuration
n_o	$\mathbb{N}$	dimensionality of the object configuration
n	$\mathbb{N}$	dimensionality of a configuration ($n = n_\mathrm{r} + n_o$)
$\mathbf{q}_\mathrm{r}$	$\mathbb{R}^{n_\mathrm{r}}$	configuration of one or multiple robots
$\mathbf{q}_\mathrm{o}$	$\mathbb{R}^{n_\mathrm{o}}$	configuration of one or multiple objects
$\mathbf{q}$	$\mathbb{R}^n$	configuration ($\mathbf{q} = \left[\mathbf{q}_\mathrm{r}^\top, \mathbf{q}_\mathrm{o}^\top\right]^\top$)
$\dot{\mathbf{q}}$	$\mathbb{R}^n$	velocity or time-derivative of the configuration
$\ddot{\mathbf{q}}$	$\mathbb{R}^n$	acceleration or second time-derivative of the configuration
T		Set of trajectories through configuration space
τ	T	Trajectory of configurations $\tau : [t_\mathrm{start}, t_\mathrm{end}] \to \mathbb{R}^n$
n_u	$\mathbb{N}$	dimensionality of the control input
$\mathbf{u}$	$\mathbb{R}^{n_\mathrm{u}}$	control input
Φ		Set of control trajectories
ϕ	Φ	Trajectory of controls $\phi : [t_\mathrm{start}, t_\mathrm{end}] \to \mathbb{R}^{n_\mathrm{u}}$
Σ		set of discrete modes or contact states
σ	Σ	discrete mode or contact state
Π		set of contact parameterizations
π	Π	contact parameterization
t	$\mathbb{R}$	time
$\mathcal{N}$		Set of nodes of a graph
$\mathcal{E}$		Set of edges of a graph

Abbreviations:

Abbreviation	Meaning
CR-DQN	Constrained, Reactive Deep Q-Network
CR-EST	Constrained, Reactive Expansive Space Tree
DoF	Degree of Freedom
DQN	Deep Q-Network
EST	Expansive Space Tree
eTaSL	expression graph-based Task Specification Language
eTC	expression graph-based Task Controller
FOBT	Factored Orbital Bellman Tree
iTaSC	instantaneous Task Specification using Constraints
LGP	Logic Geometric Programming
MM-EST	Multi Modal Expansive Space Tree
MM-PRM	Multi Modal Probabilistic Roadmap
PRM	Probabilistic Roadmap
PRM*	Optimal Probabilistic Roadmap
PTR	Probabilistic Tree of Roadmaps
RMR*	Optimal Random Manipulation Roadmap
RRT	Rapidly Exploring Random Tree
RRT*	Optimal Rapidly Exploring Random Tree

Chapter 2

Basics

The aim of this chapter is to give a brief overview of the models and algorithms for motion planning and control that form the building blocks of this thesis. A comprehensive overview of planning algorithms and the corresponding models is provided by LaValle [1]. Before we can plan or control motions of a robot, we need to model its state of motion as described in Section 2.1 and its dynamics with respect to available control inputs in Section 2.2. We use the dual-robot setup depicted in Figure 2.1 to explain the concept of a configuration space and its dynamics.

Motions of robots are constrained by physics, safety requirements, and by the task they should fulfill. In Section 2.3 we introduce different sources of such constraints that arise in manipulation. This lays out the basics for motion planning and control. In Section 2.4 we provide an overview of sampling-based motion planning. This planning paradigm has shown good empirical performance and theoretical guarantees for practically

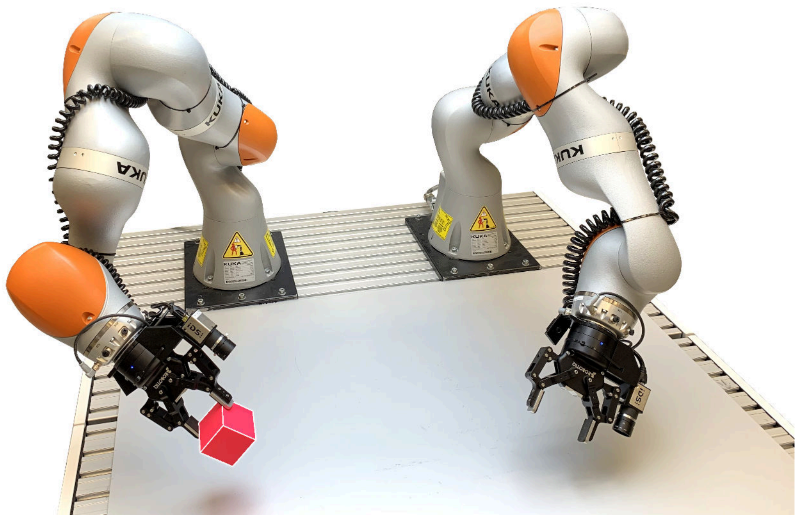


Figure 2.1: Exemplary setup of a robot and a manipulated object: The setup consists of two robot arms, two parallel grippers, and one manipulated object, a cube. Both robot arms are redundant, seven-axis manipulators.

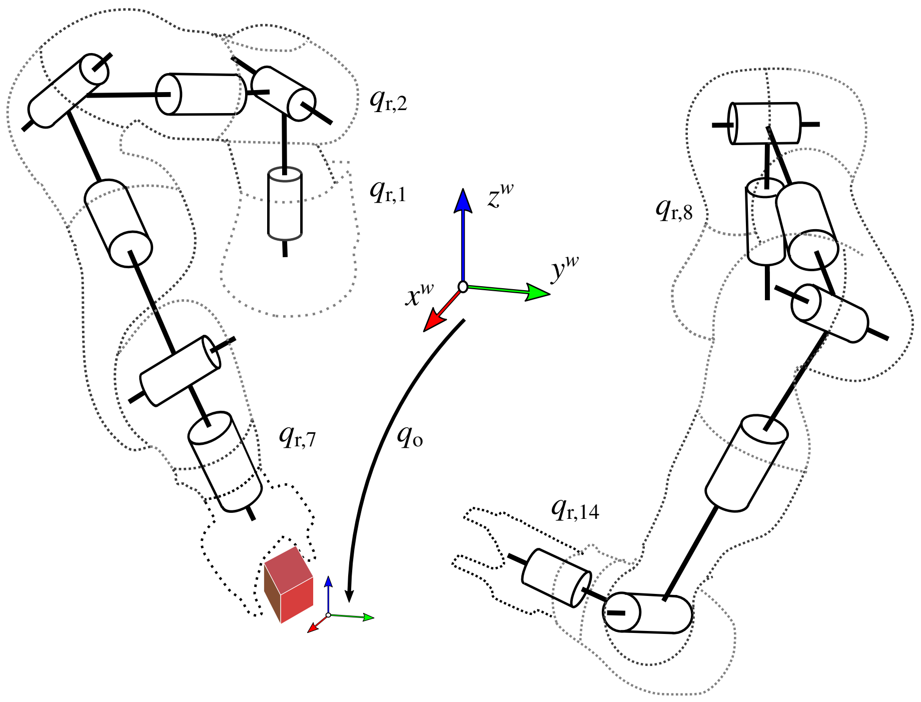


Figure 2.2: Configuration space for the dual-robot setup of Figure 2.1: The configuration of the robot $\mathbf{q}_r$ comprises 14 axis positions $q_{r,1}$ to $q_{r,14}$. The pose of the object is encoded in $\mathbf{q}_o$. A suitable representation for this pose is a vector-quaternion representation. The full configuration $\mathbf{q} = \left[\mathbf{q}_r^\top, \mathbf{q}_o^\top\right]^\top$ encodes the poses of all rigid bodies in this setup. Image taken from [2] and modified.

relevant motion planning problems. In Section 2.5 we give an overview over techniques for sampling-based motion planning with equality constraints. Finally, we introduce the concept of manipulation planning in Section 2.6.

2.1 Configurations and Kinematics

To plan or control robot motions it is necessary to describe the state of motion of a system. The models we introduce in this section are based on the work of LaValle [1]. Robots are mechanisms that are built from multiple components that we assume to be rigid bodies. Each of these rigid bodies has a position and orientation in the three-dimensional space it inhabits and thus six degrees of freedom (DoF). In principle one could describe the state of motion of a system via the six-DoF states of all of its components. However, this is not a practical approach due to the large number of resulting variables and constraints between them. For this reason we encode the state of motion of a system via n variables as the configuration $\mathbf{q} \in \mathbb{R}^n$ as shown in Figure 2.2. This configuration $\mathbf{q} = \left[\mathbf{q}_r^\top, \mathbf{q}_o^\top\right]^\top$ is composed of the configuration of one or multiple robots $\mathbf{q}_r$ and one or multiple objects $\mathbf{q}_o$. Throughout this thesis we refer to quantities related to robots or objects with subscripts $(\cdot)_r$

and $(\cdot)_\mathrm{o}$ respectively.

In the example of Figure 2.2 the pose of each robot arm can be efficiently encoded with seven axis positions. The resulting 14 values form the robot configuration $\mathbf{q}_\mathrm{r} \in \mathbb{R}^{n_\mathrm{r}}$, with n_r being the number of axes. All positions and orientations of the components of the robot can then be computed via forward kinematics given a robot configuration $\mathbf{q}_\mathrm{r}$.

The vector $\mathbf{q}_\mathrm{o}$ encodes the poses of all moving (non-robot) objects of a system. In the example, this encodes the position and orientation of the cube relative to a reference coordinate system. We chose a vector plus unit quaternion representation for the poses of objects. In the example this results in $n_\mathrm{o} = 7$ variables.

We denote time as t. In some scenarios, some components of a setup may have time-dependent positions. Examples are conveyor belts or machines that operate periodically, e. g., forge hammers.

Figure 2.2 illustrates, that few variables, $\mathbf{q}$ and t, can encode the positions of a mechanism, but also that this encoding is in general not unique or minimal. The order in which axis positions occur in $\mathbf{q}_\mathrm{r}$ is arbitrary as well as the choice of the reference coordinate system for $\mathbf{q}_\mathrm{o}$. Using a unit quaternion for orientations in $\mathbf{q}_\mathrm{o}$ introduces one redundant variable per object for computational reasons.

Given a system configuration $\mathbf{q}$ and time t we can define the velocity $\dot{\mathbf{q}}$ and acceleration $\ddot{\mathbf{q}}$ of configurations as first and second-order time derivatives. For computational reasons it is common practice to represent the orientation of objects with four-dimensional unit quaternions. The rotational velocities (and deeper derivatives) are typically represented via three-dimensional angular velocity vectors. This means that the vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$ have different dimensionalities during computations. We omit these technical details in the remainder of this thesis and represent $\dot{\mathbf{q}}$ as if it had dimensionality n.

2.2 Dynamics

In this thesis we address manipulation both from a geometric perspective for planning as well as from a dynamic perspective for planning and control. For the latter it is necessary to model the dynamics of the configuration of a system. We consider prehensile manipulation only, which covers a wide variety of manipulation tasks, especially in an industrial context. This means that at all times objects are either rigidly grasped by a robot or at rest in the environment.

Prehensile manipulation without sliding contact allows to model dynamics with a second-order and control affine differential equation:

$$\ddot{\mathbf{q}} = \mathbf{a}(\mathbf{q}, \dot{\mathbf{q}}, t) + B(\mathbf{q}, t)\mathbf{u}. \tag{2.1}$$

Here, $\mathbf{u} \in \mathbb{R}^{n_u}$ is the n_u dimensional control input. The vector-valued function $\mathbf{a}$ determines the acceleration of the system that is independent from the control input $\mathbf{u}$. The $n \times n_u$ matrix B models the affine dependency between the acceleration $\ddot{\mathbf{q}}$ and the control input $\mathbf{u}$.

Let us consider the example of Figure 2.2. In this example the object is rigidly grasped by a robot. For this reason, the velocity of the object $\dot{\mathbf{q}}_\mathrm{o}$ can be modeled by the $n_\mathrm{o} \times n_\mathrm{r}$ Jacobian matrix J as $\dot{\mathbf{q}}_\mathrm{o} = J\dot{\mathbf{q}}_\mathrm{r}$. The Jacobian J is a function of $\mathbf{q}_\mathrm{r}$, $\mathbf{q}_\mathrm{o}$, and t. If the object is placed J is a zero matrix. We assume that the robots are equipped with a controller that allows to directly command the robot accelerations $\ddot{\mathbf{q}}_\mathrm{r} = \mathbf{u}$. In the example, this results in the following equivalent for Eq. (2.1):

$$\ddot{\mathbf{q}} = \begin{bmatrix} \ddot{\mathbf{q}}_\mathrm{r} \\ \ddot{\mathbf{q}}_\mathrm{o} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{0}_{n_\mathrm{r}} \\ \dot{J}\dot{\mathbf{q}}_\mathrm{r} \end{bmatrix}}_{\mathbf{a}(\mathbf{q},\dot{\mathbf{q}},t)} + \underbrace{\begin{bmatrix} I_{n_\mathrm{r}} \\ J \end{bmatrix}}_{B(\mathbf{q},t)} \mathbf{u},$$

where I_{n_r} denotes an $n_\mathrm{r} \times n_\mathrm{r}$ identity matrix. In the example the dynamics are not dependent on the time t. When the environment comprises time-variant components, such as conveyor belts, the dynamics may be time-variant as well.

Naturally, the controls $\mathbf{u}$ must be bounded to prevent damage to the system. We assume that the bounds on controls are given as a symmetric and affine inequality:

$$-\mathbf{k}_\mathrm{max} \leq \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}, t) + D(\mathbf{q}, t)\mathbf{u} \leq \mathbf{k}_\mathrm{max}. \tag{2.2}$$

Again, $\mathbf{c}$ denotes a control-independent term and D an affine dependency on $\mathbf{u}$. The sum of these terms must fulfill the box constraint defined by $\mathbf{k}_\mathrm{max}$.

Let us again consider the example of Figure 2.2, where the control input is equal to the acceleration of the robot $\ddot{\mathbf{q}}_\mathrm{r} = \mathbf{u}$. If we simply assume that the accelerations of the robot axes are bounded by conservatively chosen maximal values $\ddot{\mathbf{q}}_{r,\mathrm{max}}$, Eq. (2.2) is equivalent to the trivial box constraint:

$$-\mathbf{k}_\mathrm{max} = -\ddot{\mathbf{q}}_{r,\mathrm{max}} \leq \underbrace{\mathbf{0}_{n_\mathrm{r}}}_{\mathbf{c}(\mathbf{q},\dot{\mathbf{q}},t)} + \underbrace{I_{n_\mathrm{r}}}_{D(\mathbf{q},t)} \mathbf{u} \leq \ddot{\mathbf{q}}_{r,\mathrm{max}} = \mathbf{k}_\mathrm{max}.$$

In case we want to accurately model the influence of the robot's motor torques τ_m we need to consider the dynamics of rigid bodies:

$$M(\mathbf{q}, t)\ddot{\mathbf{q}}_\mathrm{r} + \tau_c(\mathbf{q}, \dot{\mathbf{q}}, t) + \tau_g(\mathbf{q}, t) = \tau_m.$$

Here, M is the inertia matrix and τ_c and τ_g model Coriolis forces and gravity. As we assume prehensile manipulation, no contact forces need to be modeled. If we assume that the motor torques τ_m are bounded by τ_max this results in the following equivalent of Eq. (2.2):

$$-\mathbf{k}_\mathrm{max} = -\tau_\mathrm{max} \leq \underbrace{\tau_c(\mathbf{q}, \dot{\mathbf{q}}, t) + \tau_g(\mathbf{q}, t)}_{\mathbf{c}(\mathbf{q},\dot{\mathbf{q}},t)} + \underbrace{M(\mathbf{q}, t)}_{D(\mathbf{q},t)} \mathbf{u} \leq \tau_\mathrm{max} = \mathbf{k}_\mathrm{max}.$$

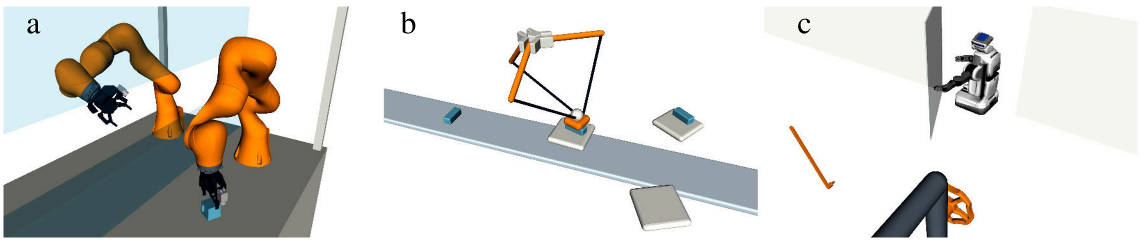


Figure 2.3: Different systems and tasks with different constraints on configurations: (a) Two fixed manipulators transport a cube. This is a standard instance of manipulation planning and only requires to model limits on axis positions, collisions, grasp poses, and placements. (b) A delta robot places objects on a conveyor belt. This example includes closed kinematic chains. Also, some constraints, e. g., collision-avoidance, are time-variant due to the moving conveyor belt. (c) A mobile manipulator operates a door. In this scenario closed kinematic chains appear or disappear depending on the current contact state.

2.3 Constraints on Configurations

Robotic motion in general and manipulation in particular must fulfill constraints that arise due to physics and due to the task of the robot. An example for a physical constraint is that objects do not move unless they are actively manipulated. Tasks may constrain motions as well, e. g., a glass of water should be held upright so that its contents are not spilled. The origin of such constraints may be ambiguous, e. g., a robot cannot move its gripper through a wall and we typically wish for it not to attempt such a motion. A variety of constraints that arise in manipulation is shown in Figure 2.3.

We denote the constraints on configurations via equality constraints $\mathbf{f}$ and inequality constraints $\mathbf{g}$:

$$\mathbf{f}(\mathbf{q}, t) = \mathbf{0}, \qquad \mathbf{g}(\mathbf{q}, t) \leq \mathbf{0}. \tag{2.3}$$

The functions $\mathbf{f}$ and $\mathbf{g}$ are vector-valued and each vector element describes a separate constraint that configurations must fulfill. Examples for equality constraints arise for example in closed kinematic chains as in example (b) of Figure 2.3. Typical examples for inequality constraints are axis limits. Technically, it is not necessary to model equality constraints as they can be represented as two inequalities with different signs. We deliberately add the distinction between equality and inequality constraints as it is critically important for sampling-based planners as discussed in the next two sections. In general the constraints may be time-variant, e. g., when the robot's environment contains a conveyor belt.

Motions may also be constrained at the velocity level via the functions $\mathbf{v}$ and $\mathbf{w}$:

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}(\mathbf{q}, t) = \mathbf{0}, \qquad \tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{w}(\mathbf{q}, t) \leq \mathbf{0}. \tag{2.4}$$

When a robot transports an object in its gripper, the relative velocity between object and gripper must be zero. Maximal velocities of robot axes are velocity inequality constraints.

At this point we make no assumptions on the properties of the functions $\mathbf{f}$, $\mathbf{g}$, $\mathbf{v}$, and $\mathbf{w}$, except that $\mathbf{v}$ and $\mathbf{w}$ must be at least piecewise differentiable with respect to t.

The following list shows examples of typical sources of constraints in manipulation.

- **Axis-limits and maximal velocities**: Due to physical limitations or due to safety aspects the positions and velocities of robot axes must typically be limited. This results in inequality constraints.

- **Collision avoidance**: One of the most important constraints is collision avoidance. Collisions could occur between different components of the robot, the robot and its environment, or manipulated objects and the robot. One way to formulate these constraints is as inequality constraints on distances between geometric bodies. Another way would be via an indicator function that returns the result of a collision check.

- **Contact state** The contact state of manipulation constrains the relative motion of robots and manipulated objects. If an object is placed it does not move at all. When it is grasped, the relative velocity between gripper and object is zero.

- **Closed kinematic chains**: Some robots, e. g., the delta-robot in example (b) of Figure 2.3 form closed kinematic chains. In some scenarios a task may produce a closed kinematic chain as well. Example (c) of Figure 2.3 shows such a scenario where a robot opens a door. These closed chains result in equality constraints.

- **Task specific constraints**: Many scenarios include constraints that are specific to the task at hand. The task of example (c) of Figure 2.3 requires that the robot points its head mounted camera at the object that it currently manipulates with its grippers.

With the constraints in place we can define motion planning problems. Starting from an initial configuration $\mathbf{q}_{\text{start}}$ we aim to reach a set of goal configurations $\mathcal{Q}_{\text{goal}} \subset \mathbb{R}^n$. We need to compute a trajectory of configurations $\tau : [t_{\text{start}}, t_{\text{end}}] \to \mathbb{R}^n$ and a trajectory of controls $\phi : [t_{\text{start}}, t_{\text{end}}] \to \mathbb{R}^{n_u}$ between the current time t_{start} and terminal time t_{end} with $t_{\text{end}} \geq t_{\text{start}}$. The trajectory of configurations τ must start in the current configuration $\tau(t_{\text{start}}) = \mathbf{q}_{\text{start}}$ and end in the set of goal configurations $\tau(t_{\text{end}}) \in \mathcal{Q}_{\text{goal}}$. At all times the dynamics of Eq. (2.1), the limits on control inputs of Eq. (2.2), and the constraints of Eq. (2.3) and Eq. (2.4) must be fulfilled.

2.4 Sampling-Based Motion Planning

Motion planning in the context of manipulation poses two substantial challenges: it involves high-dimensional configuration spaces and complex, non-linear collision constraints. Most robotic manipulators have at least six degrees of freedom. The resulting high-dimensional configuration space cannot be discretized meaningfully to apply classical search algorithms such as A*. Furthermore, in the context of collision-free motion planning, there exists no explicit representation of which areas of the configuration space are valid, i. e., at least collision free. For complex and potentially non-convex geometries it is only possible to check whether a given configuration or trajectory segment is valid. This collision checking is an operation that uses a relatively large amount of processing time.

Sampling-based motion planning bypasses both of these challenges. Instead of maintaining an explicit representation of the valid configurations, planners only check randomly sampled configurations and trajectory segments for collisions. This random sampling leverages the so called "visibility" properties of motion planning for manipulation [3]. While manipulation involves high-dimensional configuration spaces, it typically does not involve labyrinth-like geometries. Most planning problems can be solved with few point to point motions to avoid obstacles. Manipulation typically occurs in environments that are organized to allow efficient motions. Examples are industrial workplaces or kitchens.

In the following we explain three sampling-based planners and discuss their properties. These algorithms are the Probabilistic Roadmap (PRM) [4], the Rapidly exploring Random Tree (RRT) [5], and the Expansive Space Tree (EST) [6]. The reason for explaining these three algorithms is that we use variants of these approaches in this thesis. Furthermore, it is worthwhile to understand the differences in the assumptions that are made on planning domains. In their basic forms PRM and RRT are suitable for geometric planning only. For this reason we omit control inputs when describing these two algorithms.

```
Procedure: PRM( $\mathbf{q}_\text{start}$ ) infinite version
1  $\mathcal{N} \leftarrow \{\mathbf{q}_\text{start}\}$,  $\mathcal{E} \leftarrow \{\}$
2  while true do
3      $\mathbf{q}_\text{new} \leftarrow$ sampleRandom( )
4      if isValid( $\mathbf{q}_\text{new}$ ) then
5          $\mathcal{N}$.add( $\mathbf{q}_\text{new}$ )
6          for $\mathbf{q}_\text{near} \in$ kNearest( $\mathcal{N}, \mathbf{q}_\text{new}$ ) do
7              $\tau \leftarrow$ localPlanner( $\mathbf{q}_\text{near}, \mathbf{q}_\text{new}$ )
8              if pathValid( $\tau$ ) then
9                  $\mathcal{E}$.add( $(\mathbf{q}_\text{near}, \tau, \mathbf{q}_\text{new})$ )
10                 $\mathcal{E}$.add( $(\mathbf{q}_\text{new}, \tau_\text{reversed}, \mathbf{q}_\text{near})$ )
```

Kavraki *et al.* [4] introduce one of the first sampling-based motion planners, the Probabilistic Roadmap (PRM). Like all three discussed planners, PRM proceeds to build a graph of nodes $\mathcal{N}$ and edges $\mathcal{E}$. An infinite version of this graph construction without termination condition is shown in procedure **PRM**. In its main loop a random configuration $\mathbf{q}_\text{new}$ is sampled via the procedure **sampleRandom**. The procedure **isValid** checks whether this configuration fulfills the constraints of the planning problem, typically by performing a

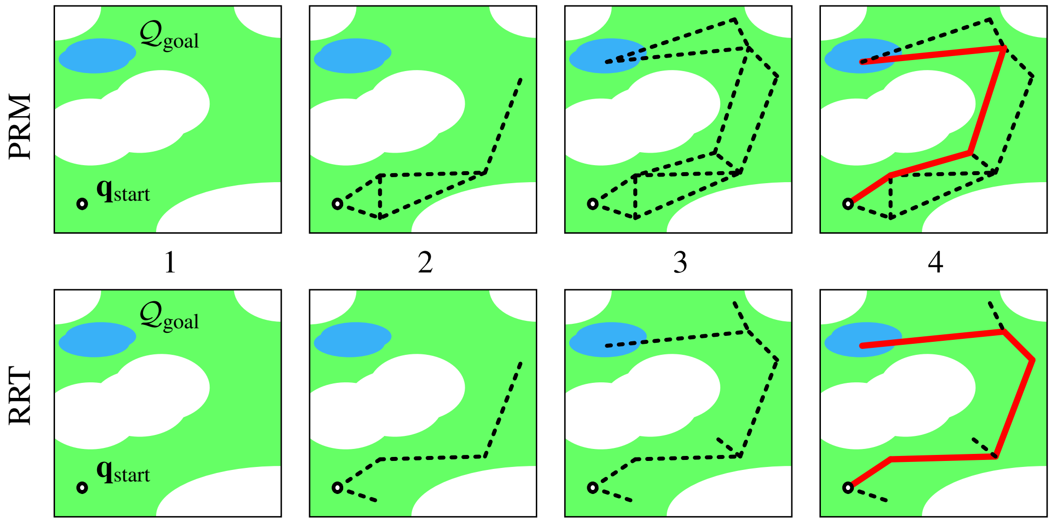

PRM
RRT
$\mathcal{Q}_{\text{goal}}$
$\mathbf{q}_{\text{start}}$
1
2
3
4
$\mathcal{Q}_{\text{goal}}$
$\mathbf{q}_{\text{start}}$

Figure 2.4: Motion planning with PRM and RRT: The upper half of the image shows the construction of a probabilistic roadmap. In the lower half the construction of a rapidly exploring random tree can be seen. (1) In both cases the goal is to plan a motion from a configuration $\mathbf{q}_{\text{start}}$ to a goal set $\mathcal{Q}_{\text{goal}}$ (marked blue). The green areas show valid configurations, i. e., configurations that are at least collision-free. (2), (3) The planners build a graph structure to search through the configuration space. For RRT this is a tree. (4) Once a solution is found (marked red) the search terminates.

collision check. If this succeeds, the configuration $\mathbf{q}_{\text{new}}$ is added to the set of nodes $\mathcal{N}$. The procedure **kNearest** returns (at most) k configurations within $\mathcal{N}$ that are closest to $\mathbf{q}_{\text{new}}$ with respect to some distance measure. For each of these configurations $\mathbf{q}_{\text{near}}$ the procedure **localPlanner** produces a trajectory τ from $\mathbf{q}_{\text{near}}$ to $\mathbf{q}_{\text{new}}$. The procedure **pathValid** checks whether all configurations along τ are valid. If this succeeds an edge from $\mathbf{q}_{\text{near}}$ to $\mathbf{q}_{\text{new}}$ is added to $\mathcal{E}$. As PRM deals with geometric planning, the trajectory τ is reversible and a reversed edge is added as well (τ_{reversed}). Naturally, in a practical implementation this is not repeated forever and the search terminates when a path to a goal is found. Figure 2.4 shows the graph construction of PRM.

One advantage of PRM is that the constructed graph is re-usable across multiple planning queries as long as the environment of the robot remains constant. When the environment changes between queries this advantage is lost and PRM performs unnecessary computations. The Rapidly exploring Random Tree (RRT) by Lavalle [5] is designed to be more efficient in single-query scenarios. Instead of building a cyclic graph, RRT builds a tree rooted in $\mathbf{q}_{\text{start}}$. This is shown in procedure **RRT** (as infinite tree construction) and in Figure 2.4. The main loop of the planner is similar to that of PRM with two changes. Instead of attempting a connection to the k nearest nodes in the graph, only the closest node is considered. Also, new nodes are only added if they can be connected to the currently built tree. This prevents the planner from exploring parts of the configuration space that are disconnected from the initial configuration.

Procedure: RRT($\mathbf{q}_{\text{start}}$ **)** *infinite version*

```
1 $\mathcal{N} \leftarrow \{\mathbf{q}_{\text{start}}\}, \quad \mathcal{E} \leftarrow \{\}$
2 while true do
3     $\mathbf{q}_{\text{new}} \leftarrow$ sampleRandom( )
4     if isValid( $\mathbf{q}_{\text{new}}$ ) then
5         $\mathbf{q}_{\text{near}} \leftarrow$ nearest( $\mathcal{N}, \mathbf{q}_{\text{new}}$ )
6         $\tau \leftarrow$ localPlanner( $\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}$ )
7         if pathValid( $\tau$ ) then
8             $\mathcal{N}$.add( $\mathbf{q}_{\text{new}}$ )
9             $\mathcal{E}$.add( $(\mathbf{q}_{\text{near}}, \tau, \mathbf{q}_{\text{new}})$ )
```

Both PRM and RRT rely on the procedure **localPlanner** to connect two configurations. This is problematic when dynamics are considered and especially difficult if the planning problem is time-variant. While there exist variants of RRT that address this, a more generic and elegant solution is to plan without the procedure **localPlanner**. This is done with the Expansive Space Tree (EST), introduced by Hsu *et al.* [6]. Like the RRT, the EST proceeds to build a search tree in configuration space. However, the tree expansion of the EST follows an inverted logic relative to RRT. With RRT and PRM a random node within the configuration space is sampled and then connected to the currently built graph structure. In contrast, EST samples an existing node of the currently built tree and expands it to a new configuration. This is shown in procedure **EST**.

Procedure: EST($\mathbf{q}_{\text{start}}$ **)** *infinite version*

```
1 $\mathcal{N} \leftarrow \{s_{\text{start}}\}, \quad \mathcal{E} \leftarrow \{\}$
2 while true do
3     $\mathbf{q}_{\text{old}} \leftarrow$ sampleWeighted( $\mathcal{N}$ )
4     $\phi \leftarrow$ randomControls( )
5     $\mathbf{q}_{\text{new}}, \tau \leftarrow$ simulateControls( $\mathbf{q}_{\text{old}}, \phi$ )
6     if pathValid( $\tau$ ) then
7         $\mathcal{N}$.add( $\mathbf{q}_{\text{new}}$ )
8         $\mathcal{E}$.add( $(\mathbf{q}_{\text{old}}, (\tau, \phi), \mathbf{q}_{\text{new}})$ )
```

In its main loop the EST algorithm selects a random node from $\mathcal{N}$ using the procedure **sampleWeighted**. This weighted sampling puts larger weights on areas of the configuration space that are sparsely populated by the currently built search tree. A typical implementation is to coarsely discretize the configuration space into a grid. Then nodes can be selected with a probability that is inversely proportional to the number of nodes in the same grid cell. After selecting a node, a random control trajectory ϕ is sampled via **sampleControls**. By integrating the system dynamics via **integrateDynamics** a new

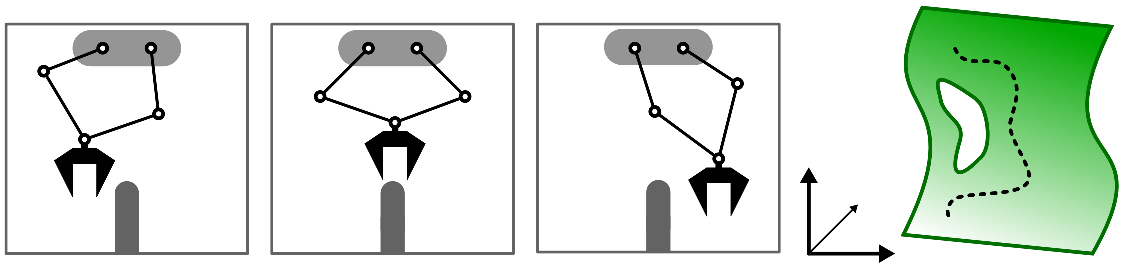


Figure 2.5: Constrained motion with a delta robot: On the left a delta robot is shown that moves around an obstacle. The robot forms a closed kinematic chain which imposes equality constraints on its axis positions. The right image shows a schematic visualization of the resulting constraint manifold. In the visualization this is a two dimensional surface within an three dimensional embedding configuration space.

configuration $\mathbf{q}_{\text{new}}$ is obtained. If the path to this configuration is valid a new node and edge is added to the graph.

2.5 Sampling-Based Planning with Equality Constraints

The typical constraints that are considered in sampling-based motion planning are axis limits and collision avoidance. Both can be represented as inequality constraints on configurations. Many applications, especially in manipulation, require to consider equality constraints as well. An example is a delta robot with closed kinematic chains as shown in Figure 2.5.

In their basic forms, the planners of the previous section are unable to solve problems with equality constraints. The reason is that equalities force solutions of the planning problem onto lower-dimensional manifolds in the configuration space. This is also shown in Figure 2.5. As these manifolds have zero relative volume with respect to the embedding configuration space, there is typically zero probability of randomly sampling configurations or trajectories on these manifolds.

To address this, a variety of modifications to standard sampling-based planners have been proposed. Kingston *et al.* [7] provide a survey of the proposed approaches and introduce a classification of the underlying mechanisms to resolve equality constraints. The following list shows the five categories of approaches that were identified in [7]:

1. **Relaxation of the constraints**: A simple approach is to relax all equality constraints within a tolerance and solve the relaxed planning problem with standard sampling-based planners. This approach has two major drawbacks. Even small violations of the constraints may have severe consequences on a real robot. In the example of the delta robot of Figure 2.5, small errors in the closure of the kinematic chain result in large internal forces in the mechanism. One motivation to use sampling-

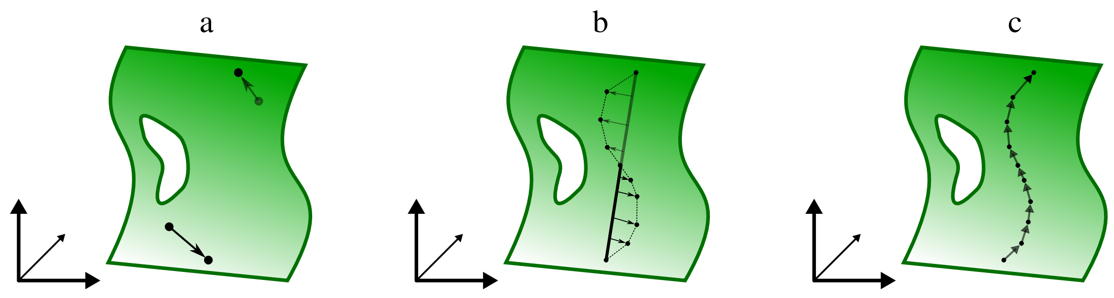


Figure 2.6: Techniques for sampling based planning with equality constraints: (a) Projected sampling: Configurations are sampled without considering the equality constraints. In a second step the resulting configurations are projected onto the constraint manifold. (b) Projected local planning: Two configurations that lie on the constraint manifold need to be connected. A local planner is used that does not adhere to the constraints and the resulting trajectory is then projected onto the manifold. (c) Local planning within a tangent space: To connect two configurations a series of motions in the local tangent of the constraint manifold is computed. Images adapted from [7].

based planners is to exploit the good visibility properties of many motion planning problems. Unless very large tolerances are used these properties are not given anymore and the performance of sampling-based planners degrades.

2. **Projection onto the constraint manifolds**: The idea of projection methods is to use the standard operations of sampling-based planners as if no equality constraint was present and then project the result onto the constraint manifolds. This can be done both for sampling constrained configurations and for constrained local planning as shown in Figure 2.6.

3. **Motion in a tangent space**: The idea of this method is to locally linearize the equality constraints. This results in a tangent plane to the constraint manifold. As shown in Figure 2.6 one can now repeatedly move in these planes for local planning.

4. **Atlas**: The idea of an atlas is to approximate the constraint manifold with a finite set of plane segments.

5. **Re-parameterization**: With this method the planning problem is reformulated in a different configuration space that does not involve equality constraints. An example would be to plan in the tool coordinates of the delta robot in Figure 2.5. The tool coordinates are not constrained by equalities and the axis positions can later be retrieved via inverse kinematics. This approach has two major advantages. The equality constraints are exactly fulfilled. The other four methods only fulfill the constraints within a numerical threshold. Also, the full spectrum of sampling-based planners may be used including optimal planners. The downside of this

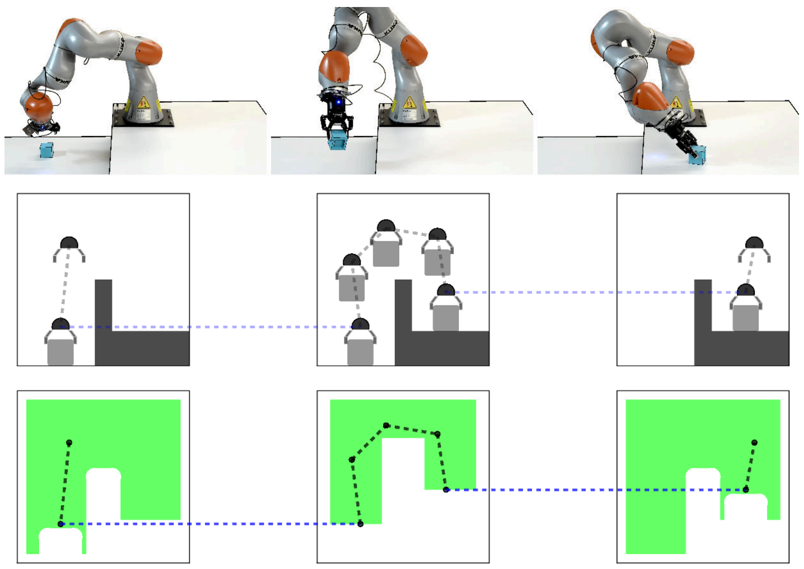


Figure 2.7: A sequential manipulation task: The upper row shows a robot that picks a cube from one surface and places it on another. In the middle row a schematic 2D representation of this task is shown. The sequence of motions comprises three contact states. In the lower row the collision free configurations of the robot are shown in green. As can be seen, the set of collision free configurations changes when the robot grasps or releases the object.

approach is that a suitable re-parameterization needs to be found. Ideally, such a re-parameterization should be able to generalize across a set of planning problems.

The algorithms that are proposed in this thesis are closely related to the tangent space approach or the re-parameterization method.

2.6 Manipulation Planning

The previous sections introduced methods to plan a single robotic motion. For manipulation this is not sufficient. Figure 2.7 shows a pick and place task in which a cube is transported by a robot. This task requires a sequence of motions in which the robot makes or brakes contact with the transported object. A naive approach to this manipulation problem would be to treat grasping and placing the object separately. Given the initial pose of the object one could compute a grasp pose, i. e., a relative transform between object and gripper. Then one would solve the inverse kinematics problem of finding a robot configuration that positions the gripper in this pose. Finally, one would plan a motion towards this configuration. After the object has been grasped, one would compute a placement pose and a corresponding robot configuration and again plan a motion towards this configuration.

This naive approach is however likely to fail. The reason for this is that the sequential robot motions and interactions with the object are interdependent. If an object is grasped

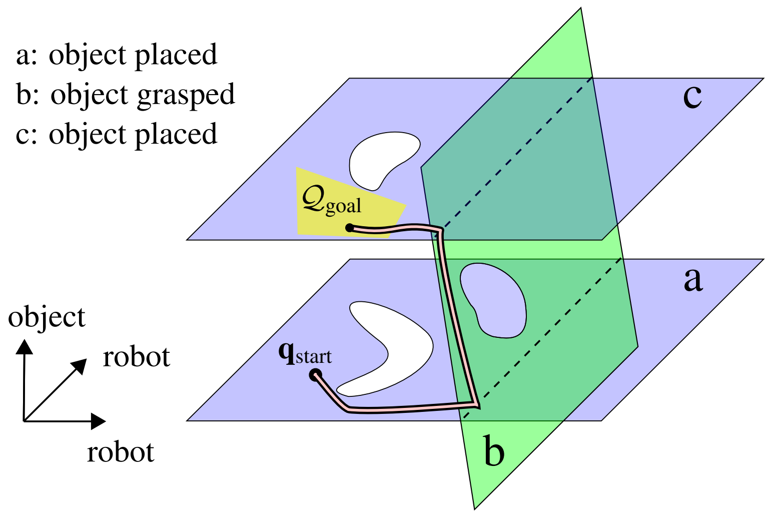


Figure 2.8: Model for a pick and place task: Three planes visualize the constraint manifolds for different contact states. Initially the object is placed (a) and the robot is not holding it. Thus, the object does not move, irrespective of the robot's motion. At the intersection of the manifolds for placement (a) and grasp (b) the object can be grasped. When grasped (b) the object moves along with the robot. Finally, the object is released in a different placement pose (c). The thick line shows a trajectory from a start configuration $\mathbf{q}_{\text{start}}$ to the goal region $\mathcal{Q}_{\text{goal}}$.

from a wrong direction it may not be possible to place it again without a collision of the gripper and the surface the object is placed upon. For this reason it is necessary to plan the motions of the robot and the object simultaneously. This leads to the problem of manipulation planning [8].

Manipulation can be modeled as an instance of constrained motion with a special structure. Let us recall that an n-dimensional configuration is composed of an n_r-dimensional robot configuration and an n_o-dimensional object configuration. For prehensile manipulation, the system is assumed to be in one of two conceptual states. The object may either be resting in the environment or be rigidly grasped by the robot. When the object is placed it does not move. When the object is grasped it moves along with a gripper. This means that manipulation occurs in an n-dimensional embedding configuration space, but all motions occur on manifolds that are at most n_r-dimensional. Figure 2.8 shows these manifolds for a pick and place task.

While manipulation planning can formally be modeled as constrained motion it is worth to view it as a separate class of planning problems for two reasons. The first reason is what has been named the crossed foliation issue [9]. This issue refers to the fact that almost all pairs of valid configurations lie on different constraint manifolds. Let us consider the example in Figure 2.8. The two parallel planes drawn in blue refer to configurations in which the object is placed. It is not possible to move the system from one of these manifolds to the other without at least two changes of contact state. To move an object it must be grasped and released again. While Figure 2.8 shows only three manifolds, real world manipulation problems typically have an infinite number of such manifolds. This is due to continuous sets of possible ways objects can be grasped or placed. A second reason to consider manipulation as a separate class of planning problem is that its multi-modal structure can be actively leveraged for planning and control.

Throughout this thesis we will use the following notation to describe the special, constrained structure of manipulation. At all times, the system is in a discrete mode or contact state σ with $\sigma \in \Sigma$. Here, Σ denotes the finite set of possible modes. In the example of Figure 2.7 there is a total of $|\Sigma| = 18$ modes that describe the contact state of the cube. The cube could either be resting on the left surface, be grasped by the gripper, or be resting on the right surface. For each of these cases one of the six sides of the cube may be in contact with a surface or the right finger of the gripper. This defines 18 modes that constrain the motion of robot and object. The definition of modes for a manipulation problem is not unique.

A mode σ typically under-constrains the system configuration. An example is a mode for the placement of the cube. In this case the cube may be in any pose on the surface in which the contacting side is parallel to the surface and at the correct height. This leaves three under-constrained degrees of freedom: translation along the x- and y-axis and rotation around the z-axis.

For the purpose of manipulation planning, it is helpful to make the constraints on the relative motion of robot and object explicit. For this purpose we introduce the contact parameterization π. This parameterization denotes both the mode and the relative transform of an object to the part of the robot or environment it is attached to. An example would be a contact state in which the object is grasped along with a relative transform between gripper and object. Generally, the contact parameterization π is an element of a continuous set of possible contact parameterizations $\pi \in \Pi$. Given a robot configuration $\mathbf{q}_\mathrm{r}$ and a contact parameterization π the object configuration $\mathbf{q}_\mathrm{o}$ can be computed via forward kinematics $\mathbf{q}_\mathrm{o} = \mathrm{fk}(\mathbf{q}_\mathrm{r}, \pi)$. With slight abuse of notation we treat a pair $(\mathbf{q}_\mathrm{r}, \pi)$ as a configuration $\mathbf{q}$.

Chapter 3

Optimal, Sampling-Based Manipulation Planning

In the previous section we introduced the problem of manipulation planning. Solving this problem is a challenging task. As with motion planning, manipulation requires reasoning about high-dimensional configuration spaces. Additionally, it involves discrete variables that indicate whether or not an object is currently grasped by the robot. To be useful in practice, a manipulation planner must also address the following challenges:

- **Continuous sets of grasps and placements**: For typical manipulation problems there exist continuous sets of grasps and placements for each object. Consider the example depicted in Figure 3.1: A seven-axis robot manipulates a cube. The cube may be placed in any position or orientation on one of two surfaces as long as one of its sides is in contact with the surface. These sets of grasps and placements are too large to be meaningfully discretized and manually selecting suitable grasps and placements defeats the point of planning. Another difficulty arises due to the redundant degree of freedom of the robot. For most pairs of grasps and placements there exists a continuous set of robot configurations that allow a transition from grasp to placement.

- **Incomplete sampling-based motion planning**: Motions for manipulation occur in high-dimensional configuration spaces. Furthermore, it is time-consuming to decide which parts of these configuration spaces are valid, i. e., at least collision free. Sampling-based planning addresses these issues and has shown good empirical performance for many relevant motion planning problems. However, sampling-based planners such as probabilistic roadmaps [4] are only probabilistically complete. This means that these algorithms cannot decide whether a planning query is infeasible.

Most of the results of this chapter have been previously published by the author in conference proceedings. For improved readability we omit citations of this previous publication throughout this thesis. Instead, the relation between this chapter and the previous publication is discussed in detail in Section 3.7.

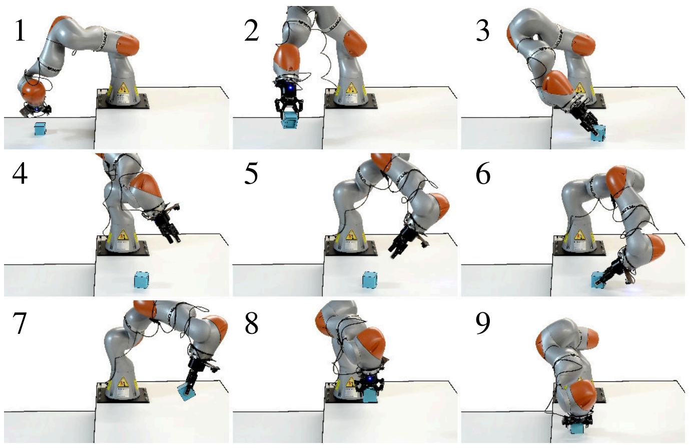


Figure 3.1: Planned manipulation sequence: A robot with seven axes and a parallel gripper must manipulate a cube. The target is to place the cube on the right table and place it upside down. To do so, the robot must (re-)grasp the object at least twice. The shown sequence of motions was computed by the planner we propose in this chapter and executed on a real robot.

For manipulation planning this is problematic, as motion planners are often used as sub-algorithms.

- **Optimality**: In most applications, the efficiency of robotic motions is important. Industrial manipulation requires low cycle times, as they directly contribute to the financial impact of automation. Robots that operate in households must move swiftly and predictably in order to be accepted by humans. These issues can be addressed by planning optimal motions.

Previous works have solved some of these challenges. Continuous grasps and placements have been addressed by Simeon *et al.* [10]. The incompleteness of sampling-based motion planners is solved with the PTR planner by Hauser [11]. For special cases of manipulation, optimal planners have been proposed, e. g., by Han *et al.* [12]. Despite the importance of robotic manipulation to the best of our knowledge no previous approach addresses all three challenges.

The contribution of this chapter is an asymptotically optimal manipulation planner. We extend an optimal, sampling-based motion planner to efficiently explore the combined configuration space of robot and object. Under a novel set of robustness conditions, we prove global, asymptotic optimality and probabilistic completeness. These result are validated in extensive simulations and on a real robot. The simulated experiments show a significant reduction in cost relative to a state-of-the-art planner. We further propose two methods to re-use and delay computations during planning. Our planner computes high quality solutions to realistic manipulation problems in less than one second.

This chapter is structured as follows: We introduce a formal problem definition in Section 3.1. For this planning problem we propose a novel manipulation planner in Sec-

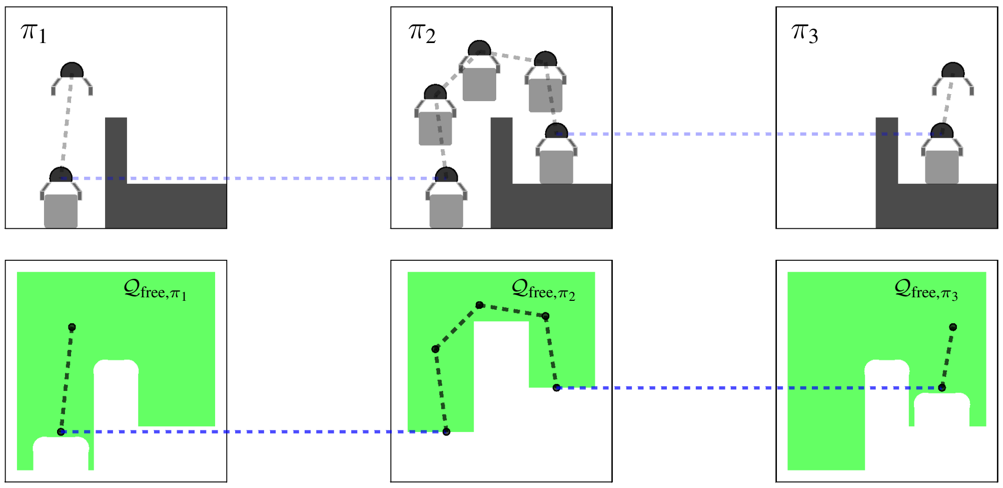

π_1
π_2
π_3
$\mathcal{Q}_{\text{free},\pi_1}$
$\mathcal{Q}_{\text{free},\pi_2}$
$\mathcal{Q}_{\text{free},\pi_3}$

Figure 3.2: Robot configuration and contact parameter: The upper row of images shows a pick and place sequence with three different contact parameters π_1 to π_3. In π_1 and π_3 the object is placed and does not move when the robot moves. In π_2 the object is grasped and moves along with the robot. The lower row of images shows the corresponding collision free configurations for the robot $\mathcal{Q}_{\text{free},\pi_1}$ to $\mathcal{Q}_{\text{free},\pi_3}$ as green areas. Note how these areas change when the object is grasped or released as the object effectively becomes part of the robot or the environment.

tion 3.2.1. To reduce planning times we introduce two generic strategies to re-use and to delay computations in Section 3.2.2. In Section 3.3 we analyze the computational complexity of the algorithm and prove asymptotic optimality and probabilistic completeness. We implemented our approach on a real robot and validated it in extensive simulations. The setup of these experiments and their results are discussed in Section 3.4. We elaborate the contribution of this chapter in relation to previous works in Section 3.5 and conclude this chapter in Section 3.6.

3.1 Problem Statement

3.1.1 Planning Problem

In this chapter we consider prehensile manipulation of a single rigid object by a robot. The configuration of a system is written as vector $\mathbf{q} \in \mathbb{R}^n$. This configuration $\mathbf{q} = \left[\mathbf{q}_\text{r}^\top, \mathbf{q}_\text{o}^\top\right]^\top$ is composed of a robot configuration $\mathbf{q}_\text{r} \in \mathbb{R}^{n_\text{r}}$ and of an object configuration $\mathbf{q}_\text{o} \in \mathbb{R}^{n_\text{o}}$. In the example of Figure 3.1 the robot configuration can be described by $n_\text{r} = 7$ axis positions and the object configuration by $n_\text{o} = 7$ values for a vector-quaternion representation.

At all times, the object is assumed to be rigidly attached to the robot or to the environment, i. e., it is either grasped or placed. This state of attachment is described via the contact parameter $\pi \in \Pi$. A contact parameter describes to which link of robot or environment the object is attached as well as the relative transform between link and object. Here, Π is the set of possible contact parameters. Figure 3.2 shows a schematic example of a robot and an object with three different contact parameters.

Given a configuration of the robot $\mathbf{q}_\mathrm{r}$ and a contact parameter π one can compute the object configuration via forward kinematics $\mathbf{q}_\mathrm{o} = \mathrm{fk}(\mathbf{q}_\mathrm{r}, \pi)$. With slight abuse of notation, we treat a pair $(\mathbf{q}_\mathrm{r}, \pi) \in \mathbb{R}^{n_\mathrm{r}} \times \Pi$ equivalently to $\mathbf{q} \in \mathbb{R}^n$.

At all times our system must fulfill constraints on the configuration $\mathbf{q}$. Axis-limits of the robot can be explicitly represented via the configuration space of the robot $\mathbf{q}_\mathrm{r} \in \mathcal{Q}_\mathrm{r} \subset \mathbb{R}^{n_\mathrm{r}}$. We want to prevent the system from colliding with itself, the object, and the environment. Given a contact parameter π, the set $\mathcal{Q}_{\mathrm{free},\pi} \subset \mathcal{Q}_\mathrm{r}$ denotes the robot configurations that do not cause collision. This set is visualized in Figure 3.2 for three different contact parameters.

To manipulate an object it is necessary to grasp or to release it, i. e., to change the contact parameter. Let π_x be a contact parameter where an object is placed and π_y where it is grasped. To change from π_x to π_y it is necessary that the gripper of the robot is positioned correctly relative to the placed object. Formally, to change between contact parameters π_x and π_y the robot configuration $\mathbf{q}_\mathrm{r}$ must be within a set of transition configurations $\mathcal{Q}_{\pi_x,\pi_y} \subset \mathbb{R}^{n_\mathrm{r}}$. In Figure 3.2 these sets consist of only one point and lie at the end of the dashed blue lines. For most pairs of contact parameters these sets are empty. With a redundant manipulator as shown in Figure 3.1 these sets may consist of several disconnected manifolds that represent the solutions of the inverse kinematics problem of positioning the gripper.

Let $\tau : [0, 1] \to \mathcal{Q}_\mathrm{r}$ be a continuous path segment in the robot configuration space. A path $\{(\tau_i, \pi_i)\}_{i \le k}$ with $i, k \in \mathbb{N}_{>0}$ is a sequence of path segments and contact parameters of length k. Please note that τ_i is a trajectory of the robot configuration only. The trajectory of the full system is encoded in the pair (τ_i, π_i). In this chapter we consider geometric planning only. For this reason we omit velocities and controls and thus may normalize time intervals as $[0, 1]$.

Using this definition of paths and path segments, we can now define valid paths. Intuitively, a valid path is a motion of robot and object that is geometrically plausible. This means that (1) the path must be collision free, (2) the configuration of the robot is continuous across path segments, and (3) the contact parameter may only change from π_i to π_{i+1} at physically plausible transition configurations within $\mathcal{Q}_{\pi_i,\pi_{i+1}}$. In Figure 3.2 the dashed line shows a valid path.

Definition 1 (Valid Path) *A path is valid iff*

(1) $\tau_i(t) \in \mathcal{Q}_{\mathrm{free},\pi_i}$ *for* $t \in [0, 1], i \in 1 \dots k$,

(2) $\tau_i(1) = \tau_{i+1}(0)$ *for* $i \in 1 \dots k-1$, *and*

(3) $\tau_i(1) \in \mathcal{Q}_{\pi_i,\pi_{i+1}}$ *for* $i \in 1 \dots k-1$.

We assume that the system is at an initial configuration $\mathbf{q}_\mathrm{start} = (\mathbf{q}_\mathrm{r,start}, \pi_\mathrm{start})$ and should reach a set of goal states $\mathcal{Q}_{\text{goal}} \subset \mathbb{R}^n$. We define feasible paths as those that (1) are valid, (2) begin in the initial state, and (3) end in the goal set.

Definition 2 (Feasible Path) *A path is feasible iff*

(1) *it is valid,*

(2) $(\tau_1(0), \pi_1) = (\mathbf{q}_{\text{r,start}}, \pi_{\text{start}})$*, and*

(3) $(\tau_k(1), \pi_k) \in \mathcal{Q}_{\text{goal}}$.

The cost function C assigns a non-negative cost to each path:

$$C : \{(\tau_i, \pi_i)\}_{i \leq k} \to \sum_{i=1}^{k} C_{\text{p}}(\tau_i) + \sum_{i=1}^{k-1} C_{\text{t}}(\pi_i, \pi_{i+1}). \tag{3.1}$$

This cost is composed of two components: The function C_{p} assigns non-negative cost to each path segment. An example for C_{p} is the Euclidean distance that the robot travels within $\mathbb{R}^{n_{\text{r}}}$. The function C_{t} assigns positive and lower-bounded cost to transitions, with $C_{\text{t}} > C_{\text{t,min}} > 0$. An example is a constant cost per transition. Our rationale here is that we want to minimize both the motions of the robot and the number of (re-)grasps. Let $\mathcal{F}$ be the set of feasible paths. Given the cost function we can define the optimal cost.

Definition 3 (Optimal Cost) *The optimal cost* $C^* \in \mathbb{R}$ *is the infinum of the cost function across all feasible paths* $\{(\tau_i, \pi_i)\} \in \mathcal{F}$*:*

$$C^* = \inf_{\{(\tau_i,\pi_i)\} \in \mathcal{F}} C(\{(\tau_i, \pi_i)\})$$

The goals for feasible and optimal planning are now to find feasible paths or feasible paths that approach the optimal cost respectively.

3.1.2 Primitive Operations

We assume that a set of primitive operations, common to manipulation, is available to our planner. The procedure **sampleFree**(π) returns a random sample of $\mathcal{Q}_{\text{free},\pi}$ or failure if this is not possible. A simple implementation is rejection sampling within $\mathcal{Q}_{\text{r}}$ combined with a collision check.

A call to **sampleContact** returns a random contact parameter $\pi \in \Pi$. This can be implemented with an algorithm that generates grasp poses and placement poses.

Finally, **sampleTransition**(π_1, π_2) returns a random sample of $\mathcal{Q}_{\pi_1,\pi_2}$. A practical implementation is a randomized inverse kinematics solver. As the set $\mathcal{Q}_{\pi_1,\pi_2}$ is empty for most pairs of contact parameters this procedure may return failure as well.

3.2 Optimal Manipulation Planner

This section introduces a novel and asymptotically optimal manipulation planner. Section 3.2.1 explains our algorithm: Optimal Random Manipulation Roadmap (RMR*). In Section 3.2.2 we introduce two strategies to speed up planning.

3.2.1 Algorithm

Our planner operates with two phases. In the first phase, called roadmap construction, a large probabilistic roadmap is built across different contact states. This roadmap is reusable for multiple planning queries if the environment of the robot and the shape of the manipulated object remain constant. In the second phase, called query, a path is planned from an initial configuration towards a goal set.

The roadmap construction phase of RMR* is shown in procedure **build-Roadmap**. It takes the integers N_c, N_i, and N_t as input to determine the size of the roadmap. **buildRoadmap** proceeds to build an undirected graph of nodes $\mathcal{N}$ and edges $\mathcal{E}$. We sample N_c contact parameters using **sampleContact** and store them in the set Π_{sampled}. For each contact parameter $\pi \in \Pi_{\text{sampled}}$, a roadmap within $\mathcal{Q}_{\text{free},\pi}$ is built according to the PRM* algorithm [13]. This is done with procedure **buildPRM***. Finally, the separate roadmaps are connected via transitions using procedure **connectRoadmaps**.

Procedure: buildRoadmap(N_c, N_i, N_t **)**

```
1 $\mathcal{N} \leftarrow \{\}$    $\mathcal{E} \leftarrow \{\}$
2 $\Pi_{\text{sampled}} \leftarrow \{\}$
3 for $1 \leq i \leq N_c$ do
4     $\pi_{\text{new}}$ = sampleContact( )
5     $\Pi_{\text{sampled}}$.add( $\pi_{\text{new}}$ )
6     buildPRM*( $\pi_{\text{new}}, N_i$ )
7 for $\pi_1 \neq \pi_2 \in \Pi_{\text{sampled}}$ do
8     connectRoadmaps( $\pi_1, \pi_2, N_t$ )
```

PRM* [13] is an asymptotically optimal motion planner. The procedure **buildPRM*** takes a contact parameter π and the integer N_i as input and proceeds to build a probabilistic roadmap with at most N_i samples within $\mathcal{Q}_{\text{free},\pi}$. The procedure **connectConfiguration** is a shorthand for the connection mechanism of PRM*. This procedure adds edges to $\mathcal{E}$ between a configuration $(\mathbf{q}_{r,\text{new}}, \pi)$ and nodes of $\mathcal{N}$ that have the same contact parameter π.

To connect two roadmaps with different contact parameters π_x and π_y the procedure **connectRoadmaps** samples at most N_t transitions within $\mathcal{Q}_{\pi_x,\pi_y}$. Then **connectConfiguration** is used to connect these transitions to the separate roadmaps. The result is a large probabilistic roadmap that spans multiple contact states.

```
Procedure: buildPRM*( $\pi, N_i$ )
1 for $1 \leq i \leq N_i$ do
2     $\mathbf{q}_{r,i} \leftarrow$ sampleFree( $\pi$ )
3     $\mathcal{N}$.add( $(\mathbf{q}_{r,i}, \pi)$ )
4 for $1 \leq i \leq N_i$ do
5     connectConfiguration( $\mathbf{q}_{r,i}, \pi$ )
```

```
Procedure: connectRoadmaps( $\pi_1, \pi_2, N_t$ )
1 for $1 \leq i \leq N_t$ do
2     $\mathbf{q}_{r,\text{new}} \leftarrow$ sampleTransition( $\pi_1, \pi_2$ )
3     $\mathcal{N}$.add( $(\mathbf{q}_{r,\text{new}}, -)$ )
4     connectConfiguration( $\mathbf{q}_{r,\text{new}}, \pi_1$ )
5     connectConfiguration( $\mathbf{q}_{r,\text{new}}, \pi_2$ )
```

The second phase of our algorithm is the query phase. In the spirit of the PRM algorithm [4], we first attempt to connect a start configuration ($\mathbf{q}_{\text{r,start}}, \pi_{\text{start}}$) to the previously constructed manipulation roadmap. Then we use a standard graph search algorithm, e. g., [14], to find the minimal-cost path to a goal set $\mathcal{Q}_{\text{goal}}$. Typically, the initial contact state π_{start} is not an element of the sampled contact parameters Π_{sampled}. Therefore, it is necessary to construct a docking roadmap with N_i nodes using PRM*. The start configuration is connected to this roadmap. We then connect the docking roadmap to the manipulation roadmap with at most N_t transitions per contact in Π_{sampled}. Figure 3.3 visualizes the roadmaps built by RMR*.

```
Procedure: query( $(\mathbf{q}_{r,\text{start}}, \pi_{\text{start}}), \mathcal{Q}_{\text{goal}}, N_i, N_t$ )
1 $\mathcal{N}$.add( $(\mathbf{q}_{r,\text{start}}, \pi_{\text{start}})$ )
2 buildPRM*( $\pi_{\text{start}}, N_i$ )
3 connectConfiguration( $\mathbf{q}_{r,\text{start}}, \pi_{\text{start}}$ )
4 for $\pi \in \Pi_{\text{sampled}}$ do
5     connectRoadmaps( $\pi, \pi_{\text{start}}, N_t$ )
6 return graphSearch( $(\mathbf{q}_{r,\text{start}}, \pi_{\text{start}}), (\mathcal{N}, \mathcal{E}), \mathcal{Q}_{\text{goal}}$ )
```

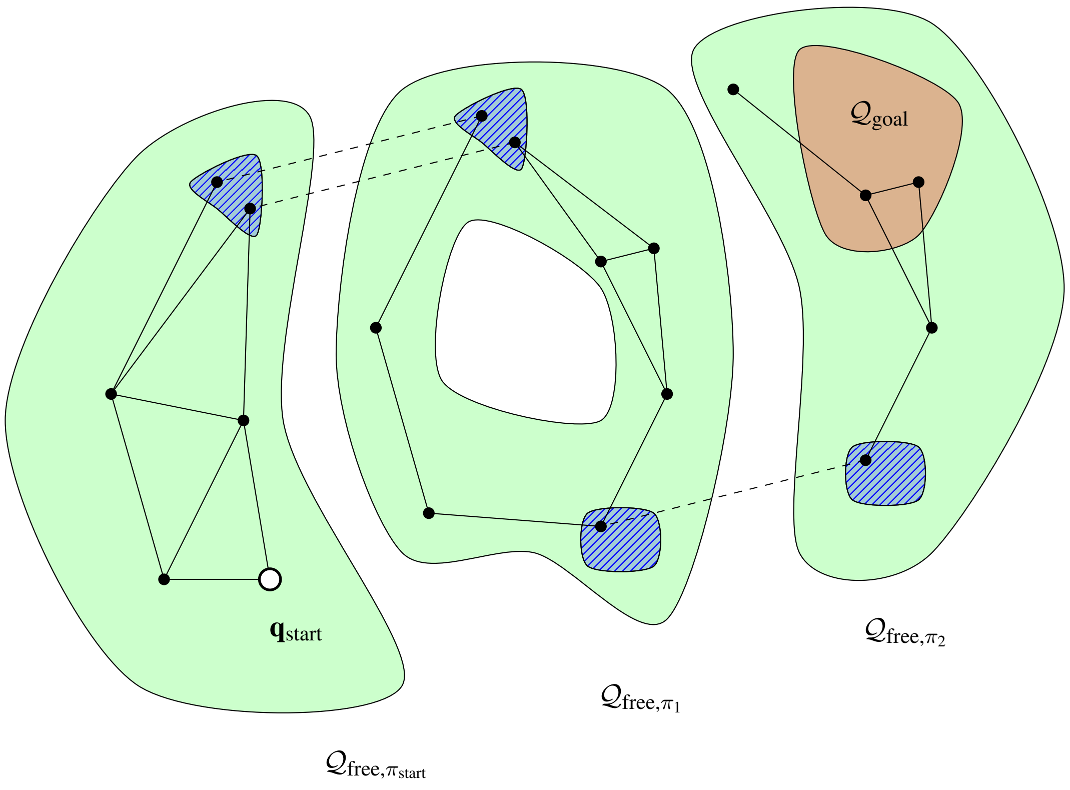

$\mathcal{Q}_{\text{goal}}$
$\mathbf{q}_{\text{start}}$
$\mathcal{Q}_{\text{free},\pi_2}$
$\mathcal{Q}_{\text{free},\pi_1}$
$\mathcal{Q}_{\text{free},\pi_{\text{start}}}$

Figure 3.3: Roadmap construction of RMR*: The green areas visualize the valid areas $\mathcal{Q}_{\text{free},\pi_x}$ of the robot configuration space for three different contact parameters π_x. The striped areas show transition regions $\mathcal{Q}_{\pi_x,\pi_y}$. Dots mark nodes of the roadmaps built by RMR*. Solid lines mark edges for which the robot moves but the contact state stays constant. Dashed lines mark a change of contact state.
The two roadmaps on the right with contact states π_1 and π_2 are constructed during the offline roadmap construction phase. The roadmap on the left with contact state π_{start} is constructed during the query phase.

3.2.2 Roadmap Re-Use and Lazy Collision Checking

Even though the roadmap construction phase of our algorithm must be called only once, its runtime quickly becomes a bottleneck as hundreds of contact states and thousands of nodes per contact are sampled. To reduce runtime, we re-use collision checks and nearest neighbor searches across the construction of the N_c within-contact roadmaps. Furthermore we employ lazy collision checking [15].

Instead of building the within-contact roadmaps at line 7 of **buildRoadmap**, we first build a PRM* with N_i nodes for a planning scene with no object. We can then check which nodes and edges of this roadmap lie within $\mathcal{Q}_{\text{free},\pi}$ for all $\pi \in \Pi_{\text{sampled}}$. This has several advantages. We will never sample a configuration or try to connect an edge for which the robot is in self-collision or collides with its static environment. Furthermore, nearest-neighbor search can be shared across all N_c roadmaps and the necessary data-structures for search can be reused for the connection of transitions.

As a second measure to reduce the runtime of our approach, we do not check if an edge of the within-contact roadmaps is valid during roadmap construction. We employ the graph search algorithm at the end of procedure **query** with the assumption that all edges are valid. If a path is returned, we check only edges on this path. Should one edge be invalid, we remove it from the set of edges E and repeat the graph search.

3.2.3 Detailed Illustration of RMR*

In this section we illustrate the operation of RMR* step by step. Manipulation planning is notoriously difficult to visualize and it is worth to go through our planner in detail to avoid misunderstandings. We use the manipulation task of Figure 3.2 as exemplary planning problem. The task comprises a robot and a cube that must be placed on the right side of the 2D environment.

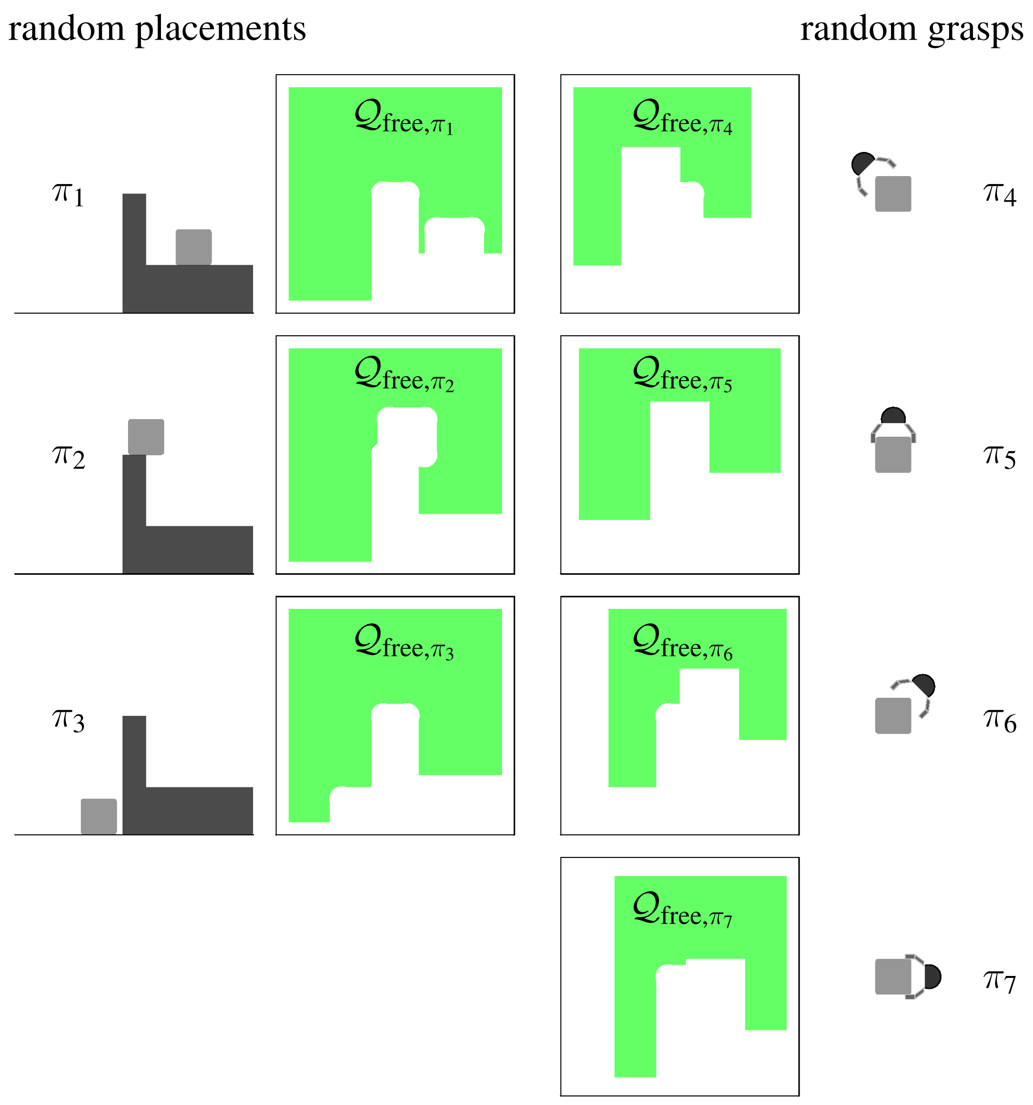

random placements
random grasps
π_1
π_2
π_3
π_4
π_5
π_6
π_7
$\mathcal{Q}_{\text{free},\pi_1}$
$\mathcal{Q}_{\text{free},\pi_2}$
$\mathcal{Q}_{\text{free},\pi_3}$
$\mathcal{Q}_{\text{free},\pi_4}$
$\mathcal{Q}_{\text{free},\pi_5}$
$\mathcal{Q}_{\text{free},\pi_6}$
$\mathcal{Q}_{\text{free},\pi_7}$

Figure 3.4: Sample contact parameters: The first step of RMR* is to sample a set of N_c contact parameters using procedure **sampleContact** (Lines 5 and 6 of **buildRoadmap**). To the left three different, random placements of the cube are shown (π_1 to π_3). Four different grasps are shown to the right (π_4 to π_7). The green areas show the corresponding collision free robot configurations.

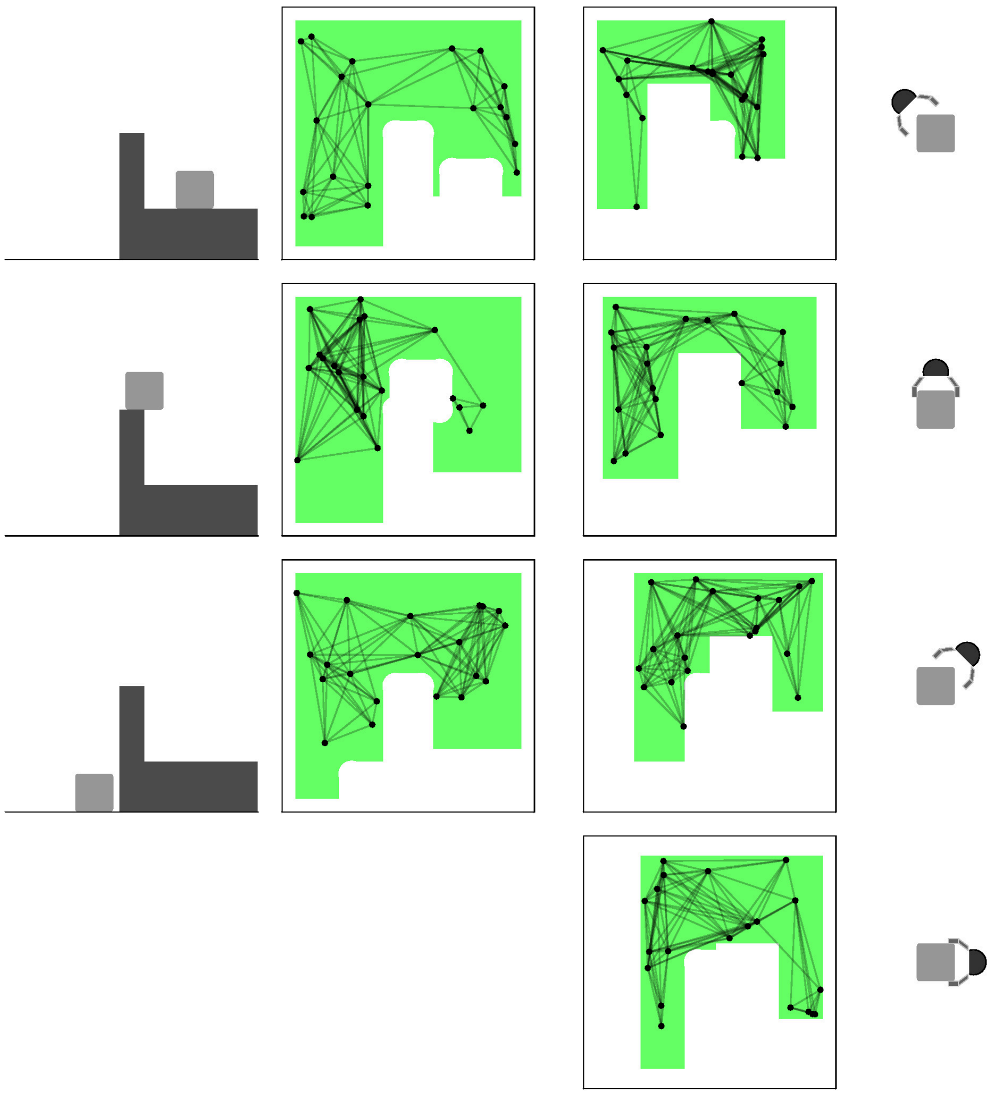


Figure 3.5: Build one roadmap per contact: For each contact parameter π a roadmap is build within $\mathcal{Q}_{\text{free},\pi}$ (Line 7 of **buildRoadmap**). Each roadmap is built with the PRM* algorithm and contains at most N_i nodes. PRM* is an optimal motion planner and in the limit of N_i all pairs of nodes within a contact parameter will be connected with optimal paths.

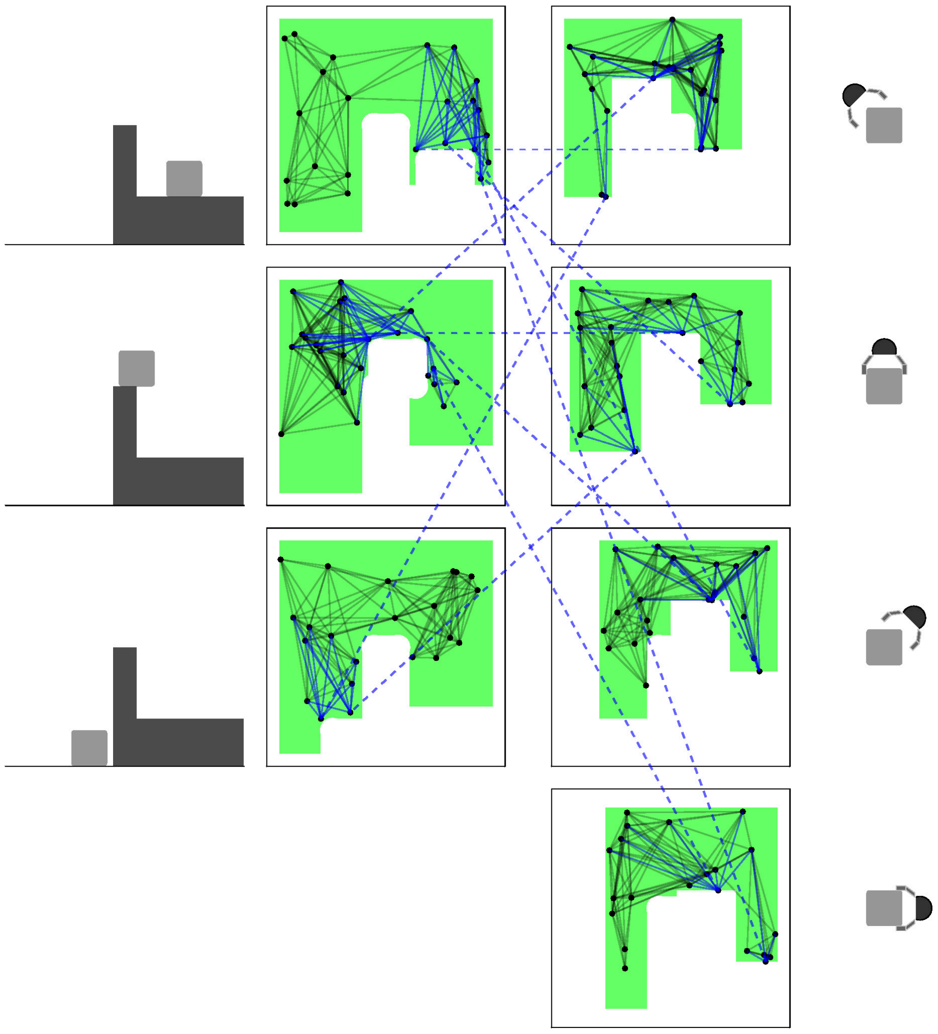


Figure 3.6: Connect roadmaps: The individual roadmaps that were built with PRM* do not have any connections across contacts. To connect the full roadmap, we sample and connect transition configurations (Line 9 of **buildRoadmap**).

This is done in procedure **connectRoadmaps**. For a pair of contacts π_x and π_y, N_t transitions are sampled within $\mathcal{Q}_{\pi_x,\pi_y}$ (Lines 2 and 3 of **connectRoadmaps**). These transitions are then connected to the roadmaps within $\mathcal{Q}_{\text{free},\pi_x}$ and $\mathcal{Q}_{\text{free},\pi_y}$ via the connection mechanism of PRM* (Lines 4 and 5 of **connectRoadmaps**). It is important to note that the transition regions $\mathcal{Q}_{\pi_x,\pi_y}$ may be continuous sets for real robots. In the 2D-example these sets consist of only one point and thus only one transition is drawn per pair of contacts.

Connecting the roadmaps results in a large manipulation roadmap that spans multiple contact states. As long as the environment of the robot and the shape of the manipulated object remain constant, this roadmap can be reused for multiple planning queries.

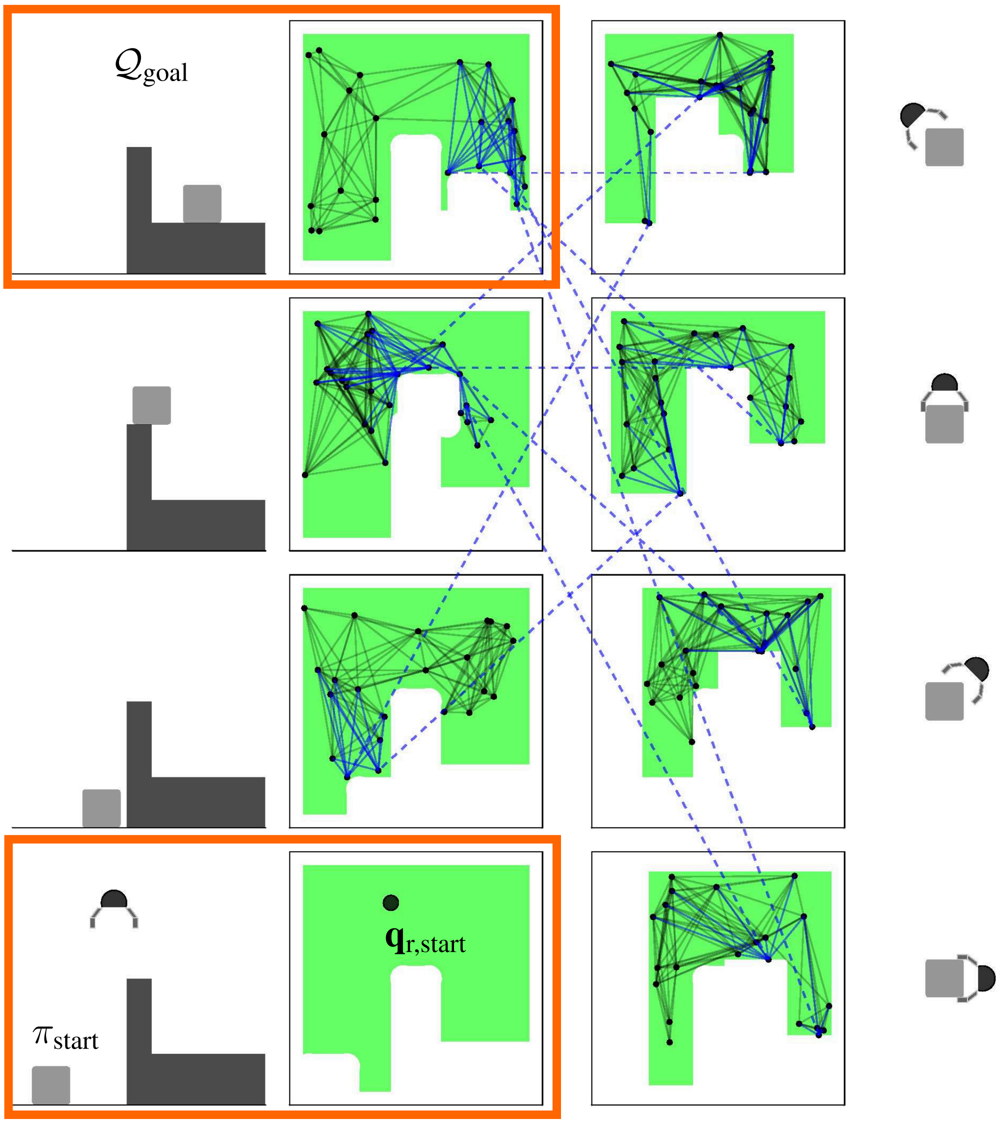


Figure 3.7: Query: An online query to the planner comprises an initial configuration ($\mathbf{q}_{\text{r,start}}$, π_{start}) and a goal set $\mathcal{Q}_{\text{goal}}$. Both the robot configuration space $\mathcal{Q}_{\text{r}}$ and the set of contact parameters Π are continuous sets. For this reason there is typically zero probability that the initial configuration is part of the roadmap.

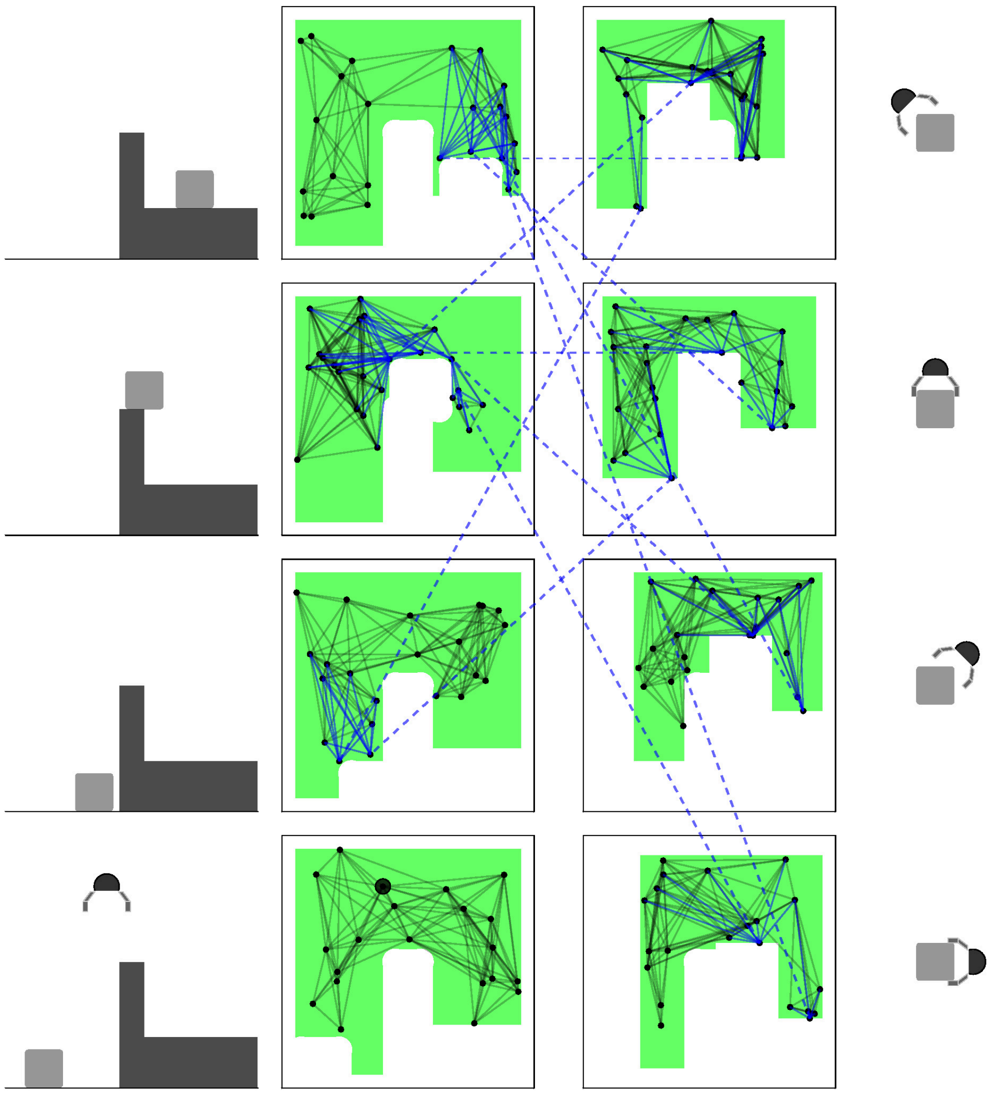


Figure 3.8: Build docking roadmap: As first step of the query a so called docking roadmap is build within $\mathcal{Q}_{\text{free},\pi_{\text{start}}}$ (Line 2 of **query**). The initial robot configuration $\mathbf{q}_{\text{r,start}}$ is connected to this docking roadmap in Line 3 of **query**.

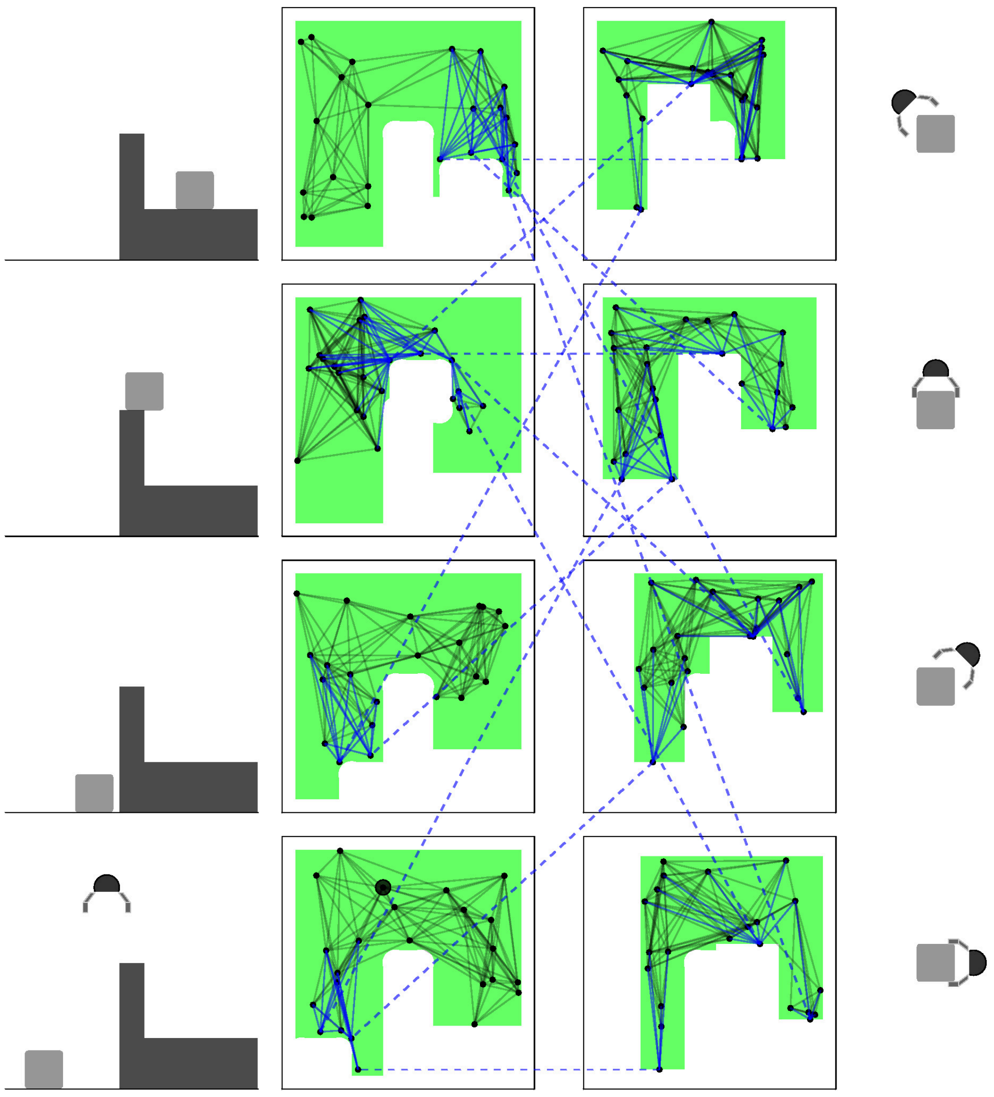


Figure 3.9: Connect docking roadmap: To connect the docking roadmap to the remaining graph we again sample and connect transitions (Line 5 of **query**). This results in a large roadmap that is also connected to the initial configuration.

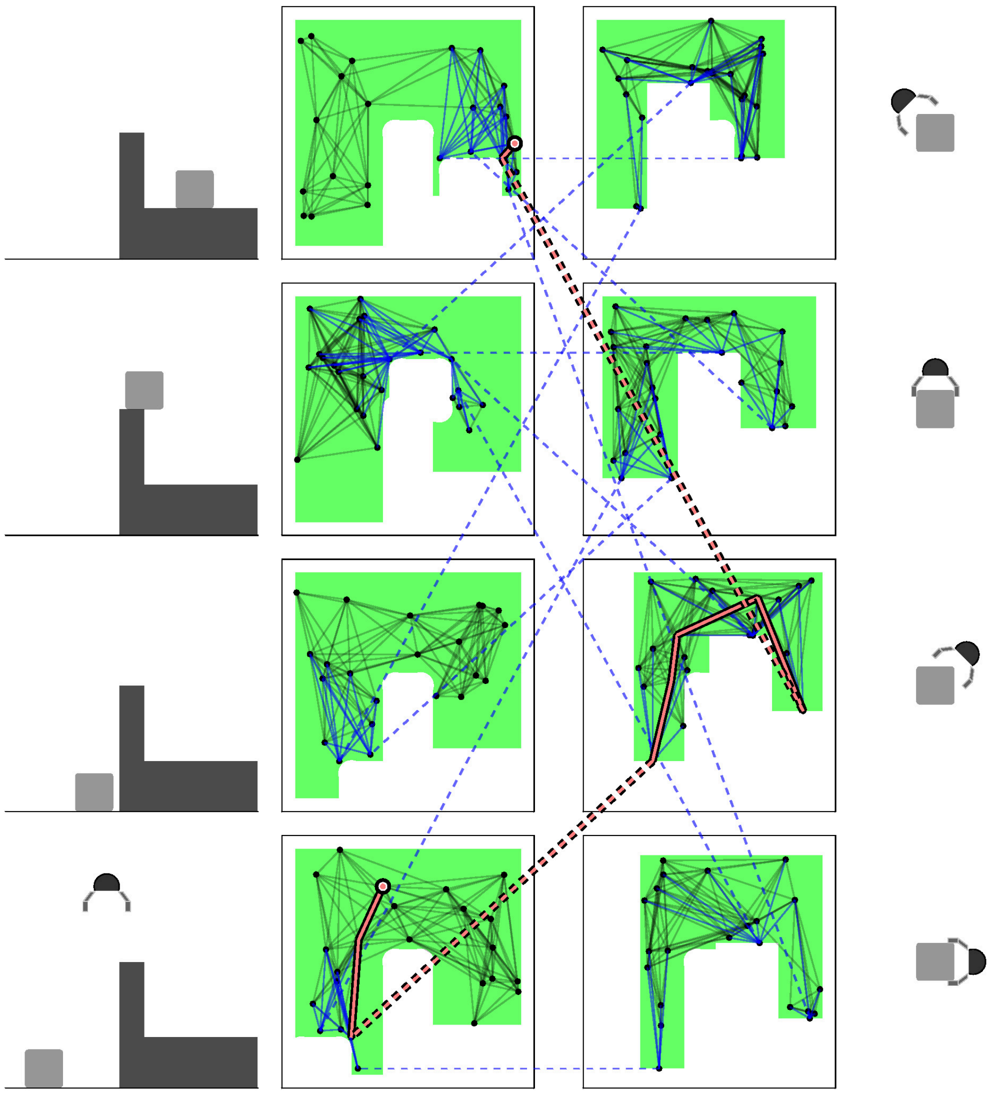


Figure 3.10: Graph search: After the initial configuration is connected to the roadmap a standard graph search is used to find a path to a configuration within the goal set. (Line 6 of **query**)

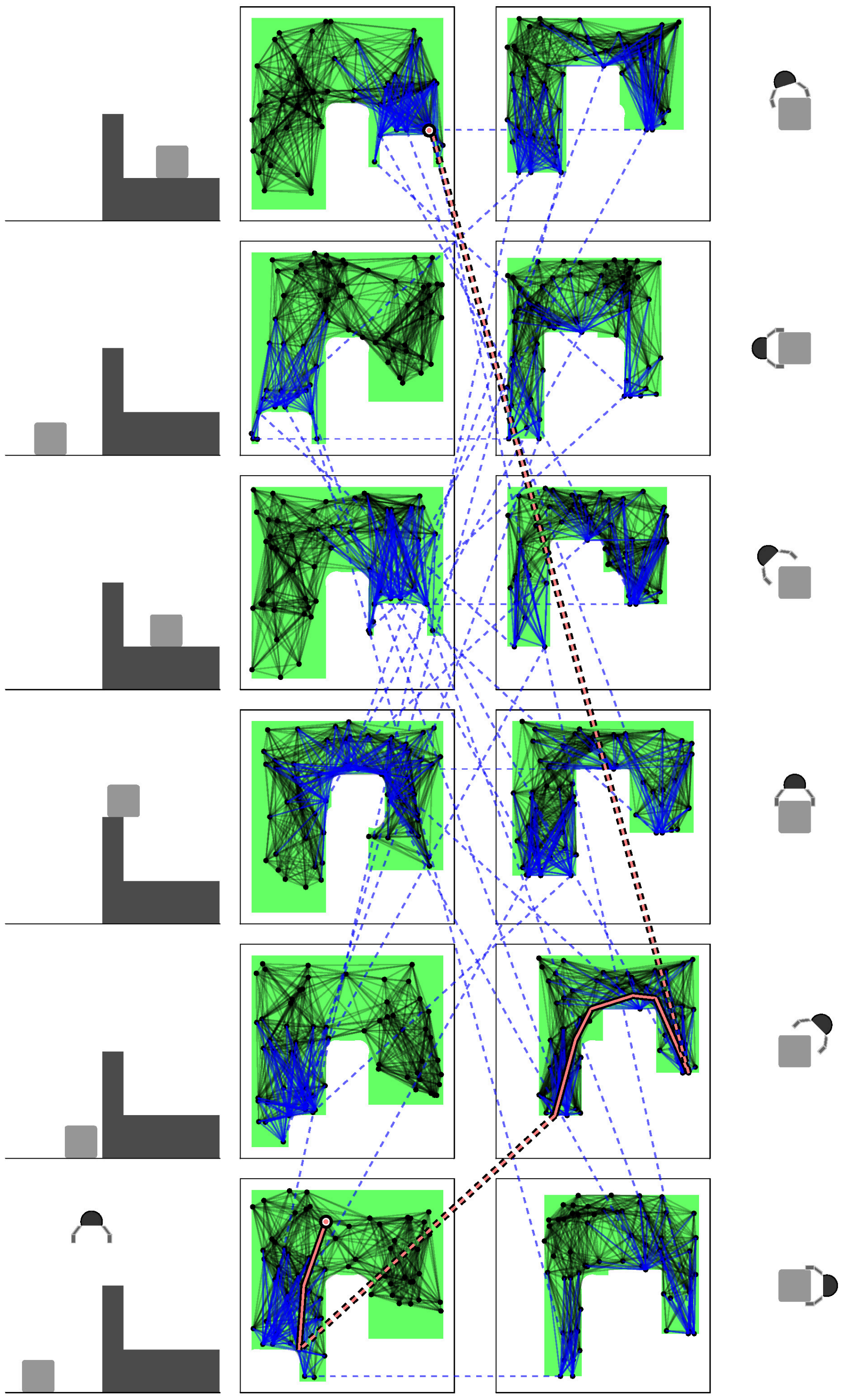


Figure 3.11: Convergence to the optimum of RMR*: As the size of the roadmap is increased with the parameters N_c, N_i, and N_t the solution improves on average. In the limit the configuration space of robot and object is densely explored and the cost of the solution almost surely converges to the optimum.

3.3 Analysis

3.3.1 Computational Complexity

Let us recall that the procedure **connectConfiguration**($\mathbf{q}_{r,\text{new}}, \pi$) represents the connection mechanism of the PRM* algorithm. This procedure has a complexity of $\mathcal{O}(\log n)$ where n is the number of nodes within the roadmap that $\mathbf{q}_{r,\text{new}}$ should be connected to. Building an entire PRM* has a complexity of $\mathcal{O}(n \log n)$ [13]. Within RMR*, n is the number of nodes that share the same contact parameter π. Thus, $n = N_i$ holds.

We are building a total of N_c roadmaps with N_i nodes each. So the construction of the roadmaps without transitions has a complexity of $\mathcal{O}(N_c N_i \log N_i)$.

As we sample N_c contact states, there are less than N_c^2 transition regions within the constructed roadmap. For each of the transition regions at most N_t transitions are sampled and connected via **connectConfiguration**. Thus, connecting the individual roadmaps via transitions has a complexity of $\mathcal{O}(N_c^2 N_t \log N_i)$. The complexity of the roadmap construction phase of RMR* is therefore $\mathcal{O}((N_c N_t + N_i) N_c \log N_i)$.

Building the docking roadmap and connecting it to the manipulation roadmap has a time complexity of $\mathcal{O}((N_c N_t + N_i) \log N_i)$. The following upper bounds hold for the number of vertices and edges within (V, E):

$$|V| \in \mathcal{O}((N_c N_t + N_i) N_c)$$

$$|E| \in \mathcal{O}((N_c N_t + N_i) N_c \log N_i)$$

These upper bounds can be used to estimate the time complexity of the graph search via $\mathcal{O}(|E| + |V| \log |V|)$ [16].

3.3.2 Completeness and Optimality

In order to prove probabilistic completeness and asymptotic optimality of RMR* we need to define some properties of the planning problems.

Definition 4 (Segment Robustness)
A triple $(\mathbf{q}_{\text{r},1}, \mathbf{q}_{\text{r},2}, \pi) \in \mathcal{Q}_{\text{r}} \times \mathcal{Q}_{\text{r}} \times \Pi$ is segment robust iff the PRM algorithm is asymptotically optimal while planning from $\mathbf{q}_{\text{r},1}$ to $\mathbf{q}_{\text{r},2}$ within $\mathcal{Q}_{free,\pi}$.*

Definition 5 (Goal Robustness)
A tuple $(\mathbf{q}_{\text{r}}, \pi) \in \mathcal{Q}_{\text{r}} \times \Pi$ is goal robust iff $(\mathcal{Q}_{free,\pi} \times \{\pi\}) \cap \mathcal{Q}_{\text{goal}} \neq \emptyset$ and the PRM algorithm is asymptotically optimal while planning from $\mathbf{q}_{\text{r}}$ to $\mathcal{Q}_{\text{goal}}$ within $\mathcal{Q}_{free,\pi}$.*

The required conditions for segment and goal robustness are omitted for brevity and can be found in [13].

Let $\{\pi_i\}_{i \leq k}$ with $i, k \in \mathbb{N}_{>0}$ be a sequence of contact states. Also, let $\{\mathbf{q}_{\mathrm{r},j}\}_{j<k}$ with $j \in \mathbb{N}_{>0}$ be a sequence of robot configurations that allow a transition between these contact states. The optimal cost is denoted as C^*. The value $C^*(\{\pi_i\})$ denotes the cost of an optimal path using only the contact states within $\{\pi_i\}$. Finally, $C^*(\{\pi_i\}, \{\mathbf{q}_{\mathrm{r},j}\})$ is defined as the optimal path cost using only the contacts in $\{\pi_i\}$ and transitions in $\{\mathbf{q}_{\mathrm{r},j}\}$. These definitions imply:

$$C^* \leq C^*(\{\pi_i\}) \leq C^*(\{\pi_i\}, \{\mathbf{q}_{\mathrm{r},j}\}).$$

Definition 6 (Transition Robustness)
*A pair of a configuration and a sequence of contacts $((\mathbf{q}_{\mathrm{r,start}}, \pi_{\mathrm{start}}), \{\pi_i\})$ is transition robust iff: For every $\epsilon \in \mathbb{R}_{>0}$ there exists a probability $P_\epsilon > 0$, such that when sampling a sequence of transitions $\{\mathbf{q}_{\mathrm{r},j}\}$ using **sampleTransition** the following holds at least with probability P_ϵ:*

- $C^*(\{\pi_i\}, \{\mathbf{q}_{\mathrm{r},j}\}) \leq C^*(\{\pi_i\}) + \epsilon$
- $(\mathbf{q}_{\mathrm{r},k-1}, \pi_k)$ *is goal robust.*
- *All consecutive pairs of $\mathbf{q}_{\mathrm{r,start}}$ and $\mathbf{q}_{\mathrm{r},j}$ are segment robust within the corresponding contact states.*

Intuitively, transition robustness states that sampling transitions between contacts and connecting them to PRM* roadmaps within these contacts is equivalent to building one large PRM*.

As transition costs are lower bounded by $C_{\mathrm{t,min}} > 0$, it is sufficient to consider only a paths with a finite number of contact states. Let k^* be the smallest number of contacts necessary to approximate C^*.

Definition 7 (Contact Robustness)
*A planning problem is contact robust iff: For every $\epsilon \in \mathbb{R}_{>0}$ there exists $P_\epsilon > 0$, such that when sampling a sequence of contacts $\{\pi_i\}_{i \leq k^*}$ using **sampleContact** the following holds at least with probability P_ϵ:*

- $C^*(\{\pi_i\}) \leq C^* + \epsilon$
- $((\mathbf{q}_{\mathrm{r,start}}, \pi_{\mathrm{start}}), \{\pi_i\})$ *is transition robust.*

For problems, that have a solution, the random variable $C_{\mathrm{RMR}^*}(N_c, N_i, N_t)$ is defined as the solution cost returned by our planner. If our planner fails this variable is set to $C_{\mathrm{fail}} \gg C^*$.

Theorem 1 (Optimality of RMR*) *For a planning problem that is contact robust, RMR* almost surely converges to the optimal cost C^* as N_c, N_i, and N_t approach infinity.*
$P(\lim_{N_C, N_i, N_t \to \infty} C_{\mathrm{RMR}^*}(N_c, N_i, N_t) = C^*) = 1$

Proof. Let $\epsilon > 0$. From the contact robustness of our planning problem follows, that RMR* will almost surely sample a sequence of k^* contacts $\{\pi_i\}$ which is transition robust and for which $C^*(\{\pi_i\}) \leq C^* + \epsilon$ holds, as N_c approaches infinity.

From the transition robustness of this sequence follows that RMR* will almost surely sample a set of k^*-1 transitions $\{\mathbf{q}_{\mathrm{r},j}\}$ which are consecutively segment robust, $(\mathbf{q}_{\mathrm{r},k^*-1}, \pi_{k^*})$ is goal robust and for which $C^*(\{\pi_i\}, \{\mathbf{q}_{\mathrm{r},j}\}) \leq C^*(\{\pi_i\}) + \epsilon$ holds, as N_t approaches infinity.

Between consecutive pairs of start configuration and transitions our algorithm will build increasingly larger PRM* roadmaps, which are asymptotically optimal in N_i due to segment robustness. Within contact state π_{k^*} PRM* is asymptotically optimal in N_i due to goal robustness. Therefore $C_{\mathrm{RMR}^*}(N_c, N_i, N_t)$ will, almost surely, not exceed $C^*(\{\pi_i\}, \{\mathbf{q}_{\mathrm{r},j}\})$ by more than ϵk^*, as N_i approaches infinity.

From this follows that RMR* will almost surely return a solution that is no larger than $C^* + \epsilon(2 + k^*)$ for any $\epsilon > 0$. This implies asymptotic optimality. ■

Theorem 2 (Probabilistic Completeness of RMR*) *For a planning problem that is contact robust, RMR* is probabilistically complete as N_c, N_i, and N_t approach infinity.*

This theorem follows trivially from asymptotic optimality. The discerning reader will have noticed, that our proofs do not build upon the underlying physical mechanics of manipulation, like the distribution of placements or grasps in Π or the shapes of $\mathcal{Q}_{\pi_x,\pi_y}$. Instead we make assumptions on probabilities of sampling contacts or transitions in helpful areas of Π and $\mathcal{Q}_{\mathrm{r}}$. We argue that the operations **sampleContact** and **sampleTransition** are typically not part of the solution but part of the problem setting. For this reason we do not build our proofs upon assumptions that lie within the mechanics of these operations.

3.4 Implementation and Experiments

3.4.1 Implementation Details and Experimental Setup

The industrial manipulator used throughout our experiments consists of a 7-axis, redundant robotic arm, a parallel gripper, and a monocular camera. We designed seven benchmark tasks that revolve around moving a cube into a target area on a table or into a box. For some of the tasks it is necessary to re-grasp the object several times in order to change its orientation.

The benchmarks are designed to pose problems with different levels of difficulty. It is well known that narrow tunnels in the configuration space pose difficult problem instances for sampling based planners [3]. In our experiments we added such tunnels by design, both in the configuration space of the robot (picking and placing in a box) and in the contact space of the object (changing the orientation of the object).

The experimental setup and one of the two initial positions of the object can be seen in Figure 3.12. In all experiments the joints of the robot were initially at their zero position.

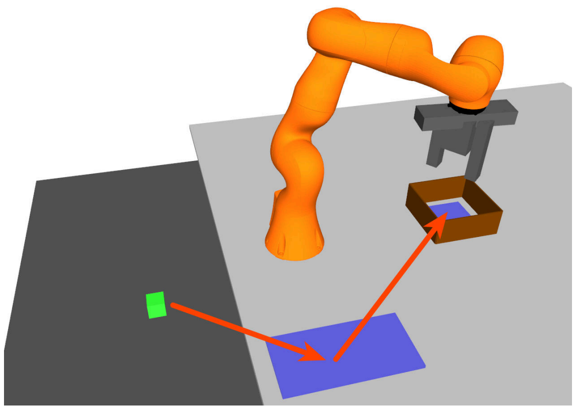


Figure 3.12: Experimental Setup: The cube on the left must be brought into one of goal regions marked in blue. The arrows depict the optimal manipulation sequence for benchmark 4. First, the object is placed on the table with a 90 degree turn. Then it is re-grasped and placed bottom up in the box.

Table 3.1: Benchmark Problems

#	Initial Object Position	Goal Area	Goal Orientation
1	lower table	upper table	any
2	lower table	upper table	bottom up
3	lower table	box	any
4	lower table	box	bottom up
5	box	upper table	any
6	box	upper table	bottom up
7	box	box	bottom up

The blue areas mark the two goal regions for the object. Table 3.1 lists the seven benchmark tasks used in our experiments.

As the cost function for the path segments C_p we chose the Euclidean distance traveled in the robot configuration space. The cost for transitions C_t was chosen to be 3.0 for all transitions. With this cost function transitions and motions of the robot have approximately equal cost in the experiments. This equally discourages unnecessary re-grasps and inefficient motions.

Grasps and placements of the cube were randomly distributed around its six faces. Placements are only sampled within the areas marked blue in Figure 3.12. Transitions are computed via rejection sampling. We used an analytic inverse kinematics solver for which the redundancy parameters were chosen randomly.

To evaluate our planner we compared the quality of its solutions to that of the Probabilis-

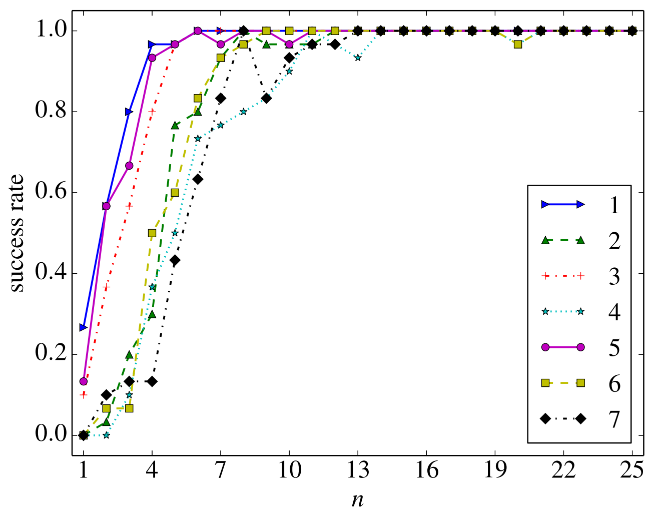


Figure 3.13: Success rates for the seven benchmark tasks: Each line visualizes the success rate for one benchmark with different parameter settings for our algorithm: $N_c = 10n$, $N_i = 100n$, $N_t = n$.

tic Tree of Roadmaps (PTR) planner presented in [11]. This approach from the literature was chosen for two reasons: It is capable of handling manipulation planning problems with continuous contact states without requiring complete motion planners. Furthermore, in its basic form, it does not require domain-specific adaptations, like heuristics, that are hand crafted for the problem class at hand.

Both algorithms were then used to solve all of the seven benchmark tasks. Our approach was run with 25 different settings for N_c, N_t, and N_i. Due to the probabilistic nature of the two algorithms, we repeated this experiment 30 times.

The solutions of both approaches were additionally run through a standard path simplification [17]. We used the open source library FCL [18] for collision checks which was accessed via the planning scene of MoveIt! [19]. For nearest neighbor search we used randomized k-d trees [20] implemented in the FLANN library [21]. To speed up roadmap construction it was run in parallel using OpenMP [22]. All experiments were run on a ten-core Intel Xeon E5-2650v3.

3.4.2 Results

Figure 3.13 shows the success rates of our approach on all seven benchmarks with 25 different parameter settings. We have chosen to increase the input parameters N_c, N_i, and N_t in a linear fashion. At their highest setting, we build a manipulation roadmap with 250 contact states, 2500 nodes per within-contact roadmap, and at most 25 transitions between each pair of contact states. As one can see, the success rates quickly converge towards one for all benchmark tasks.

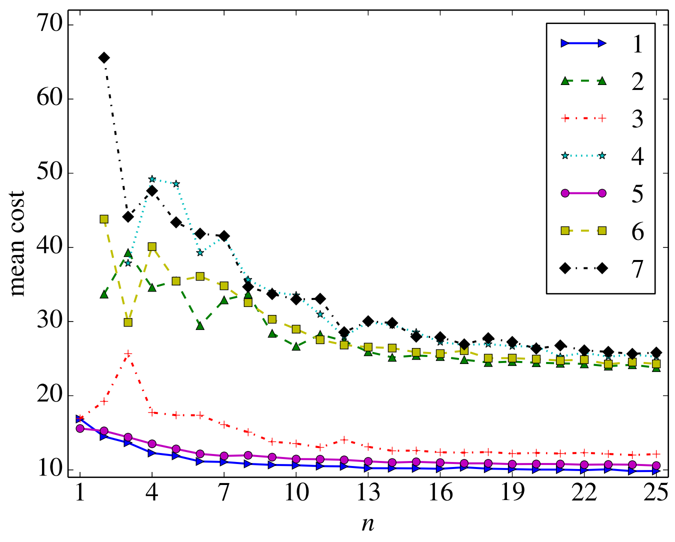


Figure 3.14: Path costs without post-processing for the seven benchmark tasks: Each line visualizes the average costs of successful runs for one benchmark with different parameter settings: $N_c = 10n$, $N_i = 100n$, $N_t = n$.

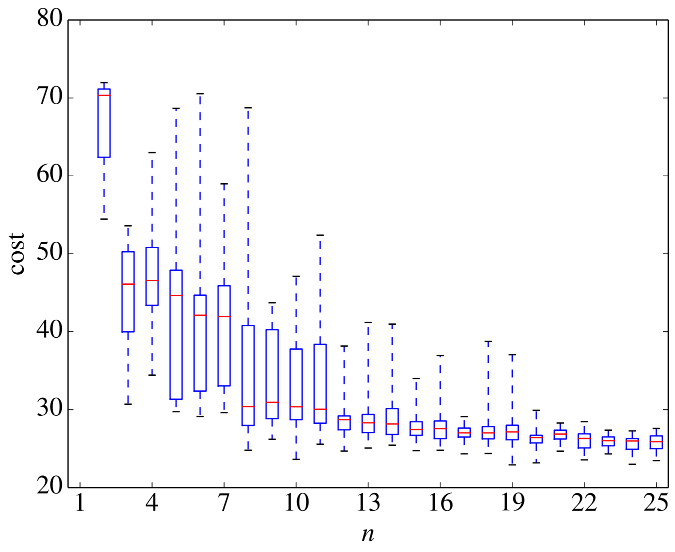


Figure 3.15: Distribution of path costs without post-processing for benchmark 7: Each box plot visualizes the distribution of costs for different parameter settings for our algorithm: $N_c = 10n$, $N_i = 100n$, $N_t = n$. The red line depicts the median. The first and third quartile are represented by the box, minimum and maximum by the whiskers.

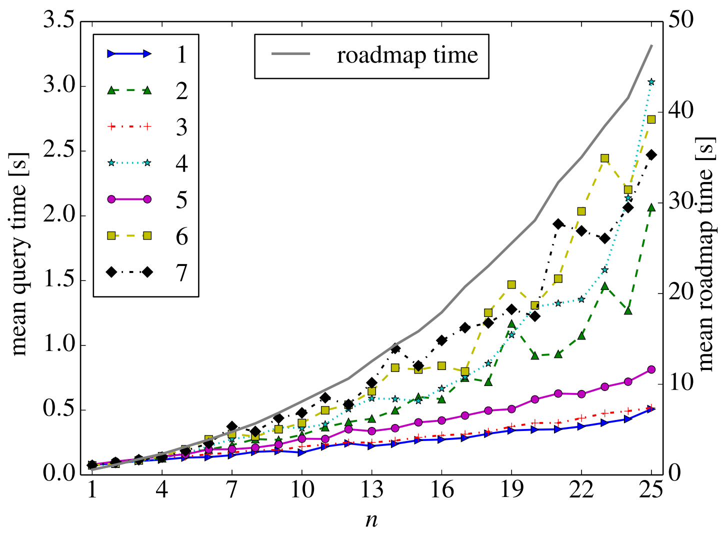


Figure 3.16: Average query and roadmap times without post-processing for the benchmark tasks with different parameter settings: $N_c = 10n$, $N_i = 100n$, $N_t = n$.

Figure 3.14 depicts the average cost for successful queries and indicates convergence in solution cost for all seven benchmarks. To visualize the distribution of costs, Figure 3.15 shows a series of box plots of the solution costs returned for benchmark 7. This benchmark is the most difficult one within our experiments, as the object has to be grasped and placed at least two times and the robot must go through two tunnels while picking from and placing into the box. As one can see, not only the average cost but also its variance is reduced as the parameter settings are increased.

The average times for roadmap construction and query needed to achieve these results are depicted in Figure 3.16. It can be seen that highly reliable and close to optimal planning is possible with query times below one second.

To compare the solution quality of our approach to that of the Probabilistic Tree of Roadmaps (PTR) planner, Table 3.2 shows the results of both approaches across our benchmark tasks. The table depicts the average cost and the standard error of the mean (in brackets) for both planners, with and without post-processing. Our planner is run with the maximum settings from the previous experiments: $N_c = 250$, $N_i = 2500n$, $N_t = 25$. Both planners were run 30 times.

We analyzed the resulting data-set under the assumption that our samples follow independent Gaussian-distributions with unequal variance. Significance levels were therefore computed via Welch's unequal variances t-test. Two observations can be made: RMR* significantly (at 0.1% level) outperforms PTR in all seven tasks, both with and without post-processing. Furthermore the post-processing significantly (also at 0.1% level) improves the solution cost of our planner. This second observation shows that our planner has not fully converged to an optimal path even at its highest settings. This result is not surprising, as 2500 nodes for the PRM* cannot be expected to sufficiently explore the 7-dimensional

Table 3.2: Average cost and standard error of the mean

	Benchmark						
	1	2	3	4	5	6	7
PTR	**15.2**	**64.3**	**19.4**	**78.1**	**15.5**	**53.8**	**70.2**
	[1.14]	[3.64]	[1.86]	[4.17]	[1.04]	[3.12]	[3.80]
PTR with	**14.6**	**56.6**	**17.4**	**69.7**	**14.9**	**49.3**	**63.6**
post-processing	[1.07]	[3.16]	[1.65]	[3.48]	[1.04]	[3.23]	[3.54]
RMR*	**9.8**	**23.8**	**12.1**	**25.3**	**10.6**	**24.3**	**25.8**
	[0.07]	[0.18]	[0.07]	[0.23]	[0.10]	[0.23]	[0.20]
RMR* with	**8.6**	**20.3**	**10.6**	**21.1**	**9.1**	**20.6**	**21.5**
post-processing	[0.04]	[0.11]	[0.06]	[0.22]	[0.07]	[0.21]	[0.14]

configuration space of the robot. To visualize the distribution of solution costs of both planners at all seven benchmarks, Figure 3.17 shows the corresponding box plots. As can be seen, RMR* produces solutions of higher quality with much lower variance in cost.

Finally, we implemented our approach on a real robot. The manipulation sequence shown in Figure 3.1 is a solution path of RMR*. This demonstrates that the proposed approach is applicable to realistic manipulation scenarios.

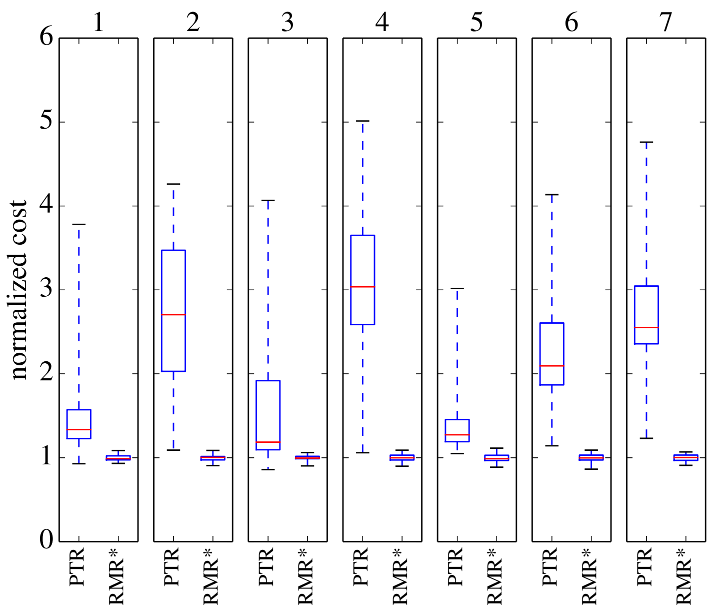


Figure 3.17: Comparison of PTR and RMR*: Each box plot depicts the distribution of normalized cost without post-processing for a different benchmark. Costs are normalized to the average cost returned by our planner. The red line depicts the median. The first and third quartiles are represented by the box, minimum and maximum by the whiskers.

3.5 Related Work

Our work integrates and extends two strands within the planning literature: optimal, sampling-based motion planning and manipulation planning.

3.5.1 Optimal, Sampling-Based Motion Planning

Planning for robotic manipulators typically involves high-dimensional configuration spaces. Sampling-based motion planning has shown good empirical performance in this setting and is the state-of-the-art paradigm for motion planning. Typical planners include RRTs [5] or PRMs [4]. Karaman and Frazzoli [13] provide a proof of the sub-optimality of these approaches as well as asymptotically optimal counterparts of the original planners: RRT* and PRM*.

For these planners several improvements and extensions have been proposed. To speed up convergence, heuristic guidance during sampling is introduced by Gammell *et al.* [23]. Hauser [24] uses lazy collision checking to defer expensive computations until a better path has been found. Optimal, sampling-based, kinodynamic planning is addressed by Karaman and Frazzoli [25]. An extension of the original RRT* for constrained motion is presented by Jaillet and Porta [26].

These optimal motion planners are not suitable for manipulation due to the so called crossed foliation issue [9]. Manipulation involves motion on a potentially infinite number of constraint manifolds that cannot be meaningfully explored with the previously discussed planners. We do however use the PRM* algorithm [13] as a subroutine of our planner.

An alternative view on optimal planning is proposed by Hauser and Zhou [27]. Instead of planning in a state space one can also plan in a state-cost space. This augmented planning problem is always a kinodynamic planning problem as cost is essentially underactuated. By repeatedly using a kinodynamic planner, e. g., the EST [6], in state-cost space one can then obtain optimal motions in the limit of repetitions. This approach is not feasible for manipulation, as no probabilistically complete kinodynamic planner has been proposed to explore the constrained configuration spaces of manipulation.

3.5.2 Manipulation Planning

The first approach to manipulation planning was presented by Alami *et al.* [8] and considers the planning problem in which an object is assumed to be either at a stable placement or grasp. Placements and grasps are chosen from a finite set. Manipulation is then formulated as a graph search with alternating transit and transfer paths, for which individual motion planning queries must be solved. Continuous grasps and placements are addressed by Simeon *et al.* [10]. This is done by modeling the connected components within the intersection of stable placements and grasps as closed-chain systems.

Typically, manipulation planners use motion planners as subroutines. This is problematic when sampling-based planners are used, as these planners are only probabilistically complete. These planners cannot decide within finite time if a planning query is infeasible. The incompleteness of sampling-based motion planners is addressed by Hauser and Latombe [28] by building a roadmap of roadmaps with the Multi-Modal-PRM (MM-PRM). This idea is then extended to the Probabilistic Tree of Roadmaps (PTR) planner [11] to also address continuous contact states. An RRT-like planner for the same problem class is proposed by Hauser and Ng-Thow-Hing [29].

To address a wider class of planning problems, Cambon *et al.* [30] propose a hybrid planner to integrate task and motion planning. Dornhege *et al.* [31] propose an extension to the PDDL standard [32] to combine symbolic and motion planning via semantic attachments. Garret *et al.* [33] reformulate heuristics from the symbolic planning domain [34] for task and motion planning with very long running tasks. In [35], a second heuristic guided planner is proposed, that is capable of handling continuous contact states and the incompleteness of sampling based motion planning.

The approaches discussed so far are capable of solving a variety of manipulation planning problems but do not compute optimal motions. Heuristic methods to improve the quality of planned trajectories have been proposed by Harada *et al.* [36] and by Zhang and Shah [37].

For special cases of manipulation, optimal planners have been proposed. When only planar tabletop rearrangement with overhand grasps and fixed placement positions is considered, manipulation planning can be reduced to a discrete graph search. Han *et al.* [12] provide an optimal planner for this scenario and analyze the complexity of this planning problem. For manipulation problems that allow a factorization of the configuration space, Vega-Brown and Roy [38] propose an asymptotically optimal manipulation planner, the Factored Orbital Bellman Tree (FOBT). This factorization is possible, e. g., for a mobile robot that pushes objects in a plane. It is however not applicable to manipulation with articulated robots.

The planner we proposed in this chapter is closest related to the MM-PRM [28] and FOBT [38] planners. Similar to FOBT we use a factored sampling strategy but use explicit sampling of transitions like with MM-PRM. To connect the samples of our roadmap we use the connection mechanism of the PRM* planner [13]. The combination of factored sampling and explicit transitions allows our approach to compute optimal motions for manipulation problems with articulated and redundant robots.

Another approach to optimal manipulation planning is the logic geometric programming by Toussaint [39]. However, this approach requires to model tasks exclusively via differentiable constraints. This type of model is incompatible with the black box model of collisions and other constraints employed in most of the literature and this chapter and makes a comparison difficult.

3.6 Discussion

This chapter presented an asymptotically optimal manipulation planner. We established convergence under a set of new robustness conditions and validated the practicality of our approach in extensive simulations and on a real industrial manipulator.

The proposed planner is capable of returning high quality solutions to complex manipulation tasks in less than a second. This is achieved without relying on problem specific heuristics or simplifications as our algorithm directly works with the primitive operations common to manipulation.

The scope of our approach is limited to prehensile manipulation of one single object. Promising areas for future research include extending the approach to new problem domains, as well as methods to increase the speed of convergence for larger problems.

Like the original PRM algorithm, the proposed planner leverages a preprocessing of the geometry of the environment. This offline preprocessing is time consuming, but allows online queries to be solved efficiently and reliably. Promising areas of application are in industrial manipulation tasks, such as machine tending. A robot could feed parts that are detected online with a camera to a CNC machine. In this scenario the environment of the robot is static and only the positions of robot and object differ between queries.

When the environment or the shape of the manipulated object changes between queries, the preprocessing turns into a disadvantage. The planner will explore areas of the configuration space that are not meaningful to the task. In the following chapters we therefore develop algorithms for manipulation planning that do not require a preprocessing phase.

3.7 Relation to a Previous Publication by the Author

The results of this chapter have been previously published within conference proceedings:

Optimal, Sampling-Based Manipulation Planning
P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G.v. Wichert, W. Burgard
2017 IEEE International Conference on Robotics and Automation
© IEEE 2017

The author of this thesis is the main contributing author of this previous publication and its contents were developed in pursuit of this thesis. For this reason, large parts of this chapter are identical to this previous publication. To improve readability we omitted citations of this publication throughout this thesis. The content has been modified and extended within this thesis.

Changes Relative to the Conference Proceedings Besides changes in wording and formatting most of this chapter is identical to the previous publication. Larger changes are listed in the following.

- Change of notation: To achieve a consistent notation across this thesis, we modified the problem statement, descriptions of the algorithms, and the mathematical analysis to follow the notation of this thesis.
- Changes to the problem statement and structure of the proof of asymptotic optimality: The definitions of Section 3.1.1 and proofs of Section 3.3 use the infinum of cost instead of the minimum.
- Detailed visualization of the algorithms: The illustrations of Section 3.2.3 were added.
- Related work: The contents of Section 3.5 were modified to follow the citation style of this thesis and additional references were added.

Contributions of Coauthors The previous publication was written together with Werner Neubauer, Wendelin Feiten, Kai Wurm, Georg v. Wichert, and Wolfram Burgard.

- Werner Neubauer provided consulting with respect to the motion planning aspects of this work. He recommended the inclusion of lazy collision checking into the algorithm.

- Wendelin Feiten provided consulting with respect to the mathematical analysis and the structure of the assumptions and proofs made in Section 3.3.
- Kai Wurm revised the paper and gave guidance on scientific writing.
- Georg v. Wichert and Wolfram Burgard provided valuable advice.

Chapter 4

Planning and Controlling Manipulation in Dynamic Environments

In the previous chapter we considered geometric manipulation planning, where a robot grasps and places a single object. We took a standard view on manipulation planning: A system is assumed to be in one of two contact states: The object is either rigidly attached to the robot or to the environment. In each of these contact states, motions of the robot can be planned with standard, geometric motion planners.

While this view is suitable for many tasks, such as machine tending, there exist a variety of scenarios to which it is not applicable. In the following we discuss a set of challenges in relevant manipulation tasks that break assumptions of this view:

- **Task specific constraints:** Some manipulation tasks constrain the motion of the robot due to physics or due to additional constraints specified by a user. Consider the task depicted in Figure 4.1. A mobile manipulator opens a door. While the robot opens the door there are two closed kinematic chains. One is formed by the robot and the door as the door has only one degree of freedom. The second chain is formed due to constraints specified by a user: The head mounted camera must point at the door while it is held by the robot. To address these constraints, we need models that can capture them and algorithms that solve constrained manipulation tasks.

- **Time-variance and dynamics:** Some manipulation tasks occur in time-variant environments. A typical example is a conveyor belt that moves parts independently of the robot as shown in Figure 4.2. Time-variance must be addressed both in the models of a planning problem and in the planner itself. Furthermore, time-variance requires that the dynamics of the robot are considered during planning. It is not possible to plan a geometric trajectory first and add a time parametrization before execution.

Most of the results of this chapter have been previously published by the author in conference proceedings. For improved readability we omit citations of these previous publications throughout this thesis. Instead, the relation between this chapter and the previous publications is discussed in detail in Section 4.7.

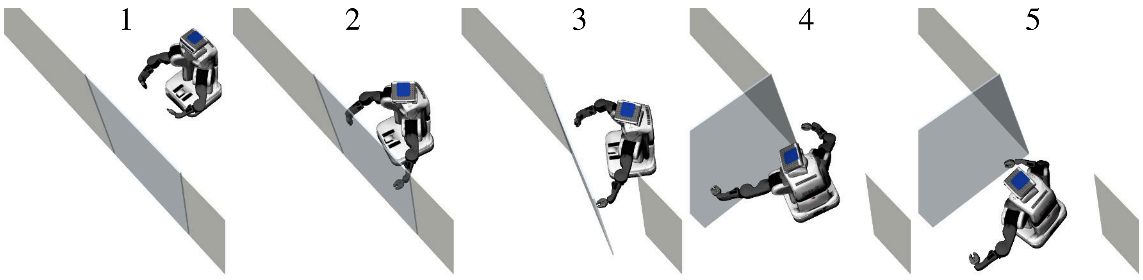


Figure 4.1: Constrained manipulation task: A mobile manipulator opens a door. (1) Initially, the robot approaches the door and (2) grasps it. (3) While the robot manipulates the door, robot and door form a closed kinematic chain. Additionally, the head mounted camera must point at the door while the robot manipulates it. (4) Once the door is opened the robot releases it and (5) proceeds through it.

- **Real-time reactions to changes in the environment:** Models of robots and estimates of environments are inherently inaccurate. Planning trajectories and executing them open-loop is therefore likely to fail eventually. In some scenarios, environments may change dynamically, e. g., a human might enter the workspace of a robot. Hence, robots should react in real-time to estimates of their environments, such as measurements of object poses and detected obstacles.

Previous work has partially addressed these challenges. Constraint-based task specification and control [41] allows to model complex and constrained robotic motion and to

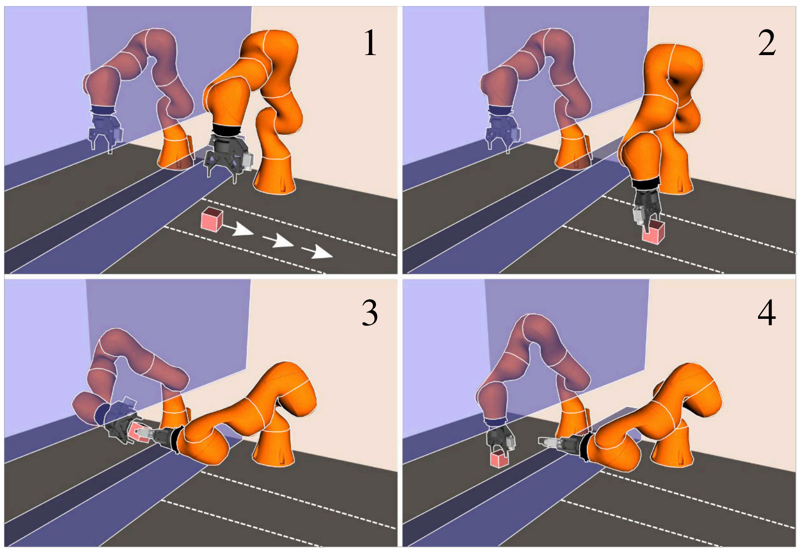


Figure 4.2: Manipulation in a dynamic environment: The goal of the robot is to place the cube on the left side of the blue wall. The problem is time-variant as the cube is initially moving to the right on a conveyor belt. In order to achieve the goal, the right manipulator must grasp the cube and hand it to the left manipulator through an opening in the wall.

control it in real-time. However, the sequential interdependencies of manipulation are not resolved. Mirabel and Lamiraux propose the constraint graph [9] as a generic model for complex manipulation tasks and a corresponding geometric manipulation planner. To the best of our knowledge no previous approach addresses all of the discussed challenges in the context of sequential manipulation tasks.

The key insight of this chapter is that a suitable model structure enables planning and control of complex, sequential manipulation tasks. As our first contribution we propose a new model for manipulation tasks: the dynamic constraint graph. This model extends the constraint graph [9] to second-order dynamics and time-variance. From this model we automatically derive a set of motion controllers. Examples include controllers that pick or place objects or that open or close a door. Sequential manipulation tasks can then be solved by switching between these controllers. As our second contribution we propose two methods to implement such a switching control scheme. The first method formulates the choice of the active controller as a kinodynamic planning problem. The second method uses reinforcement learning to switch between controllers. We implemented the proposed methods for three distinct robots and tasks. In real-world experiments, our approach enables manipulation with online collision avoidance and reactions to estimates of object poses.

4.1 The Dynamic Constraint Graph

We specify manipulation tasks in a modular way that facilitates planning and reactive execution. In this section we introduce a new model: the dynamic constraint graph. This model extends the constraint graph introduced by Mirabel and Lamiraux [9] to incorporate second-order dynamics and time-variance. We use the setup and the corresponding manipulation task depicted in Figure 4.2 as an exemplary model instance.

A **configuration** $\mathbf{q} \in \mathbb{R}^n$ and its time derivative $\dot{\mathbf{q}}$ encode the continuous state of a system. In the example this vector $\mathbf{q}$ comprises the 14 axis positions as well as a vector-quaternion representation for the pose of the cube. Time is represented by t.

A **discrete mode** $\sigma \in \Sigma$ encodes the contact state of a system and is an element of the finite set of modes Σ. In the example the cube may either be at rest on a surface, moved by a conveyor, or held by one of two grippers. Since the cube has six sides that may be in contact with surface, conveyor, or grippers the example features $|\Sigma| = 24$ discrete modes. **The discrete mode determines both the system dynamics and the constraints that a configuration q must fulfill.**

A system is governed by the control input $\mathbf{u} \in \mathbb{R}^{n_\mathrm{u}}$. In the example $\mathbf{u}$ comprises 14 axis-accelerations. Given a mode σ, a system follows a control-affine, second-order dynamic:

$$\ddot{\mathbf{q}} = \mathbf{a}_\sigma(\mathbf{q}, \dot{\mathbf{q}}, t) + B_\sigma(\mathbf{q}, t)\mathbf{u}, \tag{4.1}$$

where $\mathbf{a}_\sigma(\cdot)$ denotes the acceleration $\ddot{\mathbf{q}}$ for zero control input and $B_\sigma(\cdot)$ denotes its affine dependency on $\mathbf{u}$. In the example, the acceleration of the 14 axes is equal to $\mathbf{u}$, independently of σ. The acceleration of the cube depends on the mode σ and the configuration of the robot. In case of a grasp, the acceleration of its pose is determined by the forward kinematics of the grasping robot. When the cube is placed on a surface its acceleration is zero. If an object is placed on a conveyor belt, Eq. (4.1) is time-dependent.

Naturally, the controls $\mathbf{u}$ must be bounded to prevent damage to the system. We assume that the bounds on controls are given as a symmetric and affine inequality:

$$-\mathbf{k}_{\max} \leq \mathbf{c}_\sigma(\mathbf{q}, \dot{\mathbf{q}}, t) + D_\sigma(\mathbf{q}, t)\mathbf{u} \leq \mathbf{k}_{\max}. \tag{4.2}$$

Again, $\mathbf{c}_\sigma$ denotes a control-independent term and D_σ an affine dependency on $\mathbf{u}$. The sum of these terms must fulfill the box constraint defined by $\mathbf{k}_{\max}$. These affine limits on controls enable to model torque limits. However, in the example we assume a conservative limit on axis accelerations $-\mathbf{u}_{\max} \leq \mathbf{u} \leq \mathbf{u}_{\max}$.

Each mode σ is associated with a set of piecewise twice differentiable constraints $\mathbf{f}_\sigma$ and $\mathbf{g}_\sigma$, that must hold while the system is in that mode or to transition into that mode:

$$\mathbf{f}_\sigma(\mathbf{q}, t) = \mathbf{0}, \qquad \mathbf{g}_\sigma(\mathbf{q}, t) \leq \mathbf{0}. \tag{4.3}$$

In the example the inequality constraints $\mathbf{g}_\sigma$ comprise axis limits as well as collision avoidance. The geometry of the robots and cube is approximated by convex primitives and their pairwise penetration depth must remain negative. Equality constraints $\mathbf{f}_\sigma$ arise due to the contact state of the object. If the cube is placed, its center must be at a fixed height above the surface and its contacting side be anti-parallel to the surface. This also implies that the pose of the cube is under-constrained: two translational axes and one rotational axis are free.

Additionally, constraints may be imposed at the velocity level:

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_\sigma(\mathbf{q}, t) = \mathbf{0}, \qquad \tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{w}_\sigma(\mathbf{q}, t) \leq \mathbf{0}. \tag{4.4}$$

Axis velocity limits are given as inequality constraints. The relative velocity of the cube to the surface or a gripper must be zero when placed or grasped respectively. While the system is in mode σ the constraints $\mathbf{f}_\sigma$, $\mathbf{g}_\sigma$, $\mathbf{v}_\sigma$, and $\mathbf{w}_\sigma$ must be fulfilled.

The **full state** of a system $x = (\sigma, \mathbf{q}, \dot{\mathbf{q}}, t) \in \mathcal{X}$ comprises the current mode, the configuration, its velocity, and time. We assume that this state is known to the system. For a mode σ we denote the corresponding set of valid states $\mathcal{V}_\sigma$ as the set of states, in which the system is in that mode and Eq. (4.3) and Eq. (4.4) are fulfilled. Figure 4.3 visualizes these sets for the example task.

Following the ideas of [9], the modes $\sigma \in \Sigma$ form a graph. To transition from a mode σ_1 in this graph to mode σ_2, the second mode must be in the first mode's **set of neighbors**: $\sigma_2 \in \text{Neighbors}(\sigma_1)$. Additionally, the constraints $\mathbf{f}_{\sigma_2}$, $\mathbf{g}_{\sigma_2}$, $\mathbf{v}_{\sigma_2}$, and $\mathbf{w}_{\sigma_2}$ of σ_2

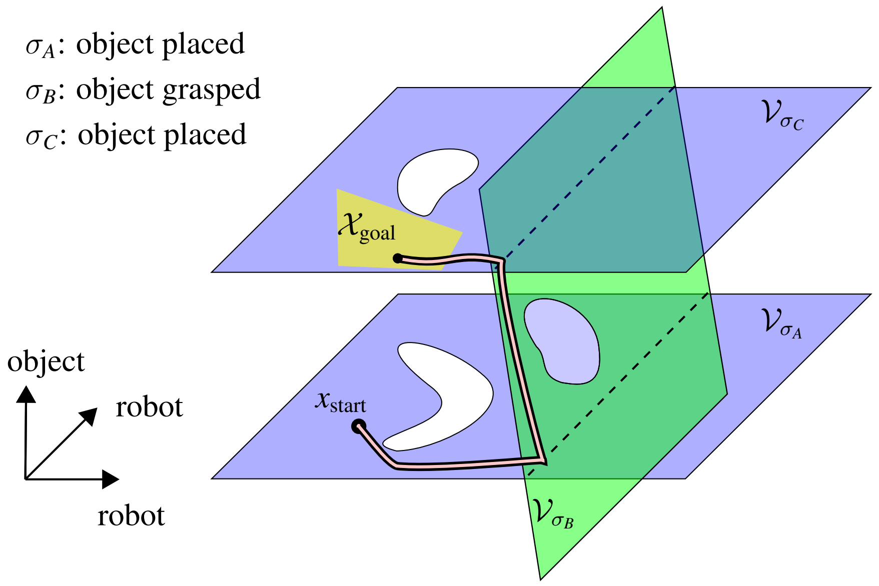


Figure 4.3: Model of a pick and place task: Three planes visualize $\mathcal{V}_\sigma$ for different modes. White areas represent configurations that are in collision. Initially the object is placed (mode σ_A) and the object does not move when the robot moves. At the intersection of $\mathcal{V}_{\sigma_A}$ and $\mathcal{V}_{\sigma_B}$ the object can be grasped. When grasped (mode σ_B) the object is moved along with the robot. Within mode σ_C the object is placed again in a different orientation. The thick line represents a trajectory for robot and object that leads from a start state x_{start} to a goal region $\mathcal{X}_{\text{goal}}$.

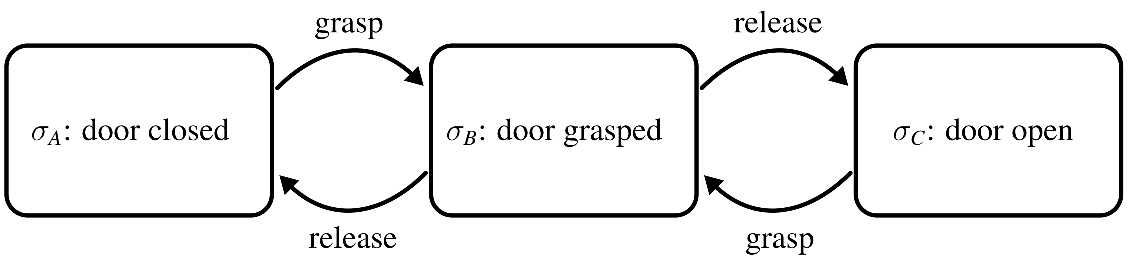


Figure 4.4: Exemplary dynamic constraint graph for the task of Figure 4.1: In mode σ_A the door is closed, which is the only constraint specific to that mode. Some constraints are shared across all modes. This includes collision avoidance and axis limits. In mode σ_B the robot is holding the door and the camera must point at the grasp point. Mode σ_C constrains the door to be open.

must be fulfilled. For example, to grasp a resting cube the gripper must be positioned in a grasping pose relative to the cube and all constraints on axis limits and distances of collision bodies must be fulfilled. Switching a mode, i. e., opening or closing grippers in the example, is assumed to happen instantaneously and is an explicit decision of the robot and not part of the system dynamics.

Mirabel and Lamiraux [9] show that this graph, described by the set of modes Σ, the neighborhood-function Neighbors$(\cdot)$, and the corresponding constraints, can be constructed automatically for typical manipulation tasks. Some of the constraints of Eq. (4.3) and Eq. (4.4) of the current mode are automatically fulfilled by the system dynamics Eq. (4.1) similar to the explicit constraints of [42]. For example, while an object is grasped the constraints on the relative velocity of object and gripper are automatically fulfilled by the forward kinematics. Figure 4.4 shows the graph structure for the example of Figure 4.1.

We assume that the system is at an initial state $x_{\text{start}} = (\sigma_{\text{start}}, \mathbf{q}_{\text{start}}, \dot{\mathbf{q}}_{\text{start}}, t_{\text{start}})$ and should reach a set of goal states $\mathcal{X}_{\text{goal}} \subset \mathcal{X}$. The goal for a manipulation planner is to find a trajectory of states x and controls $\mathbf{u}$ that has the following properties:

- The trajectory begins in x_{start} and ends within $\mathcal{X}_{\text{goal}}$.
- There is a finite number of mode switches.
- A mode σ^- may only change to another mode σ^+ if they are neighbors in the dynamic constraint graph, i. e., $\sigma^+ \in \text{Neighbors}(\sigma^-)$.
- The dynamics of Eq. (4.1) are fulfilled and the limits on the control input $\mathbf{u}$ of Eq. (4.2) are fulfilled.
- At all times the constraints of Eq. (4.3) and Eq. (4.4) are fulfilled. That is, for the current state x with mode σ we require $x \in \mathcal{V}_\sigma$.

Figure 4.3 shows such a trajectory of states x as the thick line. In this chapter we address reactive manipulation. For this reason, we aim to instantiate such a trajectory with a controller $\mathbf{u} = \mathbf{k}(x)$.

4.2 Automatic Controller Synthesis

We can use the dynamic constraint graph to automatically construct motion controllers. These motion controllers implement a movement of the robot that is goal-directed with respect to the planning problem.

We construct two types of controllers. The first type steers the system towards the intersection of the constraint manifolds of two modes, i. e., it attempts a mode change. Examples include controllers that open or close a door or that pick or place an object. The second type of controller steers the system towards a desired robot configuration. This

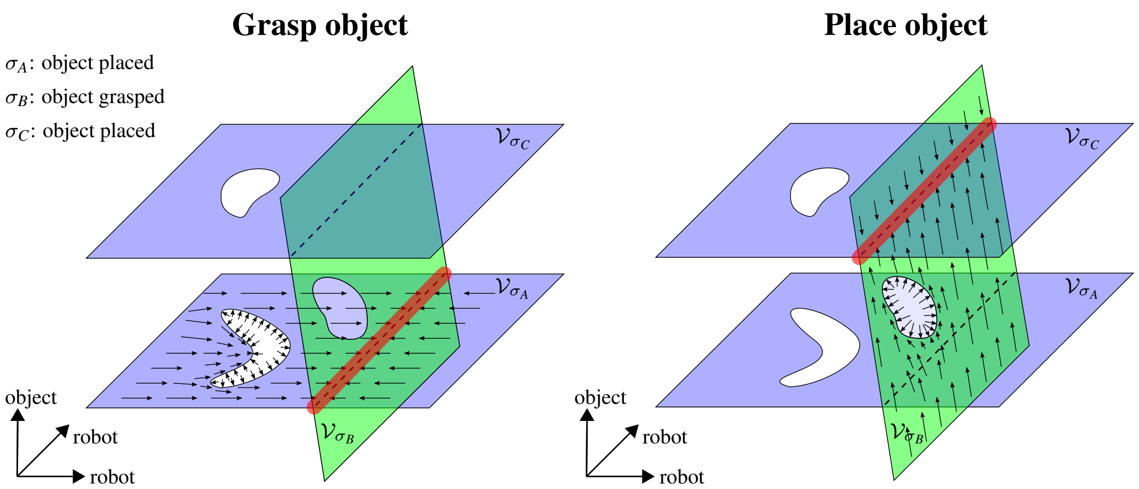


Figure 4.5: Phase diagrams for mode change controllers: The left image shows a phase diagram of a controller that attempts to grasp an object. In the right image a phase diagram is shown for placing the object in a different orientation. The controllers are not required to converge globally to their goal. This can be seen in the left image as some trajectories get stuck in the C-shaped obstacle.

type of controller is intended to explore the collision free areas of the configuration space. With these two types of controllers we can formulate manipulation planning as switching between motion controllers. In the following we explain how the two types of controllers can be constructed and their resulting properties. The controllers in this section are based on the eTC controller [43] that we modified to an acceleration resolved control scheme similar to [44].

4.2.1 Mode Change Controller

We first explain how a mode change controller operates. Let us assume our system is currently at a valid state $x = (\sigma, \mathbf{q}, \dot{\mathbf{q}}, t)$ and should transition to a mode $\sigma' \in \text{Neighbors}(\sigma)$. Thus, controls $\mathbf{u}$ need to be computed that eventually lead to a state fulfilling the constraints of mode σ'. Until these constraints are fulfilled the constraints of the current mode σ must not be violated. Figure 4.5 shows phase diagrams for two such controllers.

Let us recall that a mode σ defines the constraint on positions

$$\mathbf{f}_\sigma(\mathbf{q}, t) = \mathbf{0}, \qquad \mathbf{g}_\sigma(\mathbf{q}, t) \leq \mathbf{0},$$

and on velocities

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_\sigma(\mathbf{q}, t) = \mathbf{0}, \qquad \tfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{w}_\sigma(\mathbf{q}, t) \leq \mathbf{0}.$$

In the following we omit the equality constraints $\mathbf{f}_\sigma$ and $\mathbf{v}_\sigma$ as they can be represented as two inequalities with different signs in the computations. We can implement a mode change with controls $\mathbf{u}$ that result in the following desired dynamics of the constraint functions:

$$\begin{aligned}
\ddot{\mathbf{g}}_\sigma &\leq -K^g_\sigma \mathbf{g}_\sigma - D^g_\sigma \dot{\mathbf{g}}_\sigma, \\
\ddot{\mathbf{w}}_\sigma &\leq -D^w_\sigma \dot{\mathbf{w}}_\sigma, \\
\ddot{\mathbf{g}}_{\sigma'} &\leq -K^g_{\sigma'} \mathbf{g}_{\sigma'} - D^g_{\sigma'} \dot{\mathbf{g}}_{\sigma'}, \\
\ddot{\mathbf{w}}_{\sigma'} &\leq -D^w_{\sigma'} \dot{\mathbf{w}}_{\sigma'},
\end{aligned}$$

with $\mathbf{g}_*$, $\dot{\mathbf{g}}_*$, and $\ddot{\mathbf{g}}_*$ being the value and time-derivatives of $\mathbf{g}_*$ for given $\mathbf{q}$, $\dot{\mathbf{q}}$, t and $\mathbf{u}$. The matrices K^g_*, D^g_*, and K^w_* are diagonal and chosen to achieve stable and at least critically damped dynamics. If each vector element of the constraint functions follows its own decoupled, stable, and properly damped behavior, eventually all constraints will converge to or stay below zero. As the initial state x is already within $\mathcal{V}_\sigma$ it will stay within $\mathcal{V}_\sigma$.

The acceleration of the constraint functions can be linearized via:

$$\ddot{\mathbf{g}}_\sigma = \ddot{\mathbf{g}}_{\sigma,0} + \frac{\partial \ddot{\mathbf{g}}_\sigma}{\partial \mathbf{u}}\mathbf{u}, \qquad \ddot{\mathbf{w}}_\sigma = \ddot{\mathbf{w}}_{\sigma,0} + \frac{\partial \ddot{\mathbf{w}}_\sigma}{\partial \mathbf{u}}\mathbf{u}.$$

Here, $\ddot{\mathbf{g}}_{*,0}$ denotes the second-order time-derivative of $\mathbf{g}_*$ with zero controls $\mathbf{u} = \mathbf{0}$. Typically, the desired dynamics of the constraint functions of σ and σ' cannot be obtained at the same time. For this reason we relax the desired dynamics for the constraint functions of mode σ' with slack variables $\epsilon_{g'}$ and $\epsilon_{w'}$. This leads to the relaxed, linearized dynamics:

$$\begin{aligned}
\ddot{\mathbf{g}}_{\sigma,0} + \frac{\partial \ddot{\mathbf{g}}_\sigma}{\partial \mathbf{u}}\mathbf{u} &\leq -K^g_\sigma \mathbf{g}_\sigma - D^g_\sigma \dot{\mathbf{g}}_\sigma, \\
\ddot{\mathbf{w}}_{\sigma,0} + \frac{\partial \ddot{\mathbf{w}}_\sigma}{\partial \mathbf{u}}\mathbf{u} &\leq -D^w_\sigma \dot{\mathbf{w}}_\sigma, \\
\ddot{\mathbf{g}}_{\sigma',0} + \frac{\partial \ddot{\mathbf{g}}_{\sigma'}}{\partial \mathbf{u}}\mathbf{u} &\leq -K^g_{\sigma'} \mathbf{g}_{\sigma'} - D^g_{\sigma'} \dot{\mathbf{g}}_{\sigma'} + \epsilon_{g'}, \\
\ddot{\mathbf{w}}_{\sigma',0} + \frac{\partial \ddot{\mathbf{w}}_{\sigma'}}{\partial \mathbf{u}}\mathbf{u} &\leq -D^w_{\sigma'} \dot{\mathbf{w}}_{\sigma'} + \epsilon_{w'}.
\end{aligned} \tag{4.5}$$

The controls $\mathbf{u}$ are now computed via a quadratic program:

$$\begin{aligned}
&\underset{\mathbf{x}}{\text{minimize}} \quad \mathbf{x}^\top H \mathbf{x} \\
&\text{subject to} \quad \mathbf{b}_l \leq A\mathbf{x} \leq \mathbf{b}_u.
\end{aligned} \tag{4.6}$$

The optimization variable $\mathbf{x} = \left[\mathbf{u}^\top, \epsilon_{g'}^\top, \epsilon_{w'}^\top\right]^\top$ comprises the control input and slack variables. The matrix H is diagonal with positive weights for the accelerations and the slack variables. The vectors $\mathbf{b}_l$, $\mathbf{b}_u$ and the matrix A are derived from the relaxed, desired dynamics in Eq. (4.5) and the bounds on control inputs in Eq. (4.2).

This optimization ensures that the constraints of the current mode σ will not be violated. Remaining degrees of freedom are used to achieve the desired dynamics for mode σ' while also using as little controls as possible.

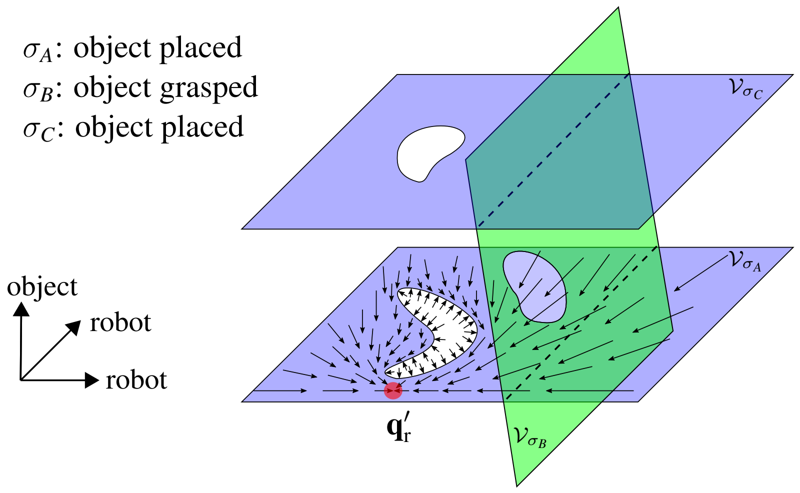


Figure 4.6: Phase diagram of a joint target controller: The goal for the controller is to reach a desired robot configuration $\mathbf{q}_\mathrm{r}'$ without leaving the set $\mathcal{V}_\sigma$ of the current mode σ.

We assume that a mode change controller performs a mode switch to the target mode σ' as soon as this is possible, i. e., as soon as the constraints of mode σ' are fulfilled within a numerical tolerance. Furthermore, we assume that after such a mode switch the task is either fulfilled or a different controller takes over. This is ensured with the planning algorithms we present in the following sections.

4.2.2 Joint Target Controller

The purpose of the joint target controller is to explore the collision free parts of the configuration space. For this purpose we separate the configuration $\mathbf{q}^\top = \left[\mathbf{q}_\mathrm{r}^\top, \mathbf{q}_\mathrm{o}^\top\right]$ into a robot configuration $\mathbf{q}_\mathrm{r}$ and an object configuration $\mathbf{q}_\mathrm{o}$.

Again, let us assume our system is currently at a valid state $x = (\sigma, \mathbf{q}, \dot{\mathbf{q}}, t)$. The goal of the joint target controller is now to steer the system to a desired robot configuration $\mathbf{q}_\mathrm{r}'$ without leaving the mode σ or violating its constraints. Figure 4.6 visualizes a phase diagram for such a controller.

This can again be implemented with controls $\mathbf{u}$ that result in the following desired dynamics of the constraint functions and the robot configuration $\mathbf{q}_\mathrm{r}$:

$$\begin{aligned}
\ddot{\mathbf{g}}_\sigma &\leq -K_\sigma^g \mathbf{g}_\sigma - D_\sigma^g \dot{\mathbf{g}}_\sigma, \\
\ddot{\mathbf{w}}_\sigma &\leq -D_\sigma^w \dot{\mathbf{w}}_\sigma, \\
\ddot{\mathbf{q}}_\mathrm{r} &= -K^\mathrm{r}(\mathbf{q}_\mathrm{r} - \mathbf{q}_\mathrm{r}') - D^\mathrm{r} \dot{\mathbf{q}}_\mathrm{r}.
\end{aligned}$$

These desired dynamics are again linearized and relaxed:

$$\begin{aligned}
\ddot{\mathbf{g}}_{\sigma,0} + \frac{\partial \ddot{\mathbf{g}}_{\sigma}}{\partial \mathbf{u}}\mathbf{u} &\leq -K^g_{\sigma}\mathbf{g}_{\sigma} - D^g_{\sigma}\dot{\mathbf{g}}_{\sigma}, \\
\ddot{\mathbf{w}}_{\sigma,0} + \frac{\partial \ddot{\mathbf{w}}_{\sigma}}{\partial \mathbf{u}}\mathbf{u} &\leq -D^w_{\sigma}\dot{\mathbf{w}}_{\sigma}, \\
\ddot{\mathbf{q}}_{\mathrm{r},0} + \frac{\partial \ddot{\mathbf{q}}_{\mathrm{r}}}{\partial \mathbf{u}}\mathbf{u} &= -K^{\mathrm{r}}(\mathbf{q}_{\mathrm{r}} - \mathbf{q}'_{\mathrm{r}}) - D^{\mathrm{r}}\dot{\mathbf{q}}_{\mathrm{r}} + \epsilon_{\mathrm{r}}.
\end{aligned}$$

The controls **u** are now computed using a similar quadratic program as with the mode change controller.

4.2.3 Discussion of the Controllers

The two kinds of controllers are used by a planner as so called steering functions or local planners. A planner can now explore the set $\mathcal{V}_\sigma$ of a mode σ using the joint target controller or attempt a mode switch with a mode change controller. Intuitively, the controllers operate similarly to the tangent space approach discussed in Section 2.5. Once a plan is found using the controllers one can store the sequence of controllers instead of the trajectory they instantiated during planning. This has the advantage, that during execution the controllers react in real-time to disturbances while respecting the constraints of the original planning problem.

We use a modified variant of the eTC controller [43] to construct the controllers. Other control frameworks, e. g., the Stack of Tasks [45], could be used as well. It is worth discussing a few technical issues that arise when using the dynamic constraint graph in combination with the presented controllers.

Stability: Even if the controllers exactly achieve the desired dynamics of the constraint functions, it is not guaranteed that the system remains stable. The reason is that the system may have redundant degrees of freedom that result in an unstable self-motion in the null-space of the constraint functions. This instability of acceleration resolved, local control schemes has been discussed by Suh and Hollerbach [46]. We address this by adding additional soft constraints to the optimization to achieve zero velocities on all robot axes. These additional constraints receive small weights so that the effect on the actual task of the controller, e. g., a mode switch, is small.

Overshoot after mode switches: For inequality position constraints, it is possible that the controllers overshoot the limit of the inequality after a mode change. This may happen if the corresponding constraint function has a large positive velocity shortly before the mode switch.

However, this is not a problem for typical manipulation tasks. The reason for this is that there are two structures of planning problems that prevent this and that are common to typical manipulation tasks. The first structure is that the same inequality constraints must be fulfilled both in the current and the following mode. Examples are axis limits and collision avoidance which must be fulfilled in all modes.

The second structure is that a position inequality constraint related to the task usually comes with a corresponding zero velocity equality constraint for the same constraint function. An example is the placement of an object on a surface. The x-position of the object during placement is usually constrained via a position inequality as we want to specify a region in which the object should be placed. This constraint could potentially be overshot. However, after the object is placed it does not move, which implies that there is an additional constraint to ensure that the velocity of the object is zero. For this reason it cannot overshoot the position constraint after the mode switch.

Feasibility of the quadratic program: The quadratic program in Eq. (4.6) that we use to compute the controls may be infeasible. This happens when the desired constraint dynamics of the current mode are in conflict.

If this happens, a straightforward approach is to relax the constraints of the current mode with high weights. Even if the desired dynamics of the constraint functions are not fulfilled in one control cycle, the system usually recovers without actual constraint violations in the following cycles. In the following sections we will present additional methods to address this in the proposed feedback planners.

4.3 Kinodynamic Feedback-Planning for Manipulation

In this section we present a kinodynamic feedback planner for manipulation. The idea is to transform the dynamic constraint graph into task specifications for motion controllers as described in the previous section. These controllers can then be simulated during planning as steering functions. A kinodynamic planner, similar to the Expansive Space Tree (EST) [6], uses these steering functions to build a search tree. As our planner extends the EST to constrained manipulation tasks and enables reactive execution it is called Constrained Reactive Expansive Space Tree (CR-EST).

4.3.1 Planning Algorithm

Before we describe our planner we introduce the primitive operations that are used to explore the configuration space. Procedure **randomController**(σ) $\rightarrow$ $\mathbf{k_q}$ returns a joint target controller $\mathbf{k_q}$ that steers towards a randomly chosen robot configuration on a trajectory that satisfies the constraints of mode σ. The procedure **simulateController**($x, \mathbf{k}$) $\rightarrow x_{\text{new}}$ takes a state x and controller $\mathbf{k}$ as input and returns the state x_{new} to which the system is moved by the controller.

Procedure **transitionController**(σ, σ') $\rightarrow \mathbf{k}_{\sigma'}$ takes modes σ and σ' as input and returns a mode change controller $\mathbf{k}_{\sigma'}$ that steers towards the intersection of both mode's constraint-manifolds while satisfying the constraints of σ. We assume that all controllers terminate if their targets are reached within a numerical tolerance and that the mode change controllers perform a mode switch when this happens.

The procedure **simulateController** simulates a controller until it reaches its target, i. e., the target mode or the random robot configuration. In this case the resulting state is returned. In case the optimization problem of equation Eq. (4.6) is infeasible the procedure returns failure. This may happen when the dynamics of safety constraints are in conflict. Controllers may also become stuck locally. To address this **simulateController** returns failure after a (simulated) time-out.

```
1  Procedure: buildSearchTree($x_{\text{start}}$) infinite version
2  $\mathcal{N} \leftarrow \{x_{\text{start}}\}$,    $\mathcal{E} \leftarrow \{\}$
3  while true do
4      $x \leftarrow (\sigma, q, \dot{q}, t) \leftarrow$ sampleWeighted($\mathcal{N}$)
5      $\mathbf{k_q} \leftarrow$ randomController($\sigma$)
6      $x_{\text{new}} \leftarrow$ simulateController($x, \mathbf{k_q}$)
7      $\mathcal{N}$.add($x_{\text{new}}$),    $\mathcal{E}$.add(($x, \mathbf{k_q}, x_{\text{new}}$))
8      for $\sigma' \in$ Neighbors($\sigma$) do
9          $\mathbf{k}_{\sigma'} \leftarrow$ transitionController($\sigma, \sigma'$)
10         $x_{\text{new}} \leftarrow$ simulateController($x, \mathbf{k}_{\sigma'}$)
11         $\mathcal{N}$.add($x_{\text{new}}$),    $\mathcal{E}$.add(($x, \mathbf{k}_{\sigma'}, x_{\text{new}}$))
```

With these procedures we can define our planner, CR-EST. Our algorithm builds a search-tree of nodes $\mathcal{N}$ and edges $\mathcal{E}$ rooted in the initial state x_{start}. The procedure **build-SearchTree** shows the pseudocode for the tree construction without termination condition. In each iteration the planner samples a random node **from the current set** (forward search!) of nodes $\mathcal{N}$ using the procedure **sampleWeighted**. This sampling puts larger weight on nodes that lie in sparsely populated areas of the state space. From this node we attempt to steer towards a random configuration, as well as to all neighboring modes. Naturally, the implemented planner has no infinite loop and the tree construction is stopped if a node within $\mathcal{X}_{\text{goal}}$ is found or a time budget is depleted. Figure 4.7 shows the construction of the search tree. If a controller gets stuck locally its result is discarded after a (simulated) time-out. Figure 4.7 shows this as the path that ends in a red cross.

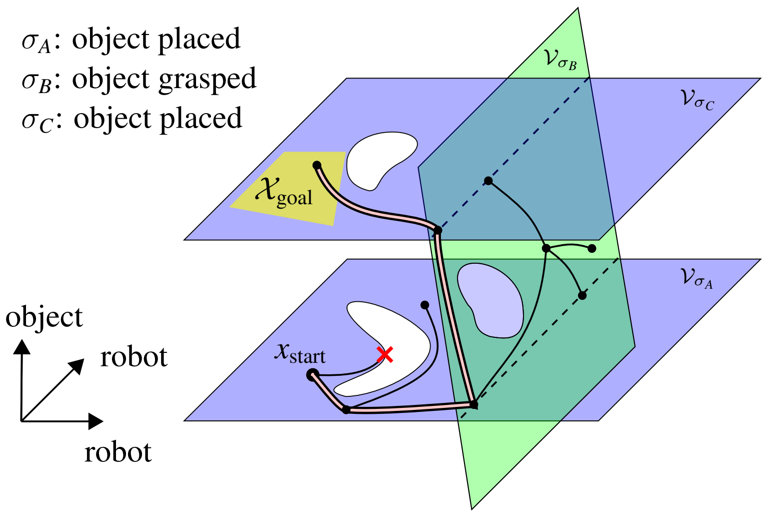


Figure 4.7: Exploration of the configuration space by CR-EST: The thick line represents a trajectory for robot and object that leads from a start state x_{start} to a goal region $\mathcal{X}_{\text{goal}}$. Thin black lines show alternative motions attempted by the planner as part of the search tree. If a controller gets stuck locally its result is discarded after a (simulated) time-out. This is shown as the black line that ends in a red cross.

4.3.2 Controlled Execution

The result of our planning algorithm is a sequence of constraint-based controllers. Figure 4.8 shows the phase portrait of such a controller sequence for a pick and place task. Using this sequence of controllers, an accurately modeled and undisturbed system will reach a goal state. If the execution is disturbed the controllers can meaningfully react. This is due to the fact that the parametrization of the controllers is based on the planning domain (safety constraints) and the decisions of the planner (non-safety constraints).

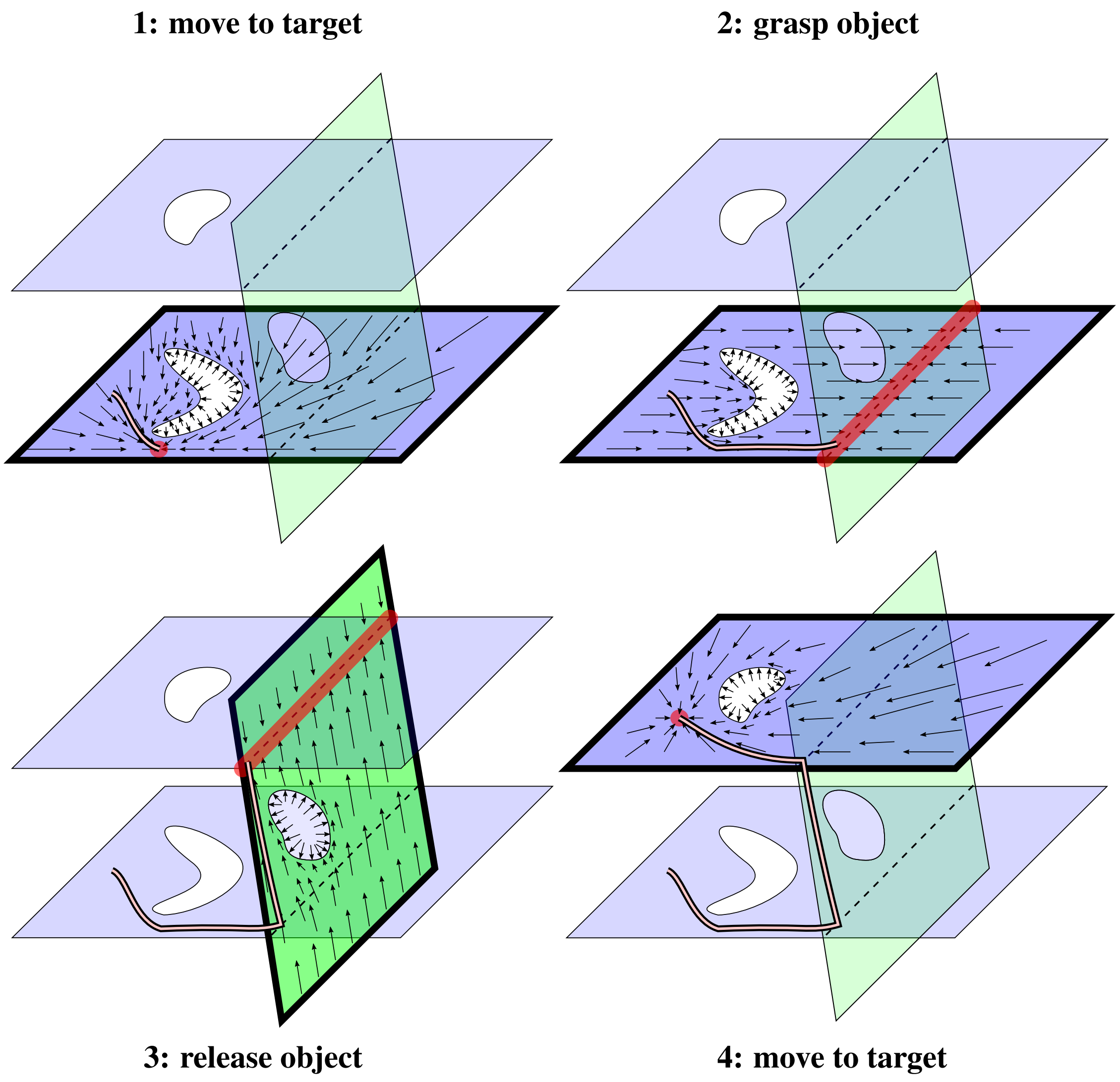


Figure 4.8: Phase-portrait of a controller sequence for a pick and place task: In step 1 the goal of the controller is to reach a target axis configuration. This is necessary for step 2, where the goal of the controller is to grasp the object by reaching the intersection of grasp mode and placement mode. If the grasping was attempted first, the system would get stuck in the C-shaped obstacle region. In step 3 the object is placed at a different placement position. Finally, in step 4, the robot retracts from the object.

4.3.3 Implementation Details

The definition of the constraints in Eq. (4.3) and Eq. (4.4), the system dynamics in Eq. (4.1), and the computation of all required derivatives is based on a custom implementation of the eTaSL/eTC framework [43]. For computations related to kinematics we use the Kinematics and Dynamics Library [47]. We solved the optimization problem inside the control loop of Eq. (4.6) using the qpOASES solver [48, 49]. All constraints share the same parameters for desired constraint dynamics K_*^* and D_*^*, assuming unit-free constraint functions. As units of measurement for our constraint functions we used seconds for time, radians for joint angles, meters for translations, and components of unit quaternions for rotatory Cartesian quantities. We chose elements of K_*^* as 20 and set D_*^* to achieve a damping ratio of 1.1. The weights for the matrix H where set to 0.001 for the control inputs and to 1.0 for the slack variables. We added weakly weighted (0.001) non-safety constraints to achieve zero axis velocities that prevent unnecessary motions and stabilize the system.

Most of the constraints, such as axis limits and collision constraints, are present in all modes of the constraint graph. For efficiency, these constraints are used only once as safety constraints in the computations. Constraints that are explicitly addressed by the system dynamics of Eq. (4.1) are also omitted in the computations, similar to the explicit constraints of Mirabel and Lamiraux [42].

The time-out parameter for the controller simulation was chosen as 10 (simulated) seconds. As control frequency we use 200 Hz during execution and 40 Hz during simulation to speed up planning. The procedure **sampleWeighted** uses a grid-based discretization of the state space typical for kinodynamic planners. As grid cells we use the mode of the state resulting in 24 cells in the experiments. This is a low number of cells compared to kinodynamic motion planners. However, this is compensated by the constraint-based controllers, that are comparatively powerful (and computationally expensive) steering functions that avoid collisions.

4.3.4 Experimental Setup

As experimental setup we use the dual-arm robot depicted in Figure 4.2. It comprises two seven-axis manipulators, each equipped with a parallel gripper. The goal is to move a cube that is placed on the right to the left side of the setup.

We designed five benchmarks to pose different challenges to manipulation planners. **Benchmark 1** comprises no additional obstacles, i. e., the walls of Figure 4.2 are missing and the left manipulator may directly pick and place the cube. **Benchmark 2** adds the lower obstacle and therefore makes a re-grasp necessary, where the right robot hands the object to the left one. **Benchmark 3** and **Benchmark 4** comprise both obstacles with a height of the opening set to 0.4 m and 0.2 m respectively. These two scenarios where added to stress the motion planning aspects of manipulation. **Benchmark 5** adds time-variance

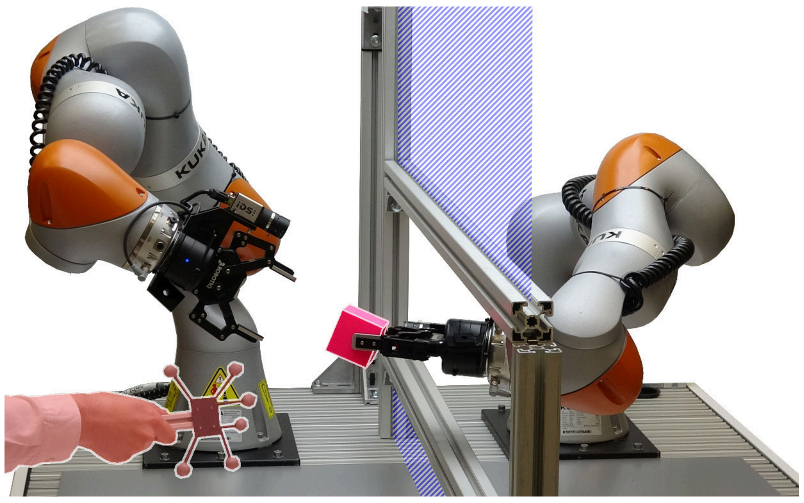


Figure 4.9: Online collision avoidance: While the robot is executing a plan, a person, that is tracked via markers, enters the workspace. The person's arm is approximated by a sphere and the robot reacts online to satisfy the collision-constraints. This is done while ensuring that no self-collisions occur. Remaining degrees of freedom are used to execute the plan.

by placing the cube on a conveyor belt as shown in Figure 4.2. At the beginning of Benchmark 5, the cube is not resting on the surface but moving away from the wall (0.2 m opening) with a constant velocity of $5 \, ^{\mathrm{cm}}\!/\!_{\mathrm{s}}$.

We implemented these benchmark scenarios for the planner presented in this chapter as well as for the RMR* algorithm of the previous chapter. RMR* is an asymptotically optimal, sampling-based manipulation planner. This allows to compare both planning times and costs of the resulting solutions. As the problem setting involves acceleration-controlled robots we implemented RMR* with the trajectory generators of [50, 51] as local planners. To allow a meaningful comparison of planning times, RMR* was implemented in an incremental fashion that does not separate between roadmap construction and query as presented in the previous chapter. Each benchmark is planned 70 times with different random seeds. The time variant Benchmark 5 was not implemented for RMR*. Both planners were implemented in C++, use multiple threads and were run on a ten-core Intel Xeon E5-2650v3.

On the real robot we implemented two qualitative experiments to validate the reactive execution of manipulation plans. The first experiment is designed to incorporate measurements of object poses into the reactive execution. A plan for Benchmark 4 is computed for an assumed pose of the cube. Then the actual pose of the cube is measured with a camera and the original plan is executed. In the second real-world experiment additional online collision-avoidance is added. While a plan is executed we use a tracking system to detect a person that must be avoided. Figure 4.9 shows this experiment for Benchmark 4. The avoidance of this obstacle must not lead to self-collisions or violations of axis limits.

Redundant degrees of freedom should be used to execute the manipulation plan.

4.3.5 Results

The average success rates of our planner over time are shown in Figure 4.10. CR-EST reliably solves the five benchmark problems. A surprising outcome is the consistency of planning times that appears to be independent of the difficulty of the benchmark.

Table 4.1 shows the average planning times and execution times of plans for both CR-EST and RMR*. RMR* shows the typical behavior of sampling-based, collision-free planners: planning times deteriorate as more obstacles and "tunnels" are added. CR-EST shows relatively consistent planning times across all benchmarks. We attribute this to the nature of the steering functions employed by our approach. As the constraint-based controllers receive the distances between collision bodies as inequality constraints, collisions simply do not occur, neither during execution nor during planning. This effectively implements a wall-following steering function that allows to efficiently solve the benchmarks with a tunnel in the configuration space (Benchmarks 3-5). However, simulating controllers is more time consuming than collision checking and RMR* plans faster when few obstacles are present.

Even though CR-EST does not attempt to compute optimal solutions, it is interesting to compare the quality of solutions to that of RMR*. RMR* was implemented to minimize the duration of solution trajectories. The first solution returned by RMR* however is arguably representative for typical sampling-based manipulation planners. Table 4.1 shows that solutions of CR-EST have longer durations than both the first solution returned by RMR* and the best solution after one minute of optimization. A possible explanation is that the linear target dynamic for the constraint-functions under-utilizes the axis limits on acceleration and velocity compared to the time-optimal local planner of [50]. In practice this means that with CR-EST the robot moves slower but smoother especially when close to obstacles.

In the real-world experiments the execution of plans reacted to measurements of object poses as well as obstacles that are detected online.The plans consisting of constraint-controllers encode not only the current target state but also the constraints of the planning domain such as axis limits and self-collision avoidance. Therefore, the robot does not only avoid the detected obstacle during execution, but does so in a way that no self-collisions occur. If redundant degrees of freedom remain, they are used to achieve the current target that is encoded in the plan.

However, it is possible to disturb the robot sufficiently so that the execution becomes stuck locally. This may happen if the controller is blocked by a combination of collision bodies or by being pushed into a kinematic reconfiguration where axis limits prevent the controllers to progress towards the goal. In practice this can be easily compensated by stopping and re-planning after a time-out.

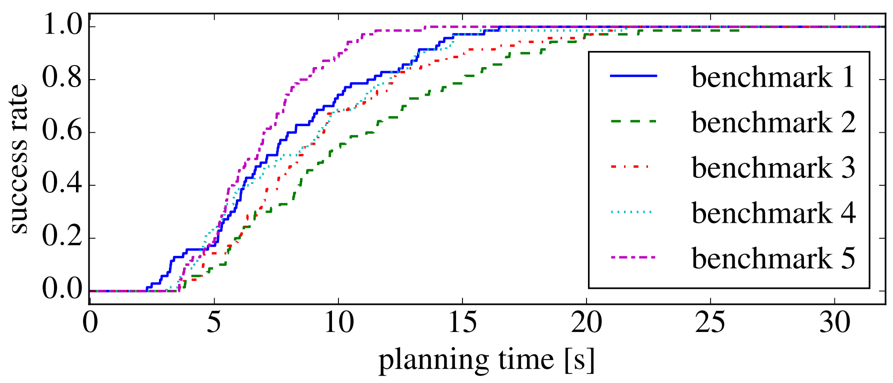


Figure 4.10: Success rates of CR-EST: Each line visualizes the success rate of 70 planning queries for one of the five benchmark tasks.

Table 4.1: Planning and Execution Times

all values in seconds		Benchmark				
[standard error of the mean]		1	2	3	4	5
CR-EST	planning time	**7.89**	**10.5**	**9.27**	**8.39**	**6.79**
		[0.43]	[0.59]	[0.50]	[0.46]	[0.26]
	execution time	**14.3**	**17.1**	**17.7**	**17.3**	**17.9**
		[0.37]	[0.36]	[0.20]	[0.09]	[0.14]
RMR*	planning time	**6.44**	**8.18**	**21.3**	**>300**	**/**
		[0.67]	[0.79]	[1.54]	/	/
	execution time	**10.7**	**11.8**	**14.9**	**/**	**/**
	of first solution	[0.34]	[0.44]	[0.57]	/	/
	best time at 60s	**5.76**	**5.90**	**7.19**	**/**	**/**

A useful property of our approach is that planning and controlled execution both function reasonably in invalid states. Examples include violations of axis limits as well as objects that are pushed into the collision margin of the robot. The controllers will steer towards valid states according to the target dynamics of the constraint-functions.

4.4 Feedback Planning for Manipulation via Q-Learning

The approach of the previous section uses a sampling-based planner to compute a sequence of motion controllers. As a consequence, it is not possible to obtain plans in deterministic time. Also, planning takes several seconds on average. This is problematic, e. g., when the environment contains fast, time-variant components, such as fast moving conveyor belts. For this reason we introduce a second feedback planner in this section, that selects the active controller in real-time. We achieve this by reformulating the problem as a reinforcement learning problem. A reinforcement learning agent selects, with a low control frequency, which motion controller to activate. Due to the hierarchical structure of this approach, we will refer to the motion controllers as low-level controllers and to the reinforcement learning agent as high-level controller.

4.4.1 Learning Algorithm

In this section we first formulate manipulation planning as a deterministic reinforcement learning problem and then approximate the solution to this problem using a Deep Q-Network (DQN) [52]. A full explanation of reinforcement learning or deep learning as a method for function approximation is out of the scope of this chapter. Sutton *et al.* [53] provide an overview of reinforcement learning methods and their theory. An introduction to deep learning is provided by Goodfellow *et al.* [54].

The core idea of this section is to reformulate the action space of a manipulation planner. Instead of computing controls $\mathbf{u}$ that are continuous across time, we select actions at discrete time instants with a cycle time of Δt. At each discrete time step we do not chose a control input but an active controller. Given a state $x = (\sigma, \mathbf{q}, \dot{\mathbf{q}}, t)$ and the controller synthesis in Section 4.2 there are two sets of possible controllers. There are $|\text{Neighbors}(\sigma)| < |\Sigma|$ applicable mode change controllers available and a continuous set of joint target controllers that steer to a robot configuration $\mathbf{q}_\text{r} \in \mathbb{R}^{n_\text{r}}$.

The idea is to create a finite set of such low-level controllers and then switch between them as needed. As the set of modes is finite, there exist only $|\Sigma|$ different mode switching controllers $\mathbf{k}_{\sigma'}$. However, a continuous set of controllers $\mathbf{k}_{\mathbf{q}'}$ with a configuration $\mathbf{q}'$ as target could be constructed. For this reason, we sample $N_\text{r} \in \mathbb{N}$ random configurations $\{\mathbf{q}_1, ..., \mathbf{q}_{N_\text{r}}\}$ as targets for N_r controllers. These random configurations do not need to be valid configurations. This leads to a finite set $\mathcal{K}$ of controllers with $|\mathcal{K}| = |\Sigma| + N_\mathrm{r}$:

$$\mathcal{K} = \{\mathbf{k}_1, ..., \mathbf{k}_{n_k}\} = \{\mathbf{k}_{\sigma_1}, ..., \mathbf{k}_{\sigma_{|\Sigma|}}, \quad \mathbf{k}_{\mathbf{q}_1}, ..., \mathbf{k}_{\mathbf{q}_{N_\mathrm{r}}}\}.$$

The high-level controller operates with a control cycle Δt, and should bring the system into a goal state within the shortest possible time. This results in a shortest path problem with a continuous state space, discrete time steps (Δt), and a discrete action set (which controller in $\mathcal{K}$ to activate).

Even though time and the action space are discretized this shortest path problem is challenging due to the continuous and high-dimensional configuration space. We address this by formulating the problem as a reinforcement learning problem in which each action yields a negative reward of -1 and termination occurs in the goal set $\mathcal{X}_\mathrm{goal}$. A policy is trained in simulation to form the high-level controller. Due to the continuous state space and finite action space, Deep Q-Networks [52] are a suitable learning method and were chosen for our approach. As input to this network we use an encoding $H(x)$. This encoding contains positions $\mathbf{q}$, velocities $\dot{\mathbf{q}}$, time t, and a one-hot encoding of the current mode σ. A DQN can now be trained in simulation for the given dynamic constraint graph an a goal set $\mathcal{X}_\mathrm{goal}$.

The action space of our reinforcement learning problem consists of motion controllers that are designed for constrained motion. As switching between these controllers is implemented with a Deep Q-Network we name our approach Constrained, Reactive Deep Q-Network (CR-DQN). The control flow of our architecture can be seen in Figure 4.11. A phase portrait of the approach is shown in Figure 4.12.

An implication of using only a finite number of random target controllers is that the approach may be incomplete for a planning problem given a choice of random targets. However, it is important to note that the random targets have a different purpose than random samples in a probabilistic roadmap planner [4]. Typically, the system cannot reach a random target within one cycle of the high-level controller (if it can reach it at all). Therefore, the samples serve as directions in configuration space along which the random target controllers may steer the system. This means that even with a low number of targets N_r, the high-level controller is typically able to move the system on a rich set of trajectories.

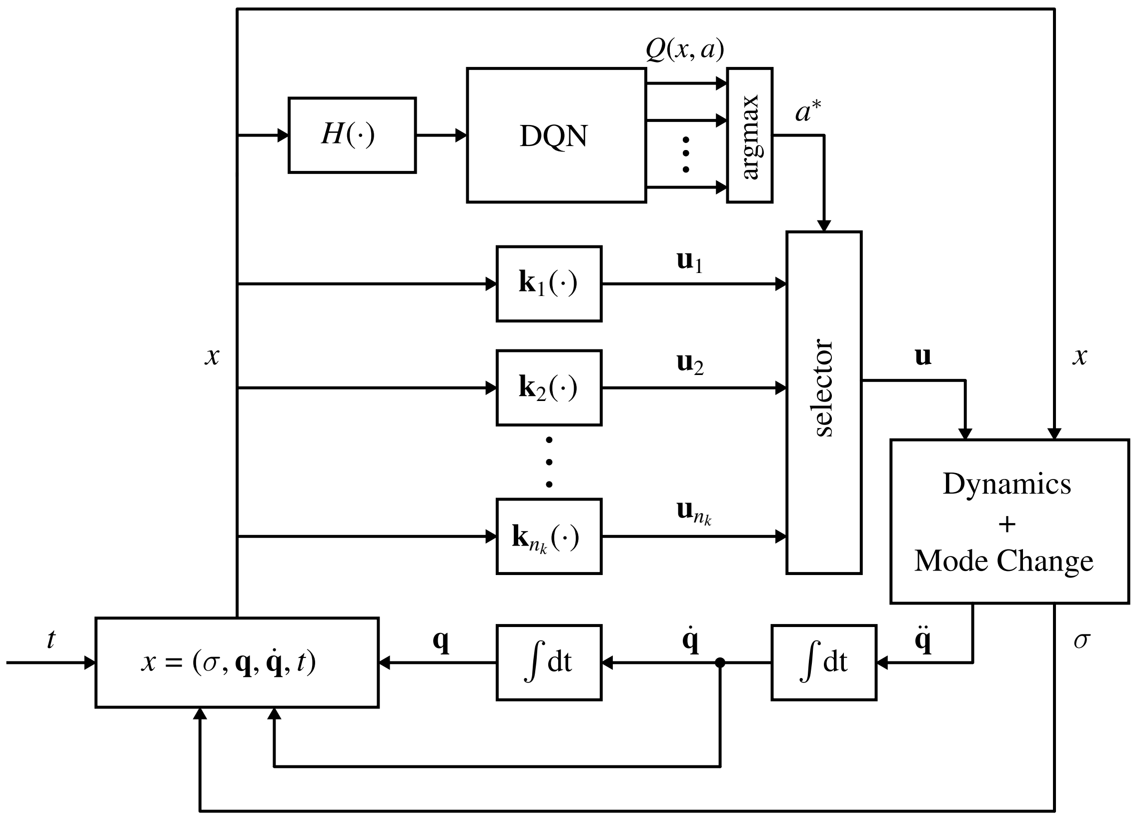


Figure 4.11: Control flow of the hierarchical controller: The block $H(x)$ produces an encoding of the current state x including a one-hot encoding of mode σ. This encoding forms the input to a deep-Q-network (DQN) that computes the state-action values $Q(x, a)$ of selecting a controller k_a. The controller with the maximal state-action value is selected and its output is used as the new control input $\mathbf{u}$. Given the current state x and controls $\mathbf{u}$ the system dynamics result in the acceleration $\ddot{\mathbf{q}}$. This acceleration is integrated twice to close the loop.

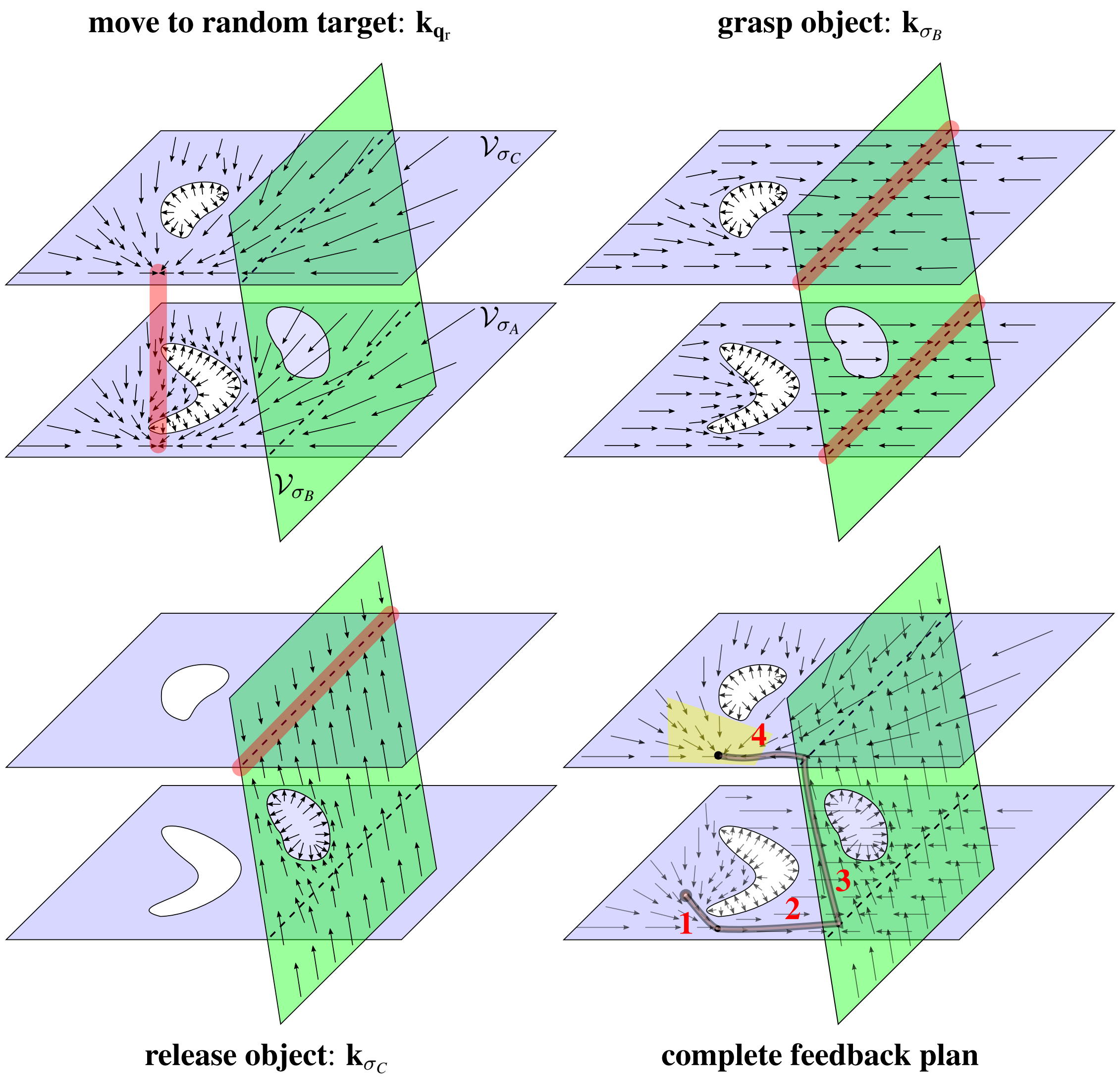

move to random target: $\mathbf{k}_{\mathbf{q}_\mathrm{r}}$
grasp object: $\mathbf{k}_{\sigma_B}$
$\mathcal{V}_{\sigma_C}$
$\mathcal{V}_{\sigma_A}$
$\mathcal{V}_{\sigma_B}$
4
3
2
1
release object: $\mathbf{k}_{\sigma_C}$
complete feedback plan

Figure 4.12: Phase portrait of the low-level controllers and the complete controller: The upper left image shows the phase portrait of a random target controller $\mathbf{k}_{\mathbf{q}_\mathrm{r}}$ that steers to a desired robot configuration $\mathbf{q}_\mathrm{r}$ (only drawn for $\mathcal{V}_{\sigma_A}$ and $\mathcal{V}_{\sigma_C}$). The random target $\mathbf{q}_\mathrm{r}$ is drawn as a red line since the object configuration is ignored. In the upper right and lower left image two mode switching controllers k_{σ_B} and k_{σ_C} are shown. The complete feedback plan is obtained by switching between these controllers. This controller is shown in the lower right image. The thick line visualizes a trajectory of the system towards the goal. For this trajectory a sequence of four active controllers is used: Step 1: The random target controller $\mathbf{k}_{\mathbf{q}_\mathrm{r}}$ steers around the C-shaped obstacle. Step 2: Controller $\mathbf{k}_{\sigma_B}$ steers towards a grasp pose. Step 3: Controller $\mathbf{k}_{\sigma_C}$ is used to move and then release the object at its new position. Step 4: Controller $\mathbf{k}_{\mathbf{q}_\mathrm{r}}$ is used again to steer towards the goal region.

4.4.2 Implementation Details

We use the same automatic controllers synthesis as in the last section with one minor modification: For all constraints, the desired dynamics are designed to have a time-constant of 0.2 s with critical damping. The number of joint-target controllers n_k was arbitrarily chosen as 20.

The focus of this chapter is not on reinforcement learning but uses it as a tool to create the high-level controller. For this reason, the network architecture and the training were designed to minimize implementation and debugging efforts.

As the ideal depth of the neural network is likely to be different for different tasks, we chose the ResNet [55] architecture for the main body of the neural network. This architecture is designed to allow the network to mimic the behavior of more shallow networks if needed. The first layer of our network is a dense layer with 64 units followed by six ResNet bocks, each with two 64 unit dense layers and ELU activations [56].

The high-level controller has a considerable action space. In the PR2 Task there are $|\Sigma| + N_\mathrm{r} = 20 + 20$ different controllers, where at least the 20 joint target controllers are available in each step. For this reason training is suffering from the typical over-estimation of Q-learning [57]. To address this we employ double Q-learning [58] and the final Dueling-Layer as presented in [59]. The total number of hidden layers is thus 13.

We strictly separate between data generation and learning. Data generation is done with 10^4 episodes with valid, random initial states and at most 1,000 steps of random actions per episode. As a mode switch requires some time for the controllers to converge, we randomly repeat actions between zero and ten times. The frequency of the high-level controller was chosen as 1 Hz. We use the resulting data set as one large replay buffer without running new episodes during training.

The discount rate was set to $\gamma = 0.95$. We optimize the weights of the network with the Adam optimizer [60] with a learning rate of 10^{-4}, its parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and a batch size of 320. The weights of the target network are updated after one epoch through the entire data set and we train for a total of 100 epochs. We model the network and optimize its parameters using Keras [61] together with Tensorflow [62].

As the controllers can be simulated several times faster than required for real-time execution and simultaneously in multiple threads, we use a policy roll-out during execution to further stabilize the policy. For the policy roll-out we use four actions with the highest Q-values and perform a roll-out of four time steps, i. e., a total of four simulated seconds.

4.4.3 Experimental Setup

To evaluate CR-DQN we designed three different manipulation-tasks as benchmarks. All benchmarks involve motion through high-dimensional state spaces with at least 20 degrees of freedom and are shown in Figure 4.13.

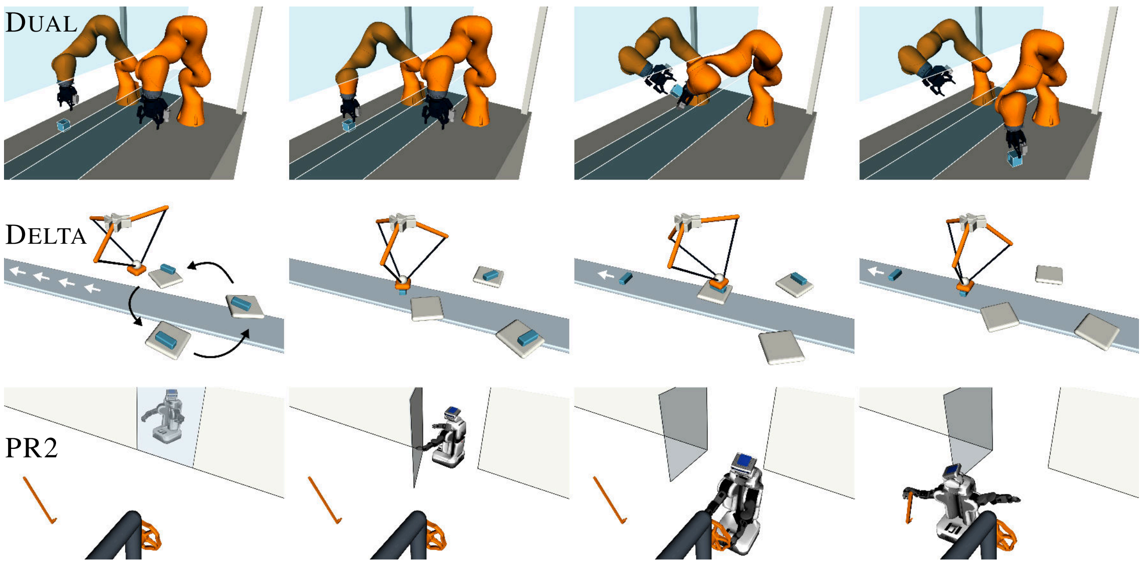


Figure 4.13: Tasks in the experiments:
DUAL Task: A cube on the left must be placed on the right side of the setup. The left robot must grasp the cube and hand it to the right through an opening in a wall.
DELTA Task: A delta-robot must grasp three boxes from rotating carriers and place them on a conveyor belt. Both for grasping and releasing the boxes the robot must synchronize its movement to the time-variant environment.
PR2 Task: A mobile manipulator has to operate a valve and a lever. It must first open a door that blocks the path. Additionally, the head-mounted camera must point at the lever, door, or valve respectively while they are being manipulated.

The DUAL task requires a dual-arm robot to transport a cube. It is designed to incorporate the typical interdependencies between motions, grasps, and placements of manipulation planning. The gap in the wall is 0.3 m wide and thus forms a narrow tunnel in the configuration space. This poses a difficult motion planning problem as well.

In the DELTA task a delta-robot has to pick three boxes from rotating carriers and place them on a conveyor. This benchmark includes closed kinematic chains, multiple objects, and a time-variant environment. To pick or to place the boxes the robot must synchronize its motion with the carriers or the conveyor respectively.

The PR2 task consists of a PR2 robot that must operate a lever and a valve. To reach lever and valve a door that blocks the way must be opened first. While the PR2 is manipulating either lever, valve, or door its head-mounted camera must point at the manipulated object. This benchmark is created to verify that our approach can address problems other than pick-and-place. Furthermore, this task contains multiple kinematic chains that are either closed or open depending on the state of the system.

For these benchmarks we implemented CR-DQN as well as CR-EST. CR-EST is the only other approach we are aware of that computes feedback plans for manipulation with

Table 4.2: Success Rates - Disturbed Execution

Benchmark		DUAL	DELTA	PR2
CR-DQN	max	100%	100%	100%
10 different networks	median	99.5%	100%	100%
	mean	98.1%	99.9%	98.5%
	min	93.0%	99.0%	86.0%
CR-EST		11%	97%	57%

articulated robots. We generated ten sets of random targets and corresponding simulated training datasets for each benchmark. This is used to train ten CR-DQNs per benchmark. Training multiple CR-DQNs is necessary for the evaluation, as the set of random targets, the training dataset, and the initial network weights are random variables.

The experiments are designed to assess the robustness of the two approaches against disturbances in the configuration $\mathbf{q}$ and the quality of the computed solutions. To evaluate the robustness we proceeded as follows: For CR-EST we randomly sampled a valid initial state and computed a plan. Then a second random and valid state within the same mode was sampled and the plan was executed for this disturbed state. It was necessary to sample a state within the same mode since CR-EST, unlike CR-DQN, cannot react to unexpected mode changes. Each trained CR-DQN was also run on a random, valid initial state (CR-DQN does not need to plan once a policy is trained). As each benchmark can be solved in less than 20 seconds, we assume that execution has failed if the goal is not reached within 60 seconds. Each experiment was repeated 100 times. They were run on two Intel Xeon 5122 quad-core CPUs at 3.6 GHz.

4.4.4 Results

Table 4.2 shows the success rates of both approaches for the three benchmarks. As the ten trained agents per benchmark have different success rates the maximal, median, average, and minimal success rates are shown. For each benchmark, at least five trained agents achieved a 100% success rate. For the DUAL and PR2 tasks every trained CR-DQN is significantly (p-value < 1%) more robust than the CR-EST. No significant difference can be observed on the DELTA task as both approaches have success rates close to 100%. We made the following observations on the failures of both approaches:

CR-EST: The typical failure mode for the DUAL task of CR-EST is to get stuck in an axis limit. For the PR2 task CR-EST is prone to get the PR2's arms tangled and cannot proceed due to collision constraints. In the DELTA task the last joint of the delta-robot is continuous and thus has no relevant axis limit and the collision geometry is mostly convex.

Table 4.3: Average Execution Time in Seconds

Benchmark		DUAL	DELTA	PR2
CR-DQN	max	20.1	19.5	19.4
10 different networks	median	16.7	14.7	19.2
	mean	16.8	15.1	19.1
	min	13.4	13.0	18.9
CR-EST	execution time	13.7	16.0	20.8
	planning time	3.83	4.58	2.44
	planning + execution	17.6	20.6	23.2

For this reason CR-EST succeeds with high probability during disturbed execution. It should be noted that the random initial states used for execution are sampled globally within the valid set of states that are within the start mode (not just local disturbances as in the previous section). Furthermore, we do not let CR-EST re-plan if it gets stuck, which would likely increase robustness considerably.

CR-DQN: The failures of CR-DQN we observed are due to cyclic switching between different low-level controllers. This occurred mostly during the hand-over of the dual-arm robot, where the trained agent alternates between different controllers that attempt to switch to different grasp modes. These infinite cycles mean that the high-level controller does not "pull through" with a mode switch. This indicates that more sophisticated training methods could further increase robustness.

Table 4.3 shows the execution times for the three benchmarks. Additionally, the planning times and combined planning and execution times of CR-EST are shown. CR-EST is not an optimizing planner and returns the first solution it finds. In contrast the goal for the reinforcement learning that forms the high-level controller of CR-DQN aims at a time-optimal policy (for the given set of controllers). If the training had converged to the global optimum one would expect a clear advantage of CR-DQN.

The results in Table 4.3 show, that this is not consistently the case. For the DUAL and DELTA tasks there are trained CR-DQNs with both better and worse average execution times compared to CR-EST. Only for the PR2 task our approach has consistently better performance. The observed behavior that causes these inefficiencies is again brief oscillations between low-level controllers.

When comparing the execution times of the proposed approach with the combined planning and execution times of CR-EST the results are, on average, in favor of CR-DQN. It should however be noted that CR-DQN requires that data generation and network training can be done offline ahead of the actual execution. For the benchmarks the combination of data generation and network training took between 4.5 hours and 25 hours.

In addition we conducted two qualitative experiments. As in the previous section we can use the the low-level controllers to add additional constraints at runtime. We use this for online collision avoidance in the DUAL task on a real robot as shown in Figure 4.9. While the high-level controller does not know about these modifications, the low-level controllers will still avoid collisions while respecting the constraints of the task.

As second qualitative experiments we execute the DUAL task in simulation with an additional disturbance. Two seconds after the cube is picked up by the left robot we place it back on the surface to simulate an object that is accidentally dropped. As the high-level controller provides a global mapping between state and active controller it should react and pick up the dropped object again. In the qualitative experiments the collision avoidance and the reaction to the dropped part operated as intended.

4.5 Related Work

Specifying and controlling robotic motions for manipulation is a challenging problem. One reason for this is that motions that are relevant to a task occur in coordinate systems that are different from the controllable degrees of freedom of the robot. In many cases, the robot has redundant degrees of freedom relative to the task. An example is a seven axis robot that picks an object with a suction cup. As the suction cup has a rotational symmetry, the task constrains only five degrees of freedom. Thus, two redundant degrees remain.

Siciliano and Slotine [63] present a framework for specifying and controlling motions for redundant manipulators. Tasks are represented as a hierarchy of constraint functions. The velocities of the robot axes are then recursively projected into the null-space of the Jacobians of the constraint functions. This allows to flexibly compose an arbitrary number of individual tasks. Mansard *et al.* [45] provide a software framework, the stack of tasks, that allows to specify such a control scheme using a graph representation.

To unify a flexible task specification, motion control, and state estimation De Schutter *et al.* [41] introduce a framework for instantaneous Task Specification using Constraints (iTaSC). In this framework, tasks are described as closed, virtual kinematic chains. These closed chains are dependent on the robot configuration, geometric feature variables, and uncertain model parameters. From the loop closure equations, motion controllers and state estimators are derived automatically. Several extensions to this framework have been developed, such as the inclusion of inequality constraints [64] and geometric path constraints [65]. Aertbeliën and De Schutter [43] introduce the eTaSL/eTC framework that enables the definition of such constraints via a graph-structure of computational expressions. This enables to efficiently program complex multi-robot motions with online collision avoidance. Manipulation requires not a single robot motion but a sequence of motions and interactions with objects. Scioni *et al.* [66] propose a scheduler that automatically composes and sequences constraint-based controllers based on a constraint satisfaction problem.

However, constraint-based controllers are not suitable to address the sequential interdependence of motions and actions that arise in multi-robot manipulation. Our approach makes use of a constraint-based specification of a manipulation problem using the eTaSL framework [43]. To address the sequential interdependencies of manipulation we simulate motion controllers as local planners.

Reasoning about the sequence of motions of robots and objects in an integrated fashion is known as manipulation planning and was first adressed by Alami *et al.* [8]. As manipulation requires reasoning about high-dimensional configuration spaces, sampling based approaches [11, 28, 67] have shown good empirical performance. Several extensions have been proposed, such as heuristic guidance for large scale planning problems [33, 35] and optimal planners [38, 40].

A shared limitation of these manipulation planners is the difficulty to adapt them to new domains or to incorporate new constraints to a given domain. This is due to the fact, that these planners rely on problem-specific sampling algorithms and sometimes also steering functions. A more generic approach to planning, is to model manipulation as constrained motion, where constraints arise due to contacts [68]. An overview of sampling-based planning methods for such constrained motion is provided by Kingston *et al.* [7]. Mirabel and Lamiraux [9] present the constraint graph as a systematic approach to model different contact states of manipulation and the resulting constraints. Furthermore, a planner is proposed that operates on this model. The HPP framework [69] supports the definition of such planning domains. In [42] a method is proposed that leverages explicit constraints to speed up planning. Toussaint proposes a constraint-based approach to manipulation planning, called Logic Geometric Programming (LGP) [39]. While the constraint graph [9] assigns different constraints to different contact states, LGP [39] views constraints as part of high-level actions. Large scale manipulation problems can then be solved by repeatedly solving trajectory optimization problems. This work has been extended to problems with partial observability of the contact state by Phiquepal and Toussaint [70].

The model proposed in this chapter, the dynamic constraint graph, modifies and extends the constraint graph [9] in several ways. As first modification we introduce an explicit parameter for time and thus enable time-dependent constraints. This allows to model time-variant environments including conveyor belts. As second modification we model our system with second-order dynamics and consequently allow constraints on velocities and control inputs. This is necessary both for time-variant systems and for a switching control scheme. As third modification our model has no need for the parametrized constraints of [9] (equivalent to the contact parameter of the previous chapter). The reason is that this kind of constraint can be subsumed as combination of position and velocity constraints in our model. This simplifies the implementation of our approach and the modeling of manipulation tasks. Like in LGP [39], we model all constraints, including collision avoidance, as constraint functions. This is critically important for online, reactive control that should also not cause collisions.

The planners discussed so far compute trajectories that solve a planning problem but assume a known and static environment. These trajectories do therefore not encode how to react to unforeseen disturbances. Burridge *et al.* [71] present a general approach to combine deliberate planning and reactive execution. Local controllers, or funnels, for which a region of attraction is known are structured as a tree of controllers that covers the state space. For manipulation this approach is problematic due to the so called crossed foliation issue [9]. For nearly all pairs of valid configurations there exists no connecting path that does not involve at least one mode switch. This makes the computation (or even definition) of regions of attraction difficult.

Bozma and Koditschek [72] present a feedback planner for a one-dimensional manipulation problem. This planner uses a set of controllers that are switched in a provably correct way to manipulate two objects. In [73] this approach is experimentally validated for a two-dimensional environment with multiple objects. Vasilopoulos *et al.* [74] combine an optimal manipulation planner with reactive controllers. A provably correct feedback manipulation plan can be constructed for two-dimensional environments with known, non-convex and unknown, convex obstacles. These approaches are currently limited to spherical robots in two-dimensional environments. In contrast, our approaches scale to scenarios with articulated robots with 20 actuated joints.

For dynamic and non-prehensile manipulation tasks Woodruff and Lynch [75] present an integrated approach to planning and control that operates in two stages. In the first stage a dynamically feasible manipulation plan is computed that may include non-prehensile actions such as throwing or balancing an object. Within the second stage a feedback controller is designed that stabilizes the motion of the object along this trajectory whenever the robot is in contact with the object. However, this controller does not address safety or task-related constraints, such as collision avoidance.

The planner-based approach we present in this chapter formulates the problem of manipulation as a kinodynamic planning problem that is solved by sequencing constraint-based controllers. As the controllers are derived from the domain model, the reactive execution adheres to the constraints of the planning domain.

A different approach to construct a feedback policy for manipulation is to train a reinforcement learning agent within a physics simulator. Andrychowicz *et al.* present recent successes for pick and place tasks [76] and in-hand manipulation [77]. Reinforcement learning can capture complex, long-term interdependencies of the decision problem and operates, by construction, in real-time. However, the task specification of reinforcement learning is a reward function, which makes the implementation of hard constraints, like collision-avoidance or precise grasping, difficult. For successful reinforcement learning, considerable efforts are typically required to shape this reward function. Furthermore, reinforcement learning struggles with high control-frequencies, as these effectively increase the time-horizon of the problem.

With the proposed learning-based approach to manipulation planning we make use of reinforcement learning but bypass some of its challenges. As we automatically derive controllers from the underlying planning problem we decouple the learning from the hard constraints, such as collision avoidance, and from the high control-frequencies. The constraint-based motion controllers prevent the system from collisions or violations of axis limits and operate with a high control frequency (in our experiments: 200 Hz). The long term reasoning about the sequence of motions and actions is performed by a reinforcement learning agent at a lower frequency. This agent switches, in deterministic time, between the different controllers to achieve its goal. As the motion controllers implement goal directed motions, such as grasping an object or opening a door, there is a high probability that the reinforcement learning agent will explore such behaviors during training. This results in a problem formulation that does not require any reward shaping.

4.6 Discussion

This chapter addressed the issue of modeling and executing complex manipulation tasks in dynamically changing environments. To this end we proposed a new model: the dynamic constraint graph. This model enables to describe tasks by composing constraint functions in a modular and flexible way.

For this model we presented a method to automatically derive motion controllers that implement goal directed motions, e.g, a controller for opening a door. These controllers were then used in two feedback planners to compute reactive manipulation plans.

The first feedback planner models manipulation as a kinodynamic motion planning problem. By using the controllers as local planners, we reformulated the Expansive Space Tree planner [6] for manipulation planning problems. Instead of computing a trajectory, this planner computes a sequence of motion controllers. During execution, these controllers react in real-time to estimates of the environment. We validated this approach in real-world experiments. In these experiments, a dual-arm robot correctly reacted to measurements of object poses and avoided obstacles that were detected online.

For the second planner, we reformulated the planning problem as a reinforcement learning problem. In this setting, a reinforcement learning agent switches between motion controllers to reach a set of goal states. This agent is trained in simulation and provides a global mapping of states to active controllers. With a global mapping this approach can also react to unanticipated mode changes. An example is an object that is accidentally dropped by the robot. In experiments our learning-based approach showed the correct reaction of grasping the object again and continuing with the task. We validated this approach on three distinct robots and tasks.

The main advantage of the methods in this chapter is that they provide a semantically rich interface to the “stakeholders” of robotic manipulation. This includes human programmers of manipulation tasks, manipulation planning algorithms, and motion controllers. Modeling manipulation tasks with the dynamic constraint graph is surprisingly intuitive and allows to compose tasks in a modular way. This is especially simplified by using the expression graph formalism of the eTaSL/eTC framework [43]. By deriving controllers from this model and using them as local planners we obtain reactive plans that encode both a sequence of actions and the original constraints of the underlying model. This means that the reactive execution is aligned with the constraints that the programmer of a task intended. In contrast, a planned trajectory “forgets” this original model. Furthermore, the constraint functions that define the controllers can be used for a variety of control schemes. We described an acceleration resolved scheme in this chapter. In unreported experiments we have also used these constraints for a jerk resolved control scheme that implements a third-order differential equation for the constraint functions. Optimal control schemes, such as model predictive control (MPC), are promising candidates for future research and can also operate on the same model. Potentially, constraint-based state estimation as presented in the iTaSC framework [41] could be integrated with the proposed methods.

A limitation of the methods in this chapter is that it is not possible to guarantee completeness or optimality. The cause for this lies in the linear dynamics of the controllers that are derived from the domain model. Due to the linear dynamics it is not possible to explore the full state space, i. e., positions, velocities, and time. This could be overcome with a different strategy for controller synthesis and is a promising area for future research. One possible approach is to derive controllers using model predictive control. MPC could enable to explore the full state space on optimal trajectories and thus enable guarantees on completeness and optimality.

4.7 Relation to Previous Publications by the Author

The results of this chapter have been previously published within conference proceedings:

- **Planning Reactive Manipulation in Dynamic Environments**

  P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G.v. Wichert, W. Burgard

  2019 IEEE/RSJ International Conference on Intelligent Robots and Systems

  © IEEE 2019

- **Modeling and Planning Manipulation in Dynamic Environments**

  P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G.v. Wichert, W. Burgard

  2019 IEEE International Conference on Robotics and Automation

  © IEEE 2019

The author of this thesis is the main contributing author of these previous publications and their contents were developed in pursuit of this thesis. For this reason, large parts of this chapter are identical to the previous publications. To improve readability we omitted citations of these publications throughout this thesis. The content has been modified and extended within this thesis.

Changes Relative to the Conference Proceedings Besides changes in wording and formatting much of this chapter is identical to the previous publications. Larger changes are listed in the following.

- Merged sections: Several sections that serve a similar purpose in the previous publications have been merged in this chapter. This includes the introduction, the problem statement defined by the dynamic constraint graph, the automatic controller synthesis, related work, and the discussion.
- Discussion of the controller synthesis: In Section 4.2.3 a detailed discussion of the synthesized controllers was added.
- Change of notation: To achieve a consistent notation across this thesis, we modified the problem statement, the automatic controller synthesis, and the descriptions of the algorithms to follow the notation of this thesis.
- Related work: The contents of Section 4.5 were not only merged from the previous publications but also modified to follow the citation style of this thesis. Additional references were added.

Contributions of Coauthors The previous publications were written together with Florian Wirnshofer, Kai Wurm, Georg v. Wichert, and Wolfram Burgard.

- Florian Wirnshofer supported the transfer of the methods to the real, dual-arm robot and the execution of the real world experiments. He provided consulting with respect to the reinforcement learning aspects of this chapter.
- Kai Wurm revised the papers and gave guidance on scientific writing.
- Georg v. Wichert and Wolfram Burgard provided valuable advice.

Chapter 5

Planning Manipulation for Assembly in a Skill-Framework

In the previous chapters we modeled manipulation as a sequence of motions of a robot and objects. Between motions the robot makes or breaks contact with the objects. While this model covers a wide range of manipulation tasks, there are other actions or process steps that cannot be addressed in this way.

Let us consider the example in Figure 5.1, which depicts a partial assembly of a switching cabinet. A dual-arm robot uses its cameras to locate two parts, an electrical component

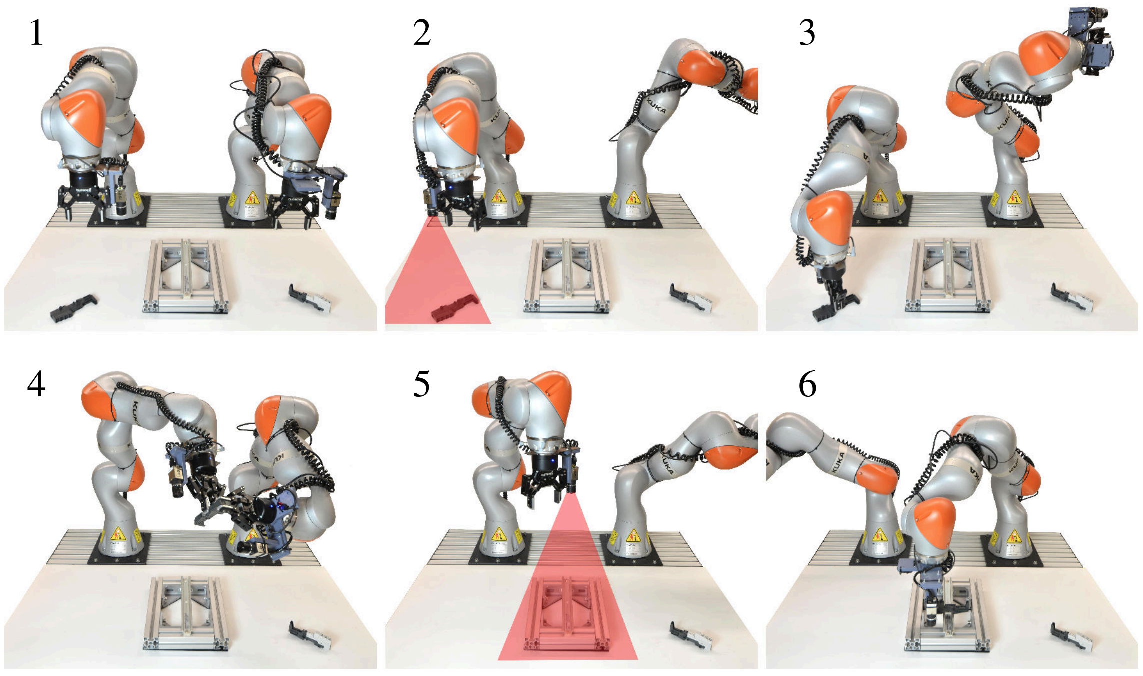


Figure 5.1: A task and motion planning problem: The task of the robot is to assemble the component on the left onto the top hat rail in the center. The component and the top hat rail must be localized using a camera (marked red). Then the component is grasped sideways, handed over to the right arm, and finally mounted onto the rail.

and a top hat rail. Then the component is grasped and assembled on the rail. This example illustrates two interesting properties of assembly tasks.

The first property is the same sequential interdependence of motions and other actions that we covered in the previous chapters. On an abstract level, the component must first be localized before it can be grasped, which again is necessary to assemble the part onto the previously localized top hat rail. On a more detailed, geometric level, the part must be reoriented by a handover to be assembled on the rail, as the gripper would otherwise collide with the rail. Also, kinematics and geometry constrain which of the two cameras can be used to localize the parts.

The second property is that not all constraints of this task and not all actions of the robot lie within motions or changes of contact states. To localize an object the camera is placed in a view pose above the object and a vision algorithm is triggered. The resulting increased accuracy of the object's pose estimate is a prerequisite to the following manipulation steps. To assemble the component onto the rail the robot employs a force controlled assembly strategy. Technically this is a motion of the robot and the objects and might be computed by a physics-based manipulation planner. It may however be more convenient to compute this strategy externally and provide a planner with a suitable and more abstract model of this process.

To address problems like the assembly in Figure 5.1 we need models and planners that address constraints and actions that do not lie within motions of robots and objects. The resulting planning problem extends manipulation planning to what is called task and motion planning. Models for this kind of task do not only need to describe the constraints of sequential motions and actions but also how they can be executed on a real robot. This poses several challenges for modeling and planning:

- High-dimensional, hybrid configuration spaces: Motions for industrial manipulators require planning in high-dimensional configuration spaces with differential constraints imposed by the robot dynamics. These motions occur in a hybrid discrete/continuous space. Typically the discrete component occurs due to the robot making or breaking contact with its environment.

- Optimality: In order to reach or outperform human efficiency, robots must compute high quality or even optimal solutions. Cycle times are one key performance indicator for most tasks. Therefore, the dynamic limits of the robot and their effect on execution times must be considered during planning.

- Diverse interactions with objects: Real world tasks for robots come with a large variety of interactions between the robot and objects in its environment. It is therefore important that a diverse set of manipulation actions and their constraints on robot motions can be handled by a planner.

The contributions of this chapter are a model for task and motion planning problems and corresponding planners that address these challenges. In a first step, we propose a model of robotic skills that encapsulates both the constraints for planning and the controlled execution on a real robot. These skills are designed to be used in a kinodynamic task and motion planner.

In a second step, we show how a planning problem that is based on our skill model can be reformulated as a multi-modal planning problem. For this multi-modal planning problem we propose an asymptotically optimal task and motion planner. Under a novel set of conditions we prove probabilistic completeness and asymptotic optimality of our approach.

We have implemented our skill model and the proposed planners for a multi-robot assembly scenario. In extensive simulated experiments our approach computes plans with significantly shorter execution times than an existing planner. In real world experiments our approach forms the high-level control loop for a robotic assembly cell. These real world experiments show that our skill model enables planning with a diverse set of manipulation actions and that the proposed planners scale to problems of practical relevance.

5.1 Skills for Autonomous Assembly

In this section we introduce our model of robotic skills. This model has two purposes: The first purpose is to enable the flexible modeling of task and motion planning problems. The second purpose is to create a mapping between solutions of these planning problems to a controlled execution on a real robot.

In its broadest definition, a skill is a parameterizable controller that

- can be reused on a variety of different kinematic structures,
- includes a model of itself in the form of a detailed simulator,
- is able to provide suitable parameterizations of itself,
- and, ideally, can be reused across a variety of tasks.

The last point in this list is a strong requirement. While some recurring components can be identified, e. g., picking or placing, we assume that it is not viable to enforce this ideal state for all automation tasks. For this reason, our proposed model is designed to be modular and extensible even if single purpose behaviors of a robot must be designed.

In Section 5.1.1 we introduce the representation of a system state. We propose our model for robotic skills in Section 5.1.2. Section 5.1.3 presents examples of skills that we implemented for an assembly use case.

5.1.1 System State

We assume that the state of a system can be encoded with two kinds of variables: a continuous configuration $\mathbf{q}$ and a discrete mode σ. The continuous configuration $\mathbf{q} \in \mathbb{R}^n$ encodes the positions of all robot axes and the poses of all objects. Its velocity or time-derivate is written as $\dot{\mathbf{q}}$. A configuration $\mathbf{q}^\top = [\mathbf{q}_\mathrm{r}^\top, \mathbf{q}_\mathrm{o}^\top]$ is composed of a robot configuration $\mathbf{q}_\mathrm{r} \in \mathbb{R}^{n_\mathrm{r}}$ and an object configuration $\mathbf{q}_\mathrm{o} \in \mathbb{R}^{n_\mathrm{o}}$. In the example of Figure 5.1 the dual-arm robot has a total of $n_\mathrm{r} = 14$ axes. The example includes three objects for which the poses can be encoded in a vector quaternion representation. This results in $n_\mathrm{o} = 21$ values to encode the object configuration $\mathbf{q}_\mathrm{o}$.

The mode $\sigma \in \Sigma$ encodes the discrete, high-level state of the system. It includes the information of the contact state of the previous chapters, i. e., which object is rigidly attached to which component of the robot or the environment. The mode σ also includes other process-related information about the system state. In the example of Figure 5.1 it encodes whether the pose of an object is currently estimated with sufficient accuracy. A mode σ is an element of a finite set of modes Σ. The literature on task planning provides a variety of possible representations of this state, e. g., the Planning Domain Definition Language (PDDL) [32].

A full state $x = (\mathbf{q}, \dot{\mathbf{q}}, \sigma) \in \mathcal{X}$ comprises the continuous configuration, its velocity, and the discrete mode. Here, $\mathcal{X}$ denotes the full state space. We omit the dynamics of $\mathbf{q}$ at this point as well as the mechanism for mode switches. These aspects will be discussed in the following section.

We assume that objects are at all times rigidly attached to components of the robot or to the environment. The contact parameter $\pi \in \Pi$ encodes both the mode σ as well as the relative transforms between the objects and the links of the system they are attached to. Thus, we can compute the object configuration $\mathbf{q}_\mathrm{o}$ via forward kinematics $\mathbf{q}_\mathrm{o} = \mathrm{fk}(\mathbf{q}_\mathrm{r}, \pi)$. For this reason and with slight abuse of notation, we will treat a tuple $(\mathbf{q}_\mathrm{r}, \dot{\mathbf{q}}_\mathrm{r}, \pi)$ equivalently to a full state x.

The mode σ, the contact parameter π, and the robot configuration $\mathbf{q}_\mathrm{r}$ form a hierarchical representation of a state that is useful for task and motion planning. The discrete mode σ encodes which series of high-level actions may potentially transform the system state into a desired state. An example is a sequence of pick and place operations that can be derived from this mode. Some of these series may be infeasible or suboptimal due to constraints that arise on the level of geometry and dynamics. As objects are assumed to be rigidly attached, the motion of a configuration $\mathbf{q}$ is constrained onto a manifold defined by the contact parameter π. The contact parameter is therefore a suitable intermediate representation that encodes the effects of the crossed foliation issue [9] that arises in manipulation. Given a contact parameter π, the robot configuration $\mathbf{q}_\mathrm{r}$ describes all positions of robots and objects. Also, if we assume that all robot axes are actuated and the robot contains no closed kinematic chains, the system is holonomic in $\mathbf{q}_\mathrm{r}$. This results in a suitable representation for motion planning.

The proposed model of robotic skills is built around this three-layer abstraction of the system state. This allows us to define skills that both partially define task and motion planning problems and are part of a planner that solves them.

5.1.2 Skill Model

The proposed model of robotic skills consists of three main components and is visualized in Figure 5.2:

1. Skill Parameterization

   Symbolic Action Parametrization

   Contact Parametrization

   Motion Parametrization

2. Simulated Execution and Validity Check

3. Execution on the Real Robot

In the following we introduce these components in detail.

Skill Parametrization The first component of a skill is its parametrization. Let us consider a skill for picking up an object. In order to execute such a high-level action several decisions must be made by the robot. On an abstract, symbolic level the robot must decide which object should be picked with which gripper. On a geometric level, the robot must select a grasp pose that determines the relative pose of gripper and object. Finally, grasping requires a controlled interaction with the object. This may include parameters for force-controlled Cartesian motions and commands to a gripper. A key aspect of our skill model is that each skill is equipped with its own sampling mechanisms to propose these parameters. This sampling is separated into three steps that mirror the three layers of abstraction of the system state.

The input to the **symbolic action parametrization** is the current state $x \in \mathcal{X}$ and it produces a set of target modes as output: $\{\sigma_{\text{new},1}, \sigma_{\text{new},2}, \ldots\}$. In the example of a pick skill, the parameter sampling considers all grippers of the robot that are currently empty and all objects that are currently placed and localized with sufficient accuracy. For each pair of empty gripper and placed object one mode, where the object is grasped, is added to the set of target modes.

The input to the **contact parametrization** is the current state $x \in \mathcal{X}$ and a target mode σ_{new}. As output it computes a set of target contact parameters $\{\pi_{\text{new},1}, \pi_{\text{new},2}, \ldots\}$. In the case of a pick skill this returns a set of grasp poses that determine the relative transformation of gripper and object after the grasp. The contact parametrization step may

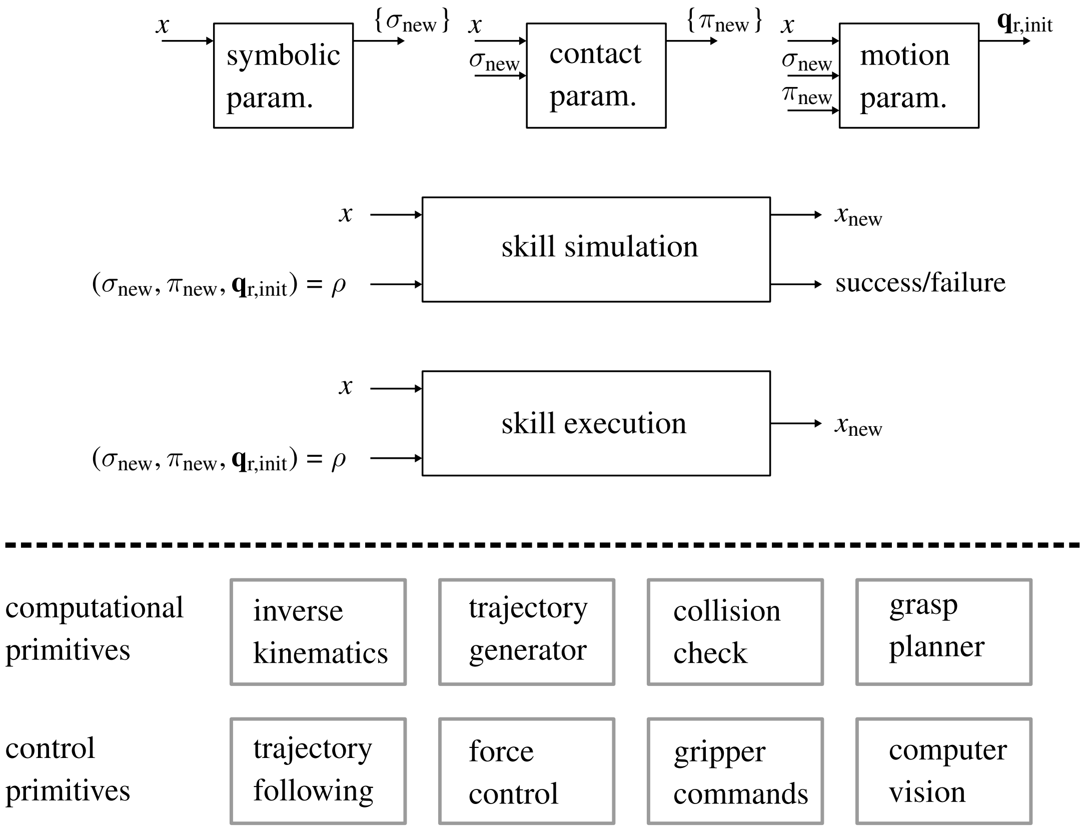


Figure 5.2: Skill model: The upper half of the image shows the components of a robotic skill. Three blocks on the top visualize different stages of skill parameterization. These parameters may then be used to simulate the outcome of a skill during planning or for execution.
We aim for robotic skill that can be re-used across different robots and tasks. For this reason the parametrization, the simulation, and the execution should be based on primitive operations that can be re-used across robots and tasks. The lower half of the image shows examples for both computational primitives and control primitives that we used for the implementation of skills.

be exhaustive for the case of a limited number of possible contacts. For continuous sets of possible contacts, e. g., for grasp poses, a set of random samples is returned.

Finally, the **motion parametrization** determines a configuration of the robot from which it can interact with its environment in order to reach a target mode σ_{new} and contact parameter π_{new}. We denote this robot configuration from which the skill may be executed as $\mathbf{q}_{\text{r,init}}$. In the example of the pick skill this step solves the inverse kinematics problem of positioning the gripper in a pre-grasp pose. For redundant robots we sample a random robot configuration that solves this problem.

In the following we denote a full skill parametrization as $\rho = (\sigma_{\text{new}}, \pi_{\text{new}}, \mathbf{q}_{\text{r,init}})$. This includes the target mode σ_{new}, the target contact parameter π_{new}, and the robot configuration $\mathbf{q}_{\text{r,init}}$ from which the skill can be executed.

Simulated Execution and Validity Check The second component of a skill is its simulation and a validity check, both used for planning. As input this component takes a system state x and a skill parameterization ρ. The purpose of this component is to compute the state x_{new} that results from executing the parametrized skill and to decide whether the actions of the robot are valid, i. e., at least collision free. This component may be as simple as a collision check for a robot motion. However, it may also involve the forward integration of a control strategy.

In the example of the pick skill, the robot approaches the object and retracts from the surface the object initially rests on with linear Cartesian motions. These motions, beginning in the robot configuration $\mathbf{q}_{\text{r,init}}$, are checked for collisions during the simulated execution. The collision checks between gripper and object and between object and the surface are disabled. Disabling these collision checks is helpful due to numerical issues in the collision check as the object and the surface are exactly touching. Due to the linear, Cartesian motions we know that these collision checks may be disabled safely, i. e., the simulation component allows to encode knowledge about the manipulation task.

Execution on the Real Robot Finally, the skill execution takes a parametrized skill and executes it on the physical robot. The result is an estimate of the current state x. In the case of a pick skill, the robot follows the Cartesian motions and closes the gripper to grasp the object. It is also possible to implement fault detection strategies. If the robot closes its gripper and does not detect an object our implemented pick skill opens the gripper again and retracts. In this failure case the current state x is updated with an uncertain location of the object.

Discussion of the Skill Model The purpose of the presented skills is twofold. First, we use them to model the action space of a planning domain and at the same time guide the search of a planner. Second, they are used to define a mapping between a plan, as a sequence of skills, and the execution on the real robot. The key aspect of our model is that it encapsulates aspects of manipulation planning that are inevitably coupled: modeling of planning domains, action selection, validity checks, and execution on the real system.

To explain this in more detail let us revisit the Cartesian motions to approach an object within the pick skill. Using Cartesian motions to approach an object during grasping is useful for multiple reasons. During execution these motions reduce the risk of colliding with the object or scratching the surface with the object once it is grasped. Furthermore, they simplify the implementation of force controlled interaction and failure detection. During planning these motions allow us to use safety margins within our collision checks as we only need to plan towards the approach configuration. This speeds up motion planning and avoids numerical issues of collision checking close to contact.

The difficulty with this integration of domain specific knowledge is that it affects the modeling of the planning domain, the behavior of the planner, and the mapping of the resulting plans to an execution on the real system. In our model of robotic skills these aspects are encapsulated within the skill model and are not visible during the implementation of another skill. At the same time, the abstractions used for the state representation allow the re-use of components such as trajectory generation, collision detection, or inverse kinematics for the definition of multiple skills.

5.1.3 Detailed Example: Mount Skill

The purpose of the MOUNT skill is to robustly assemble one part onto another. As we follow a model based approach, this skill derives its assembly motions from CAD data as shown in Figure 5.3.

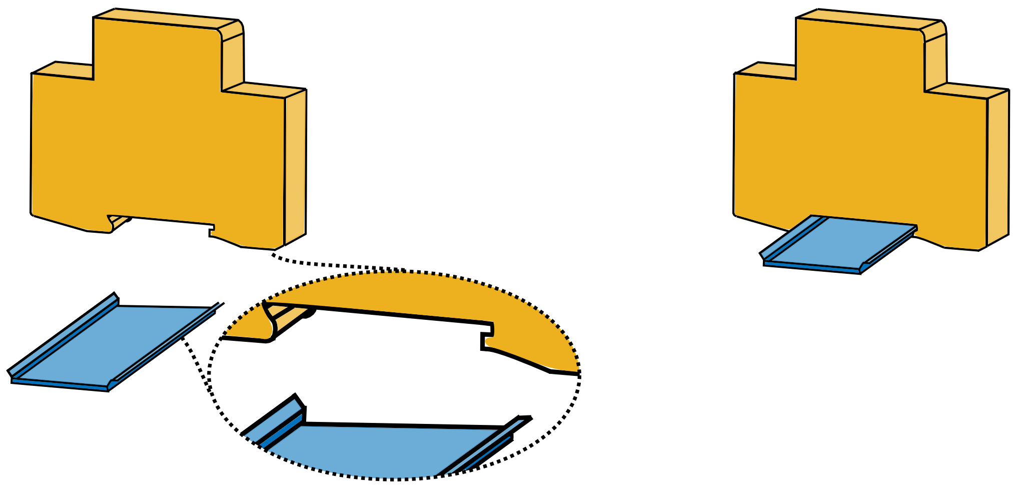


Figure 5.3: CAD-based specification of assembly tasks: The mount skill operates on CAD data that specifies the object geometries and their desired relative poses. Image taken from [80] and modified. © IEEE 2018

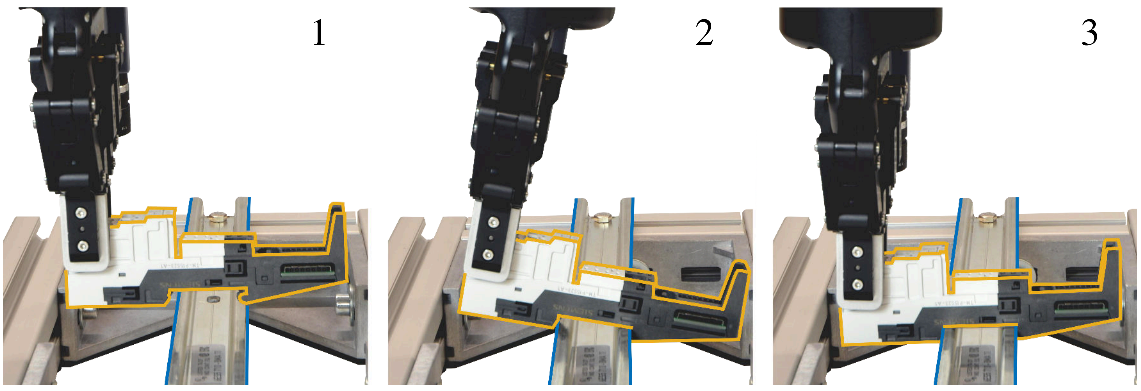


Figure 5.4: Assembly sequence computed by the MOUNT skill. Image taken from [80] and modified. © IEEE 2018

Symbolic Action Parametrization: The prerequisite for this skill is a part that is held in a gripper and may be assembled to a second part, which is resting on a surface. Both parts are localized with sufficient accuracy. For each such pair this step returns a mode where the parts are assembled and the uncertainty of their pose is increased.

Contact Parametrization: This step returns the contact state of the two objects after the assembly. In case the objects can be assembled in multiple configurations, such as on the rail in Figure 5.4, multiple contact states are returned.

Motion Parametrization: A robust Cartesian trajectory for assembly is computed using the B-EST planner presented by Wirnshofer *et al.* [80]. For efficiency reasons this trajectory is computed offline and stored for later use. A random inverse kinematics solution is sampled for the beginning of the Cartesian path to obtain the configuration $\mathbf{q}_{\mathrm{r,init}}$.

Simulation: Beginning from the configuration $\mathbf{q}_{\mathrm{r,init}}$ the Cartesian path is checked for kinematic feasibility. If this succeeds the Cartesian motions are checked for collisions. The collision check between the two objects is disabled as their contact is desired. If this succeeds, the resulting state assumes the robot axes at the end of the Cartesian motions and the parts to be assembled. The belief of the resulting assembly encodes a large variance for the poses of the involved objects as the assembly process may have shifted them.

Execution: Beginning in the robot configuration $\mathbf{q}_{\mathrm{r,init}}$ the robot follows the Cartesian assembly motion. This motion is tracked with a Cartesian impedance controller [81] to allow a compliant interaction. The parameters of this controller are chosen identically to those used for planning the assembly motions with the B-EST planner [80]. Once the parts are assembled the grippers are opened and the current state is updated.

Several aspects of this skill are worth discussing. All subcomponents that are used for parametrization, simulation, and execution are model-based and, given the models, arguably independent of the involved robots and objects. Collision checking, randomized inverse kinematics, and force-controlled motions are applicable to many robots of practical relevance. The computation of the assembly motions is also based on CAD data only.

Furthermore, the skill does not include any global motion planning or manipulation planning. This aspect is addressed by the planner in the following section.

5.2 Task and Motion Planning with Skills

This section introduces the problem domain for planning with parameterizable skills and presents a sampling-based algorithm for that domain. With this planner it is possible to define a task-level controller.

The purpose of this section is to provide an intuition of how the proposed skill model can be used for planning. For this reason we provide an informal introduction of the planning problem and the corresponding planner. This means that we do not specify the planning problem in terms of robot motions and constraints on them but in terms of primitive computational operations. A more formal description and proofs for probabilistic completeness follow in the next section.

5.2.1 Primitive Operations and Informal Problem Setting

Let us recall that a state $x = (\mathbf{q}_\mathrm{r}, \dot{\mathbf{q}}_\mathrm{r}, \pi)$ of the system can be encoded as the positions and velocities of the robot axes and a contact parameter π. We denote a continuous robot trajectory as $\tau_\mathrm{r} : [0, t_\mathrm{end}] \to \mathbb{R}^{n_\mathrm{r}}$. We use the following shorthand to abbreviate the end of a trajectory τ for a given contact parameter:

$$x_\mathrm{end} = ((\tau(t_\mathrm{end}), \dot{\tau}(t_\mathrm{end}), \pi) = \mathbf{end}(\tau, \pi).$$

Given a state $x_1 = (\mathbf{q}_{\mathrm{r},1}, \dot{\mathbf{q}}_{\mathrm{r},1}, \pi)$ and a robot configuration $\mathbf{q}_{\mathrm{r},2}$ we can compute a trajectory τ towards this configuration with the operation $\tau = \mathbf{localPlanner}(x_1, \mathbf{q}_{\mathrm{r},2})$. The operation $\mathbf{isValid}(\tau, \pi)$ returns whether such a trajectory is valid, i. e., at least collision free. With the operation $\tau = \mathbf{randomMotion}(x)$ we receive a random motion that begins in x.

We denote the set of available skills as $\mathcal{S}$. Let us recall, that a skill is parameterized with a parameter $\rho = (\sigma_\mathrm{new}, \pi_\mathrm{new}, \mathbf{q}_{\mathrm{r,init}})$. Given a skill $s \in \mathcal{S}$, the operation $\{\rho_1, \rho_2, \ldots\} =$ **sampleParams**(s, x) returns a set of such parameters that might be applicable in state x. This set may be empty for some skills in some states. The operation $x_\mathrm{new} = \mathbf{simulateSkill}(x, s, \rho)$ returns a prediction of the system state after applying a skill s with a given parameter ρ.

We assume that the system is at an initial state x_start and should reach a set of goal states $\mathcal{X}_\mathrm{goal}$. The (informal) planning problem is now to find an alternating sequence of robot trajectories τ and parametrized skills (s, ρ) that leads from the initial state to the goal region. Naturally, this sequence must be continuous and contain only valid, collision-free trajectory segments as shown in Figure 5.5.

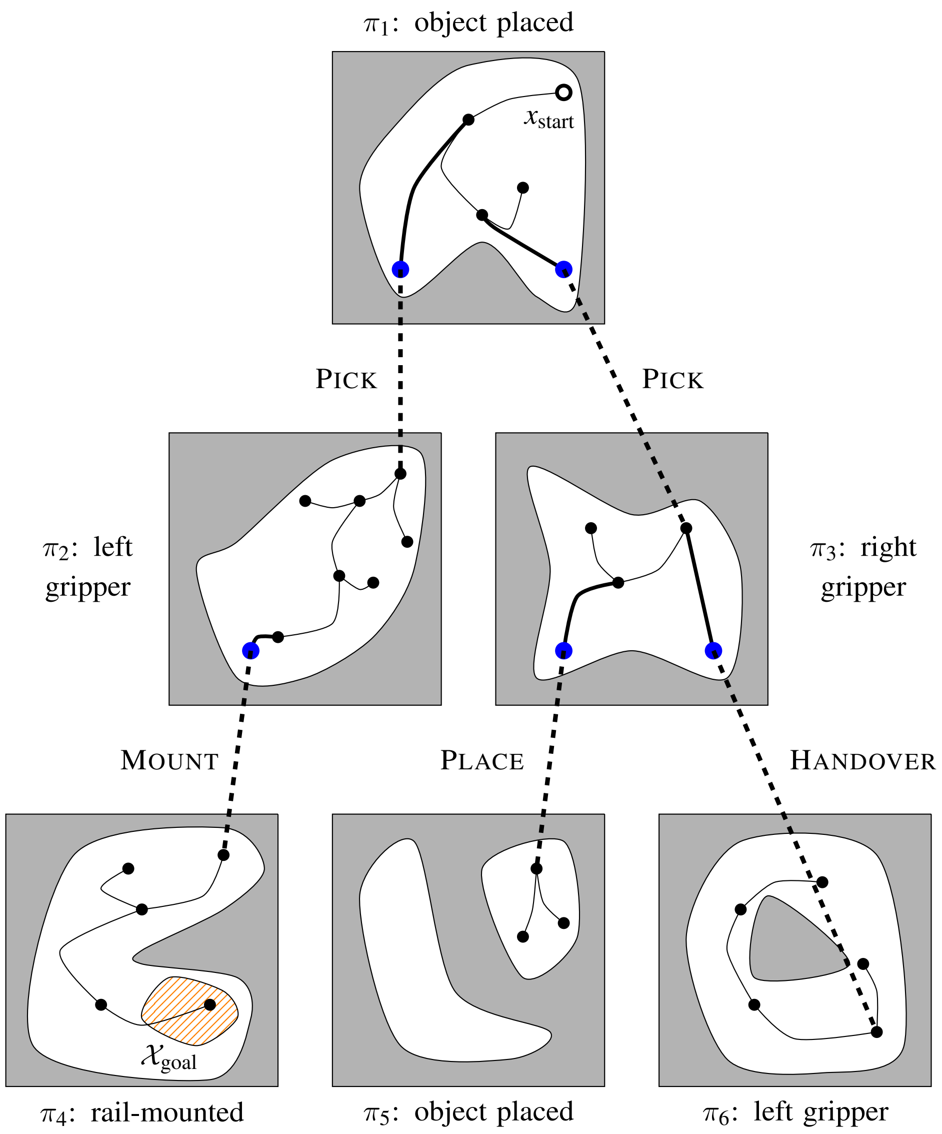


Figure 5.5: Planning problem for manipulation with skills: The boxes visualize different contact states of a component that should be mounted on the top-hat rail. Thin lines visualize free space motions. Obstacles are drawn as gray areas. Skills, such as PICK, HANDOVER, or MOUNT, that affect the object's contact state are drawn as dashed lines. The trajectory segments that lead to the start configuration of a skill $\mathbf{q}_{\text{r,init}}$ are drawn as thick lines. These start configurations $\mathbf{q}_{\text{r,init}}$ are drawn as thick, blue dots.
The trajectory segments corresponding to thin lines of this visualization are created in line 5 of **Skill-EST**. Thick lines are created in line 11. The transitions to a new contact state, drawn as dashed lines, are created in line 13.

5.2.2 Planning Algorithm

```
1  Procedure: Skill-EST($x_{\text{start}}, \mathcal{S}, \mathcal{X}_{\text{goal}}$) infinite version
2  $\mathcal{N} \leftarrow \{x_{\text{start}}\}, \quad \mathcal{E} \leftarrow \{\}$
3  while true do
4      $x \leftarrow (\mathbf{q}_r, \dot{\mathbf{q}}_r, \pi) \leftarrow$ sampleWeighted($\mathcal{N}$)
5      $\tau \leftarrow$ randomMotion($x$)
6      if isValid($\tau, \pi$) then
7          $x_{\text{new}} \leftarrow$ end($\tau, \pi$)
8          $\mathcal{N}$.add($x_{\text{new}}$), $\quad \mathcal{E}$.add(($x, \tau, x_{\text{new}}$))
9      for $(s, \rho) \in$ sampleActions($x$) do
10         $(\sigma_{\text{new}}, \pi_{\text{new}}, \mathbf{q}_{\text{r,init}}) \leftarrow \rho$
11         $\tau \leftarrow$ localPlanner($x, \mathbf{q}_{\text{r,init}}$)
12         $x_{\text{trans}} \leftarrow$ end($\tau, \pi$)
13         $x_{\text{new}} \leftarrow$ simulateSkill($x_{\text{trans}}, s, \rho$)
14         if isValid($\tau, \pi$) and $x_{\text{new}} \neq$ failure then
15             $\mathcal{N}$.add($x_{\text{new}}$), $\quad \mathcal{E}$.add(($x, \tau, s, \rho, x_{\text{new}}$))
```

As a skill is user-defined, a planner using it must treat the simulation and parameter-sampling of this skill as a black box. This creates a planning problem that is similar to kinodynamic motion planning problems. In these problem instances a planner may sample control inputs and simulate the system's behavior by integrating the dynamical evolution of its state given the control input. The sequencing of parameterized skills has similar constraints: For a given (simulated) state the planner may choose a skill, sample a skill-parameterization, and simulate the outcome of executing the skill using the state and parameters.

The proposed planner is an extension of the Expansive Space Tree (EST) planner [6, 82], a kinodynamic motion planner. Pseudo-code of our algorithm is given in procedure **Skill-EST**. To simplify notation this pseudo-code is written as an infinite construction of a search tree without termination. In practice we terminate after a state within $\mathcal{X}_{\text{goal}}$ is found or after a time budget is depleted.

The planner proceeds to build a tree with nodes $\mathcal{N}$ and edges $\mathcal{E}$ rooted in the initial state x_{start}. Edges in this tree contain a robot trajectory and optionally a skill together with its parameterization. At each iteration a random node of the tree is selected using the procedure **sampleWeighted**. This sampling is designed to approximate an even coverage of the state space. A standard method to approximate an ideal weighted sampling is to segment the state space into buckets and sample evenly over buckets. For each skill we sample random parameterizations within procedure **sampleActions**. For each pair of skill

```
1 Procedure: sampleActions($x$)
2 $\mathcal{A} \leftarrow \{\}$
3 for $s \in \mathcal{S}$ do
4     for $\rho \in$ sampleParams($x, s$) do
5         $\mathcal{A}$.add(($s, \rho$))
6 return $\mathcal{A}$
```

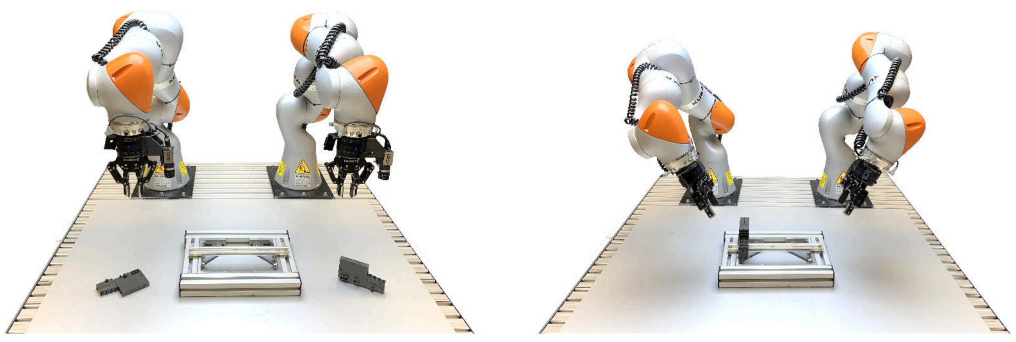
and corresponding parametrization we compute a trajectory to the start configuration of the skill $\mathbf{q}_{\mathrm{r,init}}$ and simulate the outcome of execution. If the simulation is successful the resulting state and action leading to it are added to the tree. This cycle is repeated until a state within $\mathcal{X}_{\mathrm{goal}}$ is found or a time budget depleted. Figure 5.5 visualizes the construction of the search tree of **Skill-EST**.

5.2.3 Controlled Execution

A successful query to our planner returns a sequence of trajectory segments and parameterized skills that can be executed on the physical robot. As each skill is responsible to update the current state of the system a task-level control loop can be implemented. This control loop trivially alternates planning in simulation and physical execution. Pseudo-code for this controller can be found in procedure **Task-Level-Controller**.

The task-level control loop runs at a low frequency leaving dynamic reactions to the currently active skill. In our implementation this loop is typically closed every ten seconds. While each skill is only responsible to be locally robust, the task-level controller is responsible for complex recovery strategies. An example for this would be a failed grasping attempt after which re-localization and manipulation planning are necessary. This example will be discussed in more detail in the next sections.

```
1 Procedure: Task-Level-Controller($x_{\mathrm{start}}, \mathcal{S}, \mathcal{X}_{\mathrm{goal}}$)
2 $x_{\mathrm{current}} \leftarrow x_{\mathrm{start}}$
3 while $x_{\mathrm{current}} \notin \mathcal{X}_{\mathrm{goal}}$ do
4     plan $\leftarrow$ Skill-EST($x_{\mathrm{current}}, \mathcal{S}, \mathcal{X}_{\mathrm{goal}}$)
5     if plan == failure then
6         return failure
7     $x_{\mathrm{current}} \leftarrow$ plan.executeFirstAction()
8 return success
```



(a) Initial state **(b)** Goal state

Figure 5.6: Experimental setup: The left image shows the initial state, where two electrical components are placed on a surface. Their pose is known only approximately to the robot. The task for the robot is to mount the components onto the rail as shown in the right image. To do so, the robot must determine an accurate pose estimate by moving the cameras above the objects. After picking and mounting the components on the rail the robot needs to push the components together in order to connect them.

5.2.4 Implementation Details and Experimental Setup

To validate our approach, we conducted a series of experiments both on a real robot and in simulation. In this section we describe the experiments that explore the practical applicability of the proposed skill model and planner.

These experiments are designed to answer three questions: (1) Is the proposed model of a skill capable of modeling industrial manipulation tasks? (2) Can new skills be added with an acceptable amount of engineering and without adding dependencies to previously written skills? (3) And finally, does planning at the center of our task-level control loop provide an acceptable runtime? At this point we focus mainly on architectural questions. A comparison with existing task and motion planners regarding runtime and quality of solutions is postponed to the next section.

For our experiments, we used the dual-arm setup that is shown in Figure 5.6. Each arm is equipped with a parallel gripper and a wrist-mounted camera to localize parts. The task of the robot is to assemble two components onto a top-hat rail in the center of the setup. To complete the task the following steps are necessary. First, all parts must be localized using the cameras. Then both components must be mounted onto the rail. As the left component is placed sideways, it is necessary to grasp it with the left gripper and to perform a handover to the right robot to change the orientation of the grasp. Finally, the two components must pushed together on the rail to be connected.

This task is a challenging planning problem on multiple levels. About 45% of the robot's configuration space is in collision. To pick and assemble parts, the gripper must move into close proximity of the objects, which forms tunnels in the collision-free configuration space. The necessity for a handover is not specified symbolically but only in the form of the collision constraints. This requires manipulation planning. Finally, the order in which parts are localized or grasped is up to the planner and a component that is assembled in the wrong position on the rail leads to a dead end. On the symbolic level our planning problem is more complex than typical manipulation planning problems but less complex than typical task planning problems.

For this setup we implemented a set of skills visualized in Figure 5.7. These can be divided into skills that require only one arm and those, that are multi-arm skills. During planning and execution, the skills for a single arm can be used in parallel in arbitrary combinations. The single-arm skills include: MOUNT, PERCEIVE, and PICK. As multi-arm skills we implemented: HANDOVER and PUSH. In the last skill two robots push the components on the rail to connect them. This is an example of a robotic skill that we do not expect to be reusable in another application but was required for the task at hand. It therefore challenges the modularity of our approach.

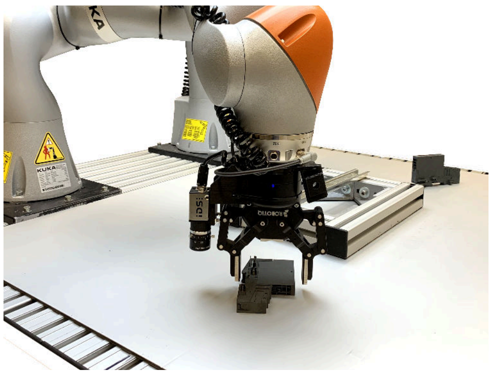
The task was given to our system with three different planning horizons. With the shortest planning horizon our system received a sequence of four goal states: (1) All objects are located. (2) The first component is mounted on the rail. (3) Both components are mounted on the rail. (4) The components are pushed together. For the intermediate planning horizon only goals (2) and (4) were given. The longest planning horizon consisted only of the final goal (4) with no intermediate goals.

Two changes were made to the high-level control loop to speed up planning and execution times. The re-planning was not triggered after every skill, but only when deviations with respect to the expected system state occur. Furthermore, we concatenated multiple sequential random motions, that are not interrupted by a skill, and optimize the results with the algorithm presented by Hauser and Ng-Thow-Hing [83].

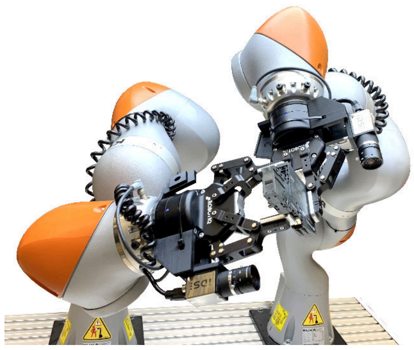
This experiment was run 60 times for the three specifications of the task in simulation with different random seeds for the planner. The simulator used the same limits on acceleration and velocity as the real robot. Therefore, the execution times measured in our experiments match those of a real execution except for image processing and the opening and closing of the grippers. Our planner was implemented to make use of multiple threads and the experiments were run on a quad-core Intel i7-6820HQ CPU.

5.2.5 Results

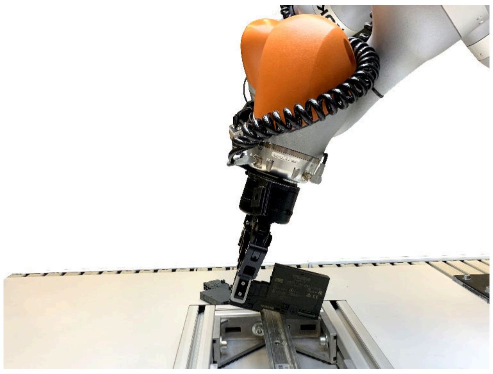
Qualitative Results - Modeling With the exception of the PUSH skill, the assembly task was implemented with skills that arguably are reusable across typical kinematic structures and a variety of manipulation tasks. The implementation of the PUSH skill, which is entirely application specific, was done without modification of other skills. On the physical robot our approach can reliably control the assembly of the benchmark task for different initial configurations of the parts and with disturbances during execution.



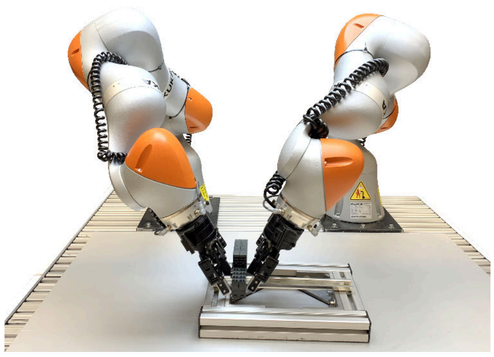
(a) PICK and PLACE



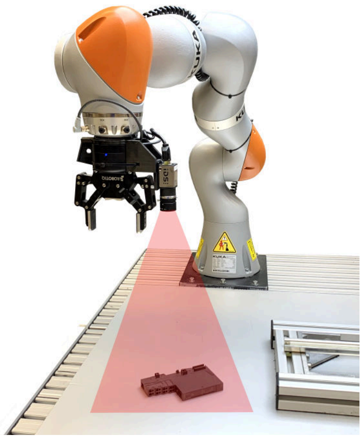
(b) HANDOVER



(c) MOUNT



(d) PUSH



(e) PERCEIVE

Figure 5.7: Implemented Skills: PICK, PLACE, and HANDOVER enable the actions typically needed for manipulation planning. The MOUNT skill is used to assemble components onto a rail. With the PUSH skill two components that are mounted on a rail can be connected. The PERCEIVE skill does not involve any motion of the robot but changes the belief state regarding the objects.

Prior to the experiments we implemented the SkiROS [84] skill architecture for a different assembly scenario. In comparison to SkiROS two observations can be made. On one hand, the modeling of skills is easier than within the SkiROS architecture. This is due to the fact, that our skill model requires local behavior only. To implement a skill it is not necessary to model the use of a motion planner and the control flow can be mostly linear. Recovery from failure is handled by the planner. On the other hand, the specification of tasks by the user is more complex, as it is less intuitive to specify the goal state than to write down a sequence of skills parameterized with task-level variables.

Qualitative Results - Behavior The separation between local skills and task and motion planner resulted in complex behaviors of the robot both during nominal execution and during automatically planned recovery. An example is the recovery strategy after a failed grasping attempt. During execution the planner decided to grasp and mount the component on the right in Figure 5.1. This component was accidentally pushed over and the grasping failed. The reaction of the pick skill was simply to change the belief state to have an uncertain pose of the object. In this state the planner decided to re-localize and pick the object before mounting it. After the perception determined that the object was pushed over, the planner automatically computed a manipulation sequence that included a handover to change the orientation of the object.

Quantitative Results Figure 5.8 shows the average cumulative planning and execution times of the task. For the task specification with short or intermediate horizon our system spent 11.9% or 13.7% of the total time to execute a task on planning. For the task specification with long planning horizon 52% was spent on planning.

Table 5.1 shows the average, minimum, and maximum times for planning and executing the entire task as well as for one iteration of our control loop. On average re-planning is triggered every 8.5 to 9.5 seconds. For the short and intermediate horizon, re-planning takes 1.1 and 1.3 seconds on average. For the long time horizon re-planning takes 10.4 seconds on average. These results indicate, that our system is well suited to execute a sequence of complex manipulation tasks but does not yet scale to tasks with long time horizons.

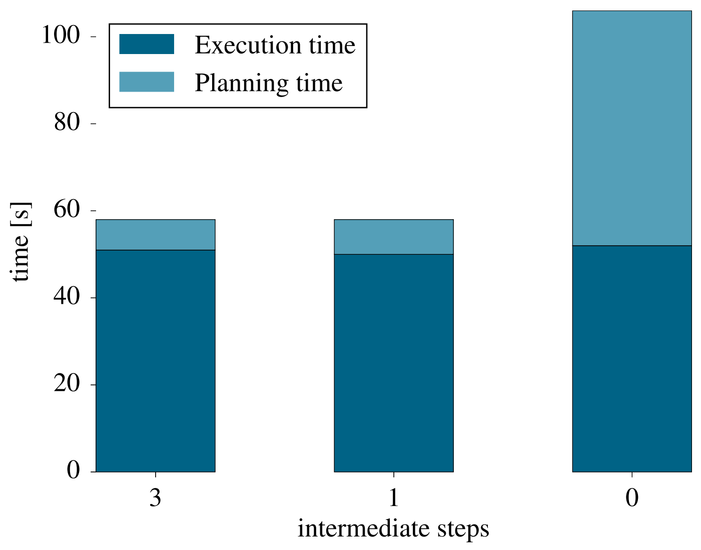


Figure 5.8: Average cumulative planning and execution times for different specifications of the task: The task of the robot is given to the planner with different time horizons, i. e., with a different number of intermediate goals.

Table 5.1: Planning and Execution times

		Intermediate steps		
		3	1	0
	min	46.9	47.3	48.3
Execution Time per task [s]	mean	50.9	50.6	51.8
	max	54.8	55.3	57.5
	min	6.0	5.9	24.6
Planning Time per task [s]	mean	6.9	8.0	56.4
	max	8.5	12.1	120.6
	min	1.3	1.4	1.6
Execution Time per cycle [s]	mean	8.5	8.5	9.5
	max	22.0	24.1	24.3
	min	1.0	1.0	1.0
Planning Time per cycle [s]	mean	1.1	1.3	10.4
	max	3.3	5.9	52.5

5.3 Optimal Planning with Skills

The previous section focused on architectural questions related to task and motion planning. We omitted a formal model of the planning problem and an analysis of the proposed planner. Furthermore, the planner of the previous section, **Skill-EST**, is not able to compute optimal plans.

These issues are addressed in this section. To this end, we reformulate the planning problem as a multi-modal motion planning problem as presented by Hauser and Latombe [11, 28] in Section 5.3.1. We discuss the relation to the proposed model of robotic skills and reformulate the **Skill-EST** planner into its multi-modal counterpart: Multi-Modal-EST (**MM-EST**).

Based on the multi-modal formulation of the problem it is now possible to utilize the AO-x meta algorithm proposed by Hauser and Zhou [27] to repeatedly plan in a cost-augmented state space. This results in an asymptotically optimal planner that is presented in Section 5.3.2.

We present an informed search strategy in Section 5.3.3 and analyze probabilistic completeness and asymptotic optimality in Section 5.3.4. In extensive simulative experiments we compare the proposed planners to an approach from the literature. The setup and the results of these experiments are presented in Section 5.3.5 and Section 5.3.6 respectively.

5.3.1 Reformulation as Multi-Modal Planning Problem

In this section we formulate task and motion planning as a multi-modal motion planning problem similar to Hauser and Latombe [11, 28] and present a planner for this problem. We show how such a planning problem can be modeled using the skills presented in the previous section.

Multi-Modal Problem Statement We consider the multi-modal planning problem, where the state $x \in \mathcal{X}$ of a robot and its environment is represented by $x = (\mathbf{q}_\mathrm{r}, \dot{\mathbf{q}}_\mathrm{r}, \pi)$. The robot configuration $\mathbf{q}_\mathrm{r}$ denotes the positions of the robot axes and $\dot{\mathbf{q}}_\mathrm{r}$ their velocities. All axis positions and velocities are assumed to be upper- and lower-bounded:

$$\mathbf{q}_\mathrm{r,min} \leq \mathbf{q}_\mathrm{r} \leq \mathbf{q}_\mathrm{r,max}, \qquad -\dot{\mathbf{q}}_\mathrm{r,max} \leq \dot{\mathbf{q}}_\mathrm{r} \leq \dot{\mathbf{q}}_\mathrm{r,max}. \tag{5.1}$$

We assume that the acceleration of the robot configuration $\ddot{\mathbf{q}}_\mathrm{r}$ is directly controllable and upper and lower bounded:

$$-\ddot{\mathbf{q}}_\mathrm{r,max} \leq \ddot{\mathbf{q}}_\mathrm{r} \leq \ddot{\mathbf{q}}_\mathrm{r,max}. \tag{5.2}$$

The contact parameter $\pi \in \Pi$ denotes the discrete component of the state space. This contact parameter encodes both the symbolic state of the world (e.g., which object is grasped) as well as its continuous parameterization (e.g., the transform from gripper to object). The set Π of modes contains a finite but potentially very large number of modes π. Assuming a finite set of contact states is a restriction that limits the methods of this section to a resolution completeness within the contact parameters.

The contact parameter π remains constant, unless the robot configuration $\mathbf{q}_{\mathrm{r},1}$ reaches a transition region $\mathbf{q}_{\mathrm{r},1} \in \mathcal{Q}_{\pi_1,\pi_2}$, where a change from contact π_1 to π_2 is allowed. An example for such a transition region would be the set of solutions for the inverse kinematics problem of grasping an object with a given gripper-to-object transform. We assume that the axis velocities must be zero at the transition. Within such a transition region a state $x_1 = (\mathbf{q}_{\mathrm{r},1}, \mathbf{0}, \pi_1)$ can change to a new state $x_2 = (T_{\pi_1,\pi_2}(\mathbf{q}_{\mathrm{r},1}), \mathbf{0}, \pi_2)$. Here, $T_{\pi_1,\pi_2}(\mathbf{q}_{\mathrm{r},1})$ denotes the robot configuration after the transition. An example of this would be a state after retracting from a grasp. Typically the transition regions $\mathcal{Q}_{\pi_1,\pi_2}$ form lower-dimensional manifolds of the robot configuration space $\mathcal{Q}_\mathrm{r}$ and are empty for most pairs of $\pi_1, \pi_2 \in \Pi$. The set of neighbor contacts Neighbors(π) of a contact π contains all contacts π' for which $\mathcal{Q}_{\pi,\pi'}$ is not empty. At all times the robot must stay within the valid, i. e., at least collision-free, portion of the configuration space $\mathcal{Q}_{\mathrm{free},\pi} \subset \mathcal{Q}_\mathrm{r}$.

Let $\tau : [0,t] \to \mathcal{Q}_\mathrm{r}$ be a continuous path segment in the robot configuration space for which the contact remains constant. A path $\{(\tau_i, \pi_i)\}_{i \leq k}$ with $i, k \in \mathbb{N}_{>0}$ is a sequence of k path segments and contact parameters. We define valid paths as paths that (1) respect the limits on axis positions, velocities, and accelerations, (2) are collision-free, and (3) contain correct mode transitions.

Definition 8 (Valid Path) *A path is valid iff*

1. *$\tau_i(t)$ respects Eq. (5.1) and Eq. (5.2) for $t \in [0, t_i], i \in 1...k$,*
2. *$\tau_i(t) \in \mathcal{Q}_{\mathrm{free},\pi_i}$ for $t \in [0, t_i], i \in 1...k$,*
3. *$\tau_i(t_i) \in \mathcal{Q}_{\pi_i,\pi_{i+1}}$,*
   $\dot\tau_i(t_i) = \dot\tau_{i+1}(0) = \mathbf{0}$, and
   $\tau_{i+1}(0) = T_{\pi_i,\pi_{i+1}}(\tau_i(t_i))$ for $i \in 1...k-1$.

Feasible paths are valid paths, that start in a given initial state $x_\mathrm{start} \in \mathcal{X}$ and end in a goal region $\mathcal{X}_\mathrm{goal} \subset \mathcal{X}$.

Definition 9 (Feasible Path) *A path is feasible iff*

1. *it is valid,*
2. *$(\tau_1(0), \dot\tau_1(0), \pi_1) = x_\mathrm{start}$, and*
3. *$(\tau_k(t_k), \dot\tau_k(t_k), \pi_k) \in \mathcal{X}_\mathrm{goal}$.*

To define the objective for optimization, we introduce the cost-function C as follows:

$$C(\{(\tau_i,\pi_i)\}_{i\leq k}) = \sum_{i=1}^{k} C_p(\tau_i) + \sum_{i=1}^{k-1} C_t(\pi_i,\pi_{i+1}) \tag{5.3}$$

$$C_p(\tau) = \int_0^t L(\tau(v))dv, \tag{5.4}$$

where C_p assigns non-negative costs to path segments, with $L_{max} > L(\cdot) > L_{min} > 0$ as incremental costs and $C_t > C_{t,min} > 0$ assigns lower bounded positive cost to transitions. Let Ξ_f be the set of feasible paths. The optimal cost c^* is defined as:

Definition 10 (Optimal Cost) $c^* = \inf\limits_{\xi\in\Xi_f} C(\xi)$.

Primitive Operations We assume that a set of primitive operations is available to our planner. The procedure **randomMotion**(x) returns a random trajectory $\tau : [0, t_{\text{end}}] \to \mathcal{Q}_{\text{r}}$ of robot configurations. This trajectory is twice differentiable and its initial positions and velocities match that of the state x. A possible implementation is to integrate random accelerations $\ddot{\mathbf{q}}_{\text{r}}$. As the transition regions $\mathcal{Q}_{\pi_1,\pi_2}$ are lower-dimensional manifolds of the robot configuration space $\mathcal{Q}_{\text{r}}$ there is zero probability of randomly moving into such a region. To enable mode transitions with positive probability the procedure **randomTransition**(x, π') computes a robot trajectory that steers the robot from the state $x = (\mathbf{q}_{\text{r}}, \dot{\mathbf{q}}_{\text{r}}, \pi)$ into the transition region $\mathcal{Q}_{\pi,\pi'}$. Finally, **isValid**(x, τ) determines whether a trajectory moves the robot on a valid trajectory within $\mathcal{Q}_{\text{free},\pi}$.

Relation to the Skill Model An implementation of the multi-modal planning problem and the corresponding primitive operations requires functions that can be separated into two categories:

The first category relates to functions typically required for collision-free motion planning. This includes procedures like **randomMotion** that generate and integrate random accelerations and a collision detection in the procedure **isValid**. Implementations of these functions are available, e. g., in the MoveIt! framework [19].

The second category relates to changes of contact state. It includes the set of contacts Π, the transition regions $\mathcal{Q}_{\pi_1,\pi_2}$, the set of neighboring contacts Neighbors(π), the transition function $T_{\pi_1,\pi_2}(\cdot)$, and the procedure **randomTransition**.

This second category is challenging to implement as an explicit model, especially in a context that requires the inclusion of domain knowledge as in the assembly scenario. For this reason we use an implicit approach. Instead of actually representing sets like the transition regions $\mathcal{Q}_{\pi_1,\pi_2}$ we draw samples from this set using the motion parametrization of a skill as presented in the previous section. The procedure **randomTransition** can then be implemented with such a sampling and a local planner. In the same spirit the set of

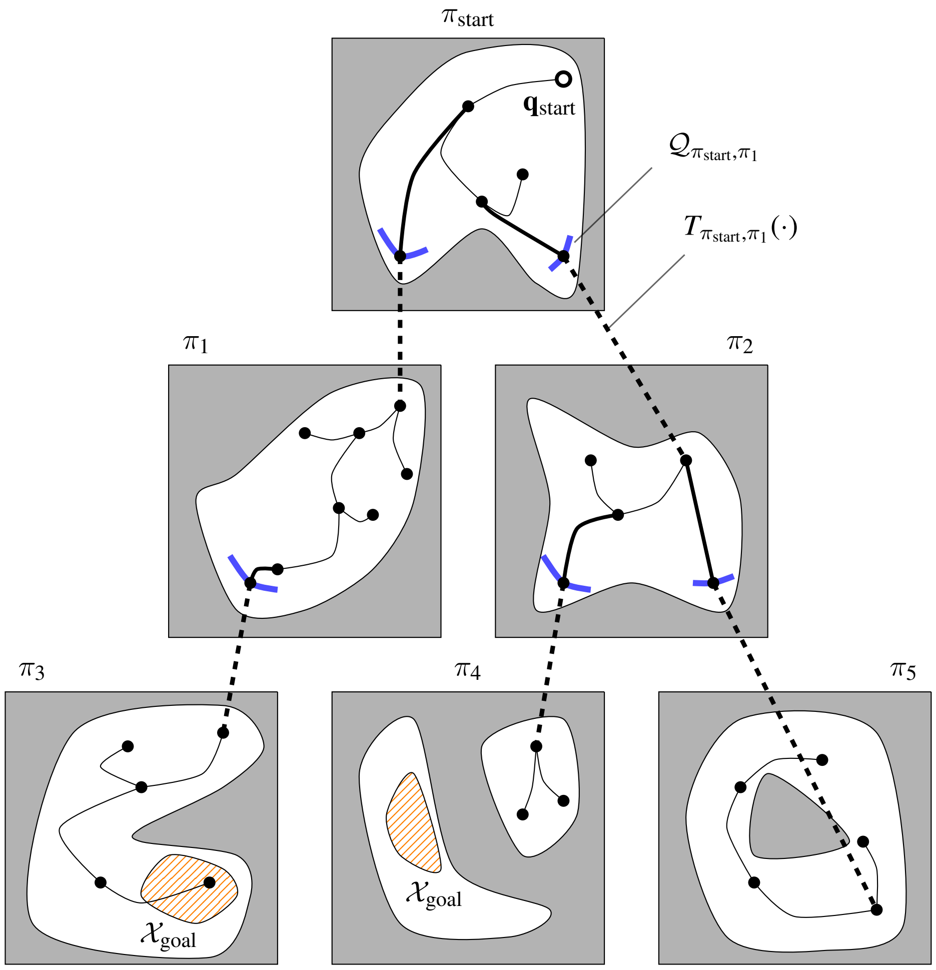

π_{start}
$\mathbf{q}_{\text{start}}$
$\mathcal{Q}_{\pi_{\text{start}},\pi_1}$
$T_{\pi_{\text{start}},\pi_1}(\cdot)$
π_1
π_2
π_3
π_4
π_5
$\mathcal{X}_{\text{goal}}$
$\mathcal{X}_{\text{goal}}$

Figure 5.9: Multi-modal planning problem and tree construction of MM-EST: The boxes visualize the robot configuration space $\mathcal{Q}_{\text{r}}$ for different contact states π. White areas show the free space $\mathcal{Q}_{\text{free},\pi}$. Thin black lines visualize motions computed using **randomMotion**, thick black lines those computed using **randomTransition**. Transition areas $\mathcal{Q}_{\pi_1,\pi_2}$ are drawn as blue lines.

neighboring contacts Neighbors(π) is not represented explicitly but one can draw samples from it using the symbolic parametrization of a skill. The transition function $T_{\pi_1,\pi_2}(\cdot)$ is implemented via the simulation of a skill.

This indirect approach is in line with the inner workings of sampling-based motion planners. Formally, motion planning is based on a notion of a configuration space and valid regions within this space. These two concepts are however not implemented in typical motion planners. Instead, procedures to sample and to reject samples are used during planning. In the same spirit we construct an implicit, multi-modal planning problem using a set of skills.

Multi-Modal Planner Based on the multi-modal problem we can reformulate the planner from the previous section. As we extend the Expansive Space Tree (EST) planner to multi-modal domains we call it Multi-Modal-EST (MM-EST). The algorithm is described in procedure **MM-EST**.

```
Procedure: MM-EST($x_{\text{start}}$, $\mathcal{X}_{\text{goal}}$)
$\mathcal{N} \leftarrow \{x_{\text{start}}\}$
$\mathcal{E} \leftarrow \{\}$
while withinTimeBudget() do
    $x \leftarrow$ sampleWeighted($\mathcal{N}$)
    for $\tau$ in sampleActions($x$) do
        $x_{\text{new}} \leftarrow$ integrate($x, \tau$)
        if isValid($x, \tau$) then
            $\mathcal{N}$.append($x_{\text{new}}$)
            $\mathcal{E}$.append(($x$, $\tau$, $x_{\text{new}}$))
            if $x_{\text{new}} \in \mathcal{X}_{\text{goal}}$ then
                plan $\leftarrow$ path to $x_{\text{new}}$
                return plan
return failure
```

Our algorithm proceeds to build a tree of nodes $\mathcal{N}$ and edges $\mathcal{E}$ rooted in the initial state x_{start} until the goal-region $\mathcal{X}_{\text{goal}}$ is reached or the time budget is depleted. In each iteration the planner samples a random node within the currently built tree using the procedure **sampleWeighted**. This weighted sampling puts larger weight on nodes that lie in sparsely populated areas of the state-space. For this node a set of robot trajectories is sampled at random using the procedure **sampleActions**. For each random trajectory τ, the procedure **integrate** computes the resulting state x_{new}. This may include a transition to a new contact parameter. If such a state and the trajectory leading to it pass validity checks it is added to the tree.

The difference to the EST algorithm lies within the procedure **sampleActions**. Instead of only choosing random control inputs, the procedure additionally steers towards all neighboring contact parameters using **randomTransition**. This is similar to a goal bias of 50% for each neighboring contact parameter. Figure 5.9 illustrates the tree construction of MM-EST. As more time is spent on building the tree, the planner will attempt to explore the configuration space of already reached contact parameters using **randomMotion**. This increases the chance of finding a collision-free control input to explore new contact parameters with **randomTransition**.

It is important to note that MM-EST does not rely on collision-free, sampling-based motion planners to reach new contacts. The exploring and the changing of contact parameters are treated as equal actions within the tree construction. This is a key difference to the

PTR [11], HBF [35], and Random-MMP [29] planners and can be leveraged for optimal planning as shown in the next section.

```
Procedure: sampleActions($x$)
$(\mathbf{q}_\mathrm{r}, \pi) \leftarrow x$
actions $\leftarrow$ {randomMotion()}
for $\pi'$ in Neighbors($\pi$) do
    actions.append(randomTransition($x, \pi'$))
return actions
```

5.3.2 Optimal Planner

As we have introduced a kinodynamic task and motion planner we can use the meta-algorithm Asymptotically-Optimal-x (AO-x) introduced by Hauser and Zhou [27] to create a new, optimal planner: AO-MM-EST. This planner is called after an initial plan has been found with **MM-EST**. The lowest cost of any plan that was computed is stored as c_best.

```
Procedure: AO-MM-EST($x_\mathrm{start}, \mathcal{X}_\mathrm{goal}$, plan)
$c_\mathrm{best} \leftarrow$ plan.getCost()
$\mathcal{N} \leftarrow \{(x_\mathrm{start}, 0)\}$
$\mathcal{E} \leftarrow \{\}$
while withinTimeBudget() do
    $(x, c_x) \leftarrow$ sampleWeighted($\mathcal{N}$)
    for $\tau$ in sampleActions(x) do
        $x_\mathrm{new} \leftarrow$ integrate($x, \tau$)
        $c_\mathrm{new} \leftarrow c_x +$ cost($x, \tau$)
        if $c_\mathrm{new} < c_\mathrm{best}$ and isValid($x, \tau$) then
            $\mathcal{N}$.append(($x_\mathrm{new}, c_\mathrm{new}$))
            $\mathcal{E}$.append((($x, c_x$), $\tau$, ($x_\mathrm{new}, c_\mathrm{new}$)))
            if $x_\mathrm{new} \in \mathcal{X}_\mathrm{goal}$ then
                $c_\mathrm{best} \leftarrow c_\mathrm{new}$
                plan $\leftarrow$ path to $x_\mathrm{new}$
                $\mathcal{N}, \mathcal{E} \leftarrow$ prune($\mathcal{N}, \mathcal{E}, c_\mathrm{best}$)
return plan
```

The AO-x meta algorithm requires a kinodynamic planner and a state space. In our case the planner is MM-EST and $\mathcal{X}$ the state space. The state-space is augmented with an additional dimension for cost: $\mathcal{X} \times [0, c_\mathrm{best})$. AO-x then uses the kinodynamic planner to plan in the cost-augmented space with the restriction that all reached state-cost pairs (x, c) with $c \geq c_{\text{best}}$ are considered invalid. Once a new state in the goal region is found, a better plan must have been found. With this new plan the cost c_{best} is updated and the process repeated.

It is important to note that AO-x does not simply call a planner repeatedly to keep track of the best solution. Instead, the cost-augmented state space is densely explored. Hauser and Zhou [27] show that this highly generic approach is surprisingly effective and that it outperforms a variety of state-of-the-art optimizing motion planners.

In procedure **AO-MM-EST** this combination of MM-EST and AO-x is shown. As can be seen, this algorithm is identical to MM-EST except for three changes: (1) The planner operates in a state-cost space. (2) Additionally to checking whether trajectories are valid the planner checks if the cost of the current best solution is exceeded (Line 10). (3) After a new solution is found the current search tree is pruned (Line 16). This operation removes all nodes and edges from the tree where the new c_{best} is exceeded.

5.3.3 Informed, Optimal Planner

By default the AO-x meta algorithm causes a kinodynamic motion planner to densely explore the entire state-cost space. This means that, in the limit, it finds optimal paths to all reachable states, not just to states in the goal area. Even more so, it not only finds optimal paths for all reachable states, but also solutions with cost up to c_{best}.

As this is highly inefficient Hauser and Zhou [27] propose an informed search strategy. Let $c^*(x)$ be the optimal cost-to-go from a state x to the goal region if it is reachable and $+\infty$ otherwise. If a conservative cost-to-go estimate $0 \leq h(x) \leq c^*(x)$ is available, it is immediately clear that any state-cost pair (x, c_x) with $c_x + h(x) \geq c_{\text{best}}$ cannot lead to a solution that improves over c_{best}. Instead of discarding all states with $c_x \geq c_{\text{best}}$ we can now safely discard all states with $c_x + h(x) \geq c_{\text{best}}$. This cost-to-go estimate may also be used to remove larger fractions of the search tree in the **prune** procedure. Also, during the initial feasible planning with **MM-EST** states with $h(x) = +\infty$ can be safely discarded as dead ends.

For many relevant motion planning problems a suitable cost-to-go estimate is available, e. g., the euclidean distance to a goal. The question is now: How can we construct a suitable cost-to-go estimate for a multi-modal planning problem?

Our reasoning to construct such a cost-to-go estimate for multi-modal planning problems is to only consider the symbolic aspects of a planning problem. If we assume a simplified planning problem in which no collisions can occur and motions that do not change the contact state are free of cost, the optimal cost-to-go $\hat{c}^*(x)$ of this planning problem is an admissible cost-to-go estimate of our original planning problem. This leads to a discrete planning problem where the state-space is the set of contact parameters Π and the start state π_{start} is the contact parameter of the current state. The set of goals is the

```
1  Procedure: MM-EST*($x_{\text{start}}, \mathcal{X}_{\text{goal}}$, plan)
2  $c_{\text{best}} \leftarrow$ plan.getCost()
3  $\mathcal{N} \leftarrow \{(x_{\text{start}}, 0)\}$
4  $\mathcal{E} \leftarrow \{\}$
5  while withinTimeBudget() do
6      $(x, c_x) \leftarrow$ sampleWeighted($\mathcal{N}$)
7      for $\tau$ in sampleActions(x) do
8          $x_{\text{new}} \leftarrow$ integrate($x, \tau$)
9          $c_{\text{new}} \leftarrow c_x +$ cost($x, \tau$)
10         if $c_{\text{new}} + h(x_{\text{new}}) < c_{\text{best}}$ and isValid($x, \tau$) then
11             $\mathcal{N}$.append(($x_{\text{new}}, c_{\text{new}}$))
12             $\mathcal{E}$.append((($x, c_x$), $\tau$, ($x_{\text{new}}, c_{\text{new}}$)))
13             if $x_{\text{new}} \in \mathcal{X}_{\text{goal}}$ then
14                 $c_{\text{best}} \leftarrow c_{\text{new}}$
15                 plan $\leftarrow$ path to $x_{\text{new}}$
16                 $\mathcal{N}, \mathcal{E} \leftarrow$ prune($\mathcal{N}, \mathcal{E}, c_{\text{best}}$)
17 return plan
```

set of all contacts that intersect with the goal region. The actions available in a state are determined by the set of neighboring contacts Neighbors(π) and lead to the corresponding costs $C_t(\pi, \pi')$. Dijkstra's algorithm can efficiently compute this cost-to-go estimate for each reachable contact and the estimate can be stored for later use. Pseudocode for the informed variant of our planner is shown in procedure **MM-EST***. The only change relative to AO-MM-EST is the use of the cost-to-go estimate in line 10.

5.3.4 Completeness and Optimality

For the analysis of **MM-EST** and **MM-EST*** we assume three properties about our multi-modal planning domain: (1) α, β-expansiveness, (2) ideal sampling, and (3) the existence of a γ, δ-transition tunnel.

1. The definition of α, β-expansiveness as proposed by Hsu *et al.*can be found in [6] and we assume, that this property holds for every contact parameter of the planning problem. We do not require it to hold across transitions.

2. Our definition of ideal sampling extends that of Hsu *et al.* [6] to multi-modal domains. It refers to both the procedure **sampleWeighted** and **randomMotion**. An ideal sampler selects nodes in such a way, that all currently reached modes are sampled with equal probability and each tree node will be selected infinitely often in the limit of iterations. Furthermore, the currently reachable subset of a mode is

sampled uniformly by the combination of **sampleWeighted** and **randomMotion**. In section Section 5.3.5 we discuss how to approximate ideal sampling.

3. Intuitively, a γ, δ-transition tunnel implies, that there exists a sequence of contacts from start state to goal region that allows a sampling-based motion planner and sampling-based transitioning to go through a sequence of transition manifolds. For one contact π_i in the sequence we start in a set $\mathcal{X}_{\text{entry},i}$ from which the states $\mathcal{X}_{\text{reach},i}$ are locally reachable. From these states the set $\mathcal{X}_{\text{exit},i}$ is reachable without transitions. A transition from $\mathcal{X}_{\text{exit},i}$ leads to $\mathcal{X}_{\text{entry},i+1}$ with probability of at least δ. Let $\mu(S)$ denote the volume of a subset $S \subset \mathcal{X}$ of the state space. $\mathcal{R}_l(x)$ denotes the locally reachable set of points of state x with one call to **randomMotion** (no transitions).

Definition 11 (γ, δ-transition tunnel) *A sequence* $\{\pi_i, \mathcal{X}_{\text{entry},i}, \mathcal{X}_{\text{reach},i}, \mathcal{X}_{\text{exit},i}\}_{1 \leq i \leq k}$ *is a* γ, δ-*transition tunnel iff:*

- $\mathcal{X}_{\text{entry},1} = \{x_{\text{start}}\}$
- $\mathcal{X}_{\text{exit},k} \subset \mathcal{X}_{\text{goal}}$
- $\mathcal{X}_{\text{entry},i}, \mathcal{X}_{\text{reach},i}, \mathcal{X}_{\text{exit},i} \subset \{x = (\mathbf{q}_r, \dot{\mathbf{q}}_r, \pi) \in \mathcal{X} \mid \pi = \pi_i\}$
- $\mu(\mathcal{X}_{\text{reach},i}) \geq \gamma, \; \mu(\mathcal{X}_{\text{exit},i}) \geq \gamma$
- *Every point of* $\mathcal{X}_{\text{exit},i}$ *is reachable from every point within* $\mathcal{X}_{\text{reach},i}$ *without transitions.*
- *For* $x \in \mathcal{X}_{\text{exit},i}$ ***randomTransition***(x, π_{i+1}) *leads to* $\mathcal{X}_{\text{entry},i+1}$ *with probability of at least* δ *for* $i \leq k-1$.
- $\mathcal{X}_{\text{reach},i} \subset \bigcap_{x \in \mathcal{X}_{\text{entry},i}} \mathcal{R}_l(x)$

The definition of the γ, δ-transition tunnel relies on properties of the sampling distribution of the procedure **sampleTransition**. It is therefore dependent on the implementation of this procedure. In section Section 5.3.5 we will show an implementation of **sampleTransition** that ensures the existence of a γ, δ-transition tunnel.

Theorem 3 (probabilistic completeness of MM-EST) *If the configuration space in every contact state is* α, β*-expansive, a* γ, δ*-transition tunnel exists, and an ideal sampler is used then MM-EST is probabilistically complete.*

Proof. Without loss of generality we assume, that the volume of the state-space for only one contact parameter has been scaled to have unit volume. The number of contact parameters is denoted as $N = |\Pi|$. The idea of the proof is to show that **MM-EST** proceeds along the transition tunnel with probability one in the limit of iterations.

By definition of the transition tunnel, x_{start} lies within $\mathcal{X}_{\text{entry},1}$. Let us assume we have sampled a state within $\mathcal{X}_{\text{entry},i}$. Then the ideal sampler will proceed to sample random motions with at least probability γ/N in the connected component of the state space that leads from $\mathcal{X}_{\text{entry},i}$ over $\mathcal{X}_{\text{reach},i}$ to $\mathcal{X}_{\text{exit},i}$. This is due to the fact, that $\mu(\mathcal{X}_{\text{reach},i}) \geq \gamma$. Each such sample is equivalent to planning with the EST planner towards $\mathcal{X}_{\text{exit},i}$. As we assume α, β-expansiveness of each contact state and $\mu(\mathcal{X}_{\text{exit},i}) \geq \gamma$ the proof of [6] tells us that the probability of reaching $\mathcal{X}_{\text{exit},i}$ converges towards one. Once we have reached $\mathcal{X}_{\text{exit},i}$ there is a δ probability of transitioning towards $\mathcal{X}_{\text{entry},i+1}$ if $i < k$ when selecting a node in this region. As all nodes are selected infinitely often by the ideal sampler in the limit, the probability of reaching contact $i + 1$ and thus contact k converges towards one. Once we sample a state within $\mathcal{X}_{\text{exit},k}$, which we know happens with probability converging towards one, we reach a goal state. Thus, **MM-EST** reaches the goal with probability one in the limit of iterations and is probabilistically complete. ■

Let c_n be the cost returned by MM-EST* after n iterations.

Theorem 4 (Optimality of MM-EST*) *If for every $\epsilon > 0$ every contact in the state-cost space $\mathcal{X} \times [0, c^* + \epsilon]$ is α, β-expansive, a γ, δ-transition tunnel exists towards the goal $\mathcal{X}_{\text{goal}} \times [0, c^* + \epsilon]$, and an ideal sampler is used then c_n converges in probability towards c^*.*

$$\lim_{n \to \infty} P(c_n - c^* > \epsilon) = 0$$

Proof. **MM-EST*** is started after a solution has been found by **MM-EST** and plans in a cost augmented state space $\mathcal{X} \times [0, c_{\text{best}}]$ where c_{best} is the cost of the previously found solution. Let us assume for a moment that we do not reduce c_{best} once a better solution is found and that $h(x) = 0$.

Let $0 < \epsilon < c_{\text{best}} - c^*$. From the existence of a γ, δ-transition tunnel towards a goal region $\mathcal{X}_{\text{goal}} \times [0, c^* + \epsilon]$ and the proof of probabilistic completeness above we know that the probability of sampling a state with cost of less than $c^* + \epsilon$ converges towards one.

This does not change if c_{best} is reduced or an admissible cost-to-go estimate $h(x)$ is used. Both a reduction of c_{best} or the use of $h(x)$ can only affect the γ, δ-transition tunnel if c_{best} is reduced below $c^* + \epsilon$. This only happens after a solution with cost of less than $c^* + \epsilon$ is found. From this follows that for every $\epsilon > 0$ the probability of finding a solution with cost of less than $c^* + \epsilon$ converges towards one in the limit of iterations. This is equivalent to the convergence in probability towards c^*.

■

5.3.5 Implementation Details and Experimental Setup

To validate our theoretical results we conducted a set of experiments. For these experiments we used the same robot as in the previous section. Figure 5.10 shows the setup and the

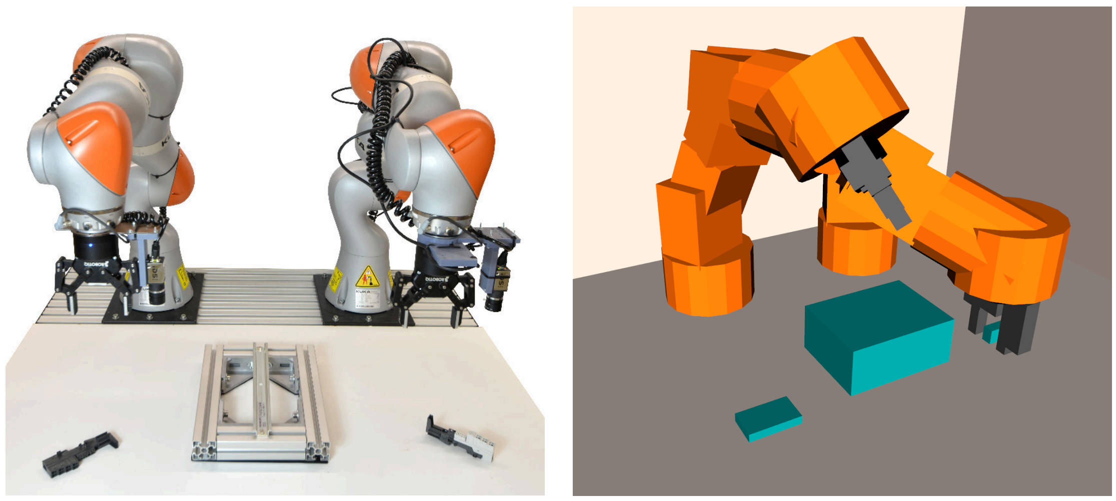


Figure 5.10: Experimental setup: Two cabinet components can be mounted onto a rail. Initially, the poses of all objects are not known precisely. To pick a component it must be fine-localized with a camera. Before a component can be mounted onto the rail, the rail must be fine-localized as well. The left component is placed sideways and can only be mounted after a handover.

Table 5.2: Benchmark Problems

#	Goal
1	Right part assembled on left side of rail
2	Left part assembled on left side of rail (one handover)
3	Both parts assembled on rail in correct positions (one handover)

approximation of the robot geometry that we use for collision checking.

Within the reach of each arm lies an electrical component that can be mounted on a top hat rail in the center of the setup. Before a part can be picked or mounted onto the rail, part and rail must be fine-localized using the flange-mounted cameras. The left part is placed sideways. To mount it on the rail it is necessary to grasp the left part with the left arm and hand it over to the right arm in order to change the grasp transform. Initially both robots are in a upright position with all axis positions and velocities at zero.

For this setup we implemented AO-MM-EST, our planner MM-EST wrapped with the AO-x meta algorithm, as well as our informed planner MM-EST*. Furthermore we implemented the Probabilistic Tree of Roadmaps (PTR) [11] as a baseline. This planner is a highly general task and motion planner that can be adapted to new problems without state-space factorizations. As the PTR is a hierarchical planner it is not suitable to be used in the AO-x algorithm. We attempted to implemented RMR*, the optimal planner we present in Chapter 3, for the experiments as well. This attempt did not succeed due to the

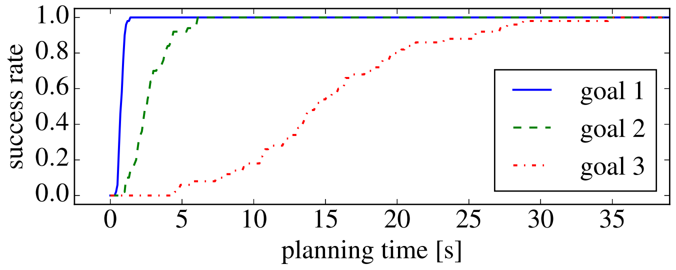


Figure 5.11: Success rates of MM-EST*: Each line visualizes the success rate of 50 planning queries for one of the benchmark tasks.

factorization of the state space, that is necessary for RMR*. The factorization is highly difficult to map on manipulation with diverse actions and also excludes the asymptotically optimal FOBT planner [38] from the experiments.

The procedure **sampleWeighted** is implemented via a grid-based approximation of the sample-density. To determine the grid cells we use the mode of a state, 256 grid cells for robot configurations, and if cost is used 20 grid cells in the cost dimension from 0 to c_{best}.

We created a set of three benchmark problems shown in Table 5.2. The experiments are designed to be increasingly difficult both with respect to the motion component as well as the symbolic task component of the problem. Approximately 45% of the configuration space is in collision and to manipulate objects the gripper must be brought close to the objects. This forms narrow tunnels in the configuration space of the robot. To manipulate the object that is placed sideways, a series of mode transitions is required, that is not specified symbolically. Finally, the order in which objects are localized or grasped can be chosen by the planner and components that are mounted in the wrong position on the rail lead to dead ends. On a symbolic level our planning problem is therefore less complex than typical task planning problems and more complex than typical manipulation planning problems.

As cost function we use the execution duration of the solutions. Transitions are assumed to be executed in three seconds which approximately matches the real execution times of the interactions. We ran all benchmarks 50 times with each planner. AO-MM-EST and MM-EST* receive a time limit of 900 seconds. As the PTR is not an optimizing planner, we use it repeatedly for 900 seconds and store the best solution. Both the PTR and our planners were implemented efficiently in C++ and are optimized to use multiple threads. Transition sampling and weighted sampling is implemented identically for all planners. All experiments were run on a ten-core Intel Xeon E5-2650v3.

5.3.6 Results

The average success rates of MM-EST* on all benchmarks can be seen in Figure 5.11. After less than 35 seconds of planning time, all 50 planning queries return success on all

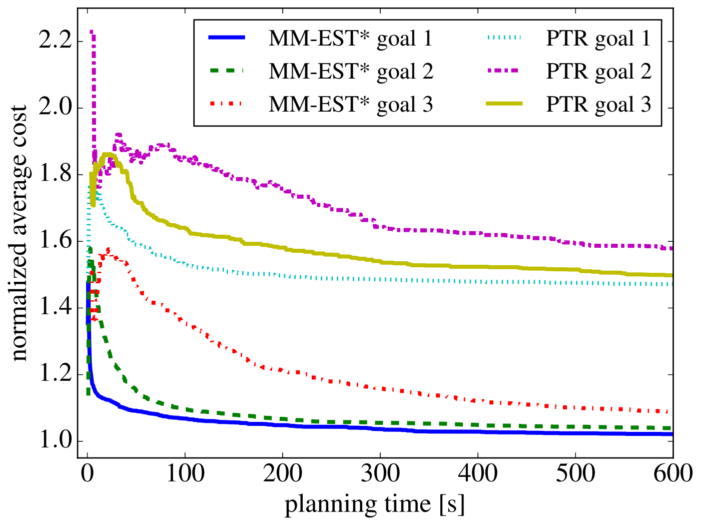


Figure 5.12: Average solution costs of MM-EST* and PTR: The lines visualize the mean cost returned by the planners in 50 runs. All curves are normalized to the minimum cost returned by any planning query at 900 seconds planning time.

of the benchmark tasks. Figure 5.12 visualizes the average cost returned by MM-EST* and PTR. Due to the different lengths of the tasks, cost is normalized to the best result at 900 seconds planning time returned by any planner.

For all benchmarks the average and the variance of cost is reduced over time. However, it can be seen that even at 600 seconds of planning time, costs have not yet converged. This is not surprising as in order to reach convergence the 14-dimensional robot configuration space must be densely sampled.

A comparison of the results of PTR, AO-MM-EST, and MM-EST* is depicted in Table 5.3. For each benchmark the bold numbers represent the average planning time for finding a first solution or cost (execution time) at 600 seconds of planning time. The numbers in brackets show the standard error of the mean. Across all benchmarks MM-EST* achieves an average cost reduction between 27.3% to 34.2% over PTR and 4.5% to 7.4% over AO-MM-EST. Furthermore, AO-MM-EST achieves an improvement of 23.8% to 28.9% over PTR.

Under the assumption of normal distributions with unequal variance for costs we analyzed the results using Welch's unequal variance t-test We test against the null hypothesis of equal average cost. This analysis shows that both AO-MM-EST and MM-EST* significantly (p-value below 1%) outperform the PTR planner. Also, MM-EST* significantly (p-value below 1%) outperforms AO-MM-EST.

Finally, Figure 5.13 visualizes the distribution of costs of all three planners as box plots. It can be seen, that AO-MM-EST and MM-EST* achieve not only lower average cost, but also much lower variance.

Table 5.3: Planning and Execution Times

all values in seconds [standard error of the mean]		Benchmark 1	2	3
PTR	planning time (fist solution)	**3.87**	**64.62**	**14.79**
		[0.40]	[6.46]	[1.27]
	execution time (600 s planning)	**17.98**	**27.01**	**40.17**
		[0.05]	[0.23]	[0.21]
AO-MM-EST	planning time (fist solution)	**1.97**	**13.40**	**15.33**
		[0.21]	[1.18]	[0.85]
	execution time (600 s planning)	**13.49**	**19.20**	**30.59**
		[0.06]	[0.12]	[0.19]
MM-EST*	planning time (fist solution)	**0.75**	**2.76**	**15.37**
		[0.03]	[0.18]	[0.93]
	execution time (600 s planning)	**12.49**	**17.78**	**29.20**
		[0.02]	[0.04]	[0.16]

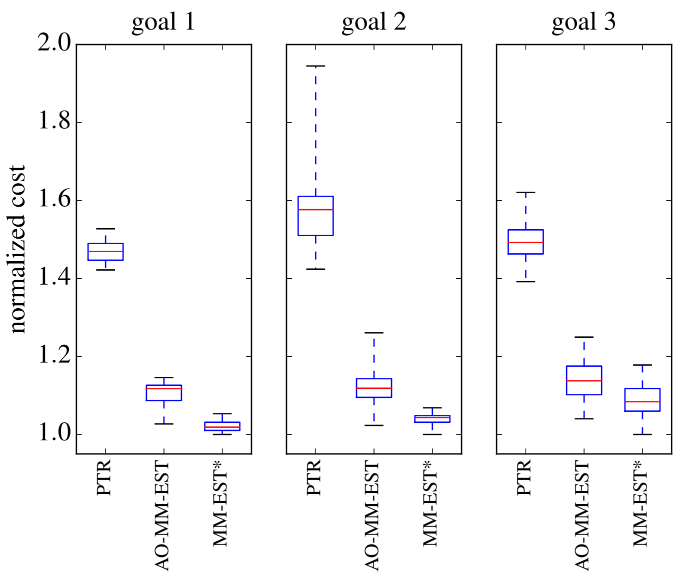


Figure 5.13: Comparison of PTR, AO-MM-EST, and MM-EST*: Each box plot depicts the distribution of normalized cost for a different benchmark after 600 seconds of planning time. Costs are normalized to the minimum cost returned by any planner after 900 seconds. The red line depicts the median. The first and third quartile are represented by the box, minimum and maximum by the whiskers.

5.4 Related Work

The work in this chapter integrates and extends three strands of research: skill-based architectures, manipulation planning, and optimal, kinodynamic motion planning. We view manipulation planning problems from the perspective of a kinodynamic motion planner. This motion planner explores a hybrid state space and has access to an additional set of high-level actions, such as for picking and placing objects. These high-level actions are implemented as re-usable skills. The core idea is to model these skills with a built-in simulator and parameter samplers which renders them as black boxes to the planner.

Skill-Based Frameworks for Robotics Encapsulating robotic control software is a long standing goal in the research community. Examples are recurring control strategies for assembly or high-level actions for picking an object with a known pose. We refer to such a re-usable component as a skill.

Several questions arise in the context of robotic skills: How are re-usable motions of robots defined and controlled? How are multiple motions of the robot composed to create reactive behaviors and how is their execution scheduled both in parallel and sequentially? What is a suitable set of skills? What are the programming interfaces to users of such robotic skills?

Early work on specifying motions that are independent from the kinematics of a robot and are re-usable across similar tasks resulted in the Task Frame Formalism [85]. In this formalism separate control strategies are assigned to different Cartesian degrees of freedom of a robotic tool relative to a reference frame. This work has been generalized by constraint-based task specification frameworks, such as iTaSC [41, 86], the stack of tasks [45], or eTaSL [43].

These frameworks allow to specify and control multiple, parallel robot motions independently of the robot kinematics. However, manipulation requires a series of motions. To this end finite state machines [87] and statecharts [88] have been used extensively. An example in the context of assembly are manipulation primitive networks [89] that switch between different control strategies. To simplify the programming and use of robotic skills several domain specific languages have been developed [90, 91, 92].

The methods discussed so far involve the design of low-level control strategies. For a shop floor worker it may be more convenient to instruct the robot in an object-centered way that is closer to natural language descriptions, e. g., “Place object A on surface B.” To this end Kresse and Beetz [93] model the outcomes of high-level actions using the constraint functions that define constraint-based controllers for the robot motions. This method was extended to use high-level geometric features of objects such as edges or planes for the definition of actions [94] and integrated with the KnowRob [95] knowledge base [96]. This allows the definition of tasks and reasoning about them on an object-centric level. Scioni *et al.* [66] propose a real-time scheduler that composes these actions both sequentially and in parallel.

An alternative approach to object-centered skills is the SkiROS architecture proposed by Rovida *et al.* [84, 97]. In this architecture, object-centered skills are composed of primitives, such as motion planning or compliant controllers. With these skills abstract commands are realized, such as picking an object with a given ID. Suitable skills can be identified by analyzing standard operating procedures [98, 99]. The SkiROS model of skills can be translated into a task planning problem specified in PDDL [32]. Skills are then sequenced by a standard symbolic planner [100].

A key aspect that has not been addressed in the literature on robotic skills is how to resolve the geometric and kinematic interdependencies that arise in manipulation. The assembly scenario used throughout this chapter demonstrates these interdependencies: It may be necessary to grasp and re-grasp an object multiple times before it can be assembled. The first contribution of this chapter is a model of robotic skills that is suitable for manipulation planning. We follow an object centered view on robotic skills similar to the SkiROS [84] architecture but extend it by a parameter sampling and simulation mechanism that is used by a task and motion planner.

Optimal, Kinodynamic Motion Planning Motions of industrial manipulators typically occur in high-dimensional configuration spaces. In these configuration spaces sampling-based planners such as the Probabilistic RoadMap [4], Rapidly exploring Random Tree [5], or the Expansive Space Tree [101] have proven to be highly efficient.

When systems are non-holonomic, time-variant, or when dynamics need to be considered the planning problem is called a kinodynamic planning problem. The RRT algorithm was extended to kinodynamic planning by Lavalle *et al.* [102]. Kindel *et al.* [82] propose a version of the EST planner, that is able to plan with kinodynamic constraints and time-varying environments.

Manipulation planning is similar to kinodynamic motion planning as the motion of the system is non-holonomic. For example, if an object is not in contact with the robot its pose cannot be controlled. For this reason we model manipulation planning as kinodynamic planning with additional high-level actions.

Most sampling-based planners return highly suboptimal solutions that are typically improved via a post-processing step to simplify the solution [17] and to address the dynamic constraints of the robot [83]. Karaman and Frazzoli [13] present a proof for the non-optimality of RRT and PRM as well as asymptotically optimal extensions: RRT* and PRM*. The RRT* algorithm is then extended to optimal, kinodynamic planning [25, 103]. To speed up convergence an informed sampling strategy is introduced by Gammel *et al.* [23]

Li *et al.*propose the kinodynamic planner Stable Sparse RRT (SST) and an asymptotically optimal variant SST* [104]. Hauser and Zhou [27] introduce the AO-x meta-algorithm. This algorithm uses a probabilistically complete planner repeatedly to plan optimally in the limit. The optimal planner proposed in this chapter makes use of the AO-x algorithm, originally intended for motion planning, to achieve global, asymptotic optimality for manipulation planning problems. We conjecture, that the SST* planner could potentially be used in combination with the MM-EST planner proposed in this chapter to obtain optimal plans as well. However, this idea needs further investigation.

Manipulation Planning and Integrated Task and Motion Planning The related fields of manipulation planning and integrated task and motion planning deal with hybrid state spaces of continuous and discrete variables. Both the motion of a system and a sequence of discrete actions must be computed. Discrete variables mostly occur due to the robot making or breaking contact with objects in its environment. This manipulation planning problem was addressed by Alami *et al.* [8] for the case of a fixed set of grasp and placement poses of the object and by Simeon *et al.* [10] for continuous sets of grasps and placements.

Cambon *et al.* [30] introduce a hybrid planner for task and motion planning to address a wider class of problems. Dornhege *et al.* [31] introduce semantic attachments as an extension for the Planning Domain Definition Language (PDDL) [32] to allow task and motion planning with classical symbolic planners. Srivastava *et al.* [105] propose an interface layer between task planners and motion planners. The skills proposed in this chapter can be viewed as a bottom up variant of the semantic attachments by Dornhege *et al.* [31]. With semantic attachments, a *task planner* may query other algorithms, such as motion planners, to determine whether the preconditions of an action hold and what the effects of an action are. In contrast, the skills we propose allow a *motion planner* to perform actions that result in a change of task-level variables.

A key issue in combining motion and task planners is that efficient, sampling-based planners are only probabilistically complete. Thus, guarantees on completeness and optimality are lost if a sampling-based motion planner is simply used as a black box component in a hierarchical planner. Hauser and Latombe [28] introduce a probabilistically complete, multi-modal planner by extending the PRM algorithm to task and motion planning in a way that the motion planner also plans the task component. The key idea of using one holistic planner instead of a hierarchy of planners is a basis for the approach proposed in this chapter. Hauser and Ng-Thow-Hing present two similar, probabilistically complete planners for multi-modal problems [11, 29]. To speed up planning, the use of hierarchy [106], heuristic guidance [33, 35], and learning from previous planning queries [107] has been proposed.

Closest to our work are the PTR planner by Hauser [11], the SMAP planner by Plaku and Hager [108], and the DARRT planner by Barry *et al.* [109]. The PTR planner treats the change of a mode similar to kinodynamic motion planning and hierarchically calls a motion planner for each potential mode transition. The key difference of our approach to the PTR planner is that we treat the exploration of the state space via motions and mode transitions equally without hierarchy. The SMAP planner interleaves high-level symbolic planning with kinodynamic motion planning similar to our approach. However, it is not clear how this planner maps onto the foliated structure that arises in manipulation planning. The DARRT planner extends a kinodynamic version of the RRT by manipulation actions but presents no interface to a symbolic level of abstraction. It is worth to mention the motion planner proposed by Zickler and Veloso [110]. This planner uses a variety of motion controllers as local planners for kinodynamic motion planning in a physics simulator.

So far, the discussed approaches aim at feasible but not optimal planning. Harada *et al.* [36] propose a post processing step to shorten solutions for manipulation planning problems. Zhang and Shah [37] introduce an iterative method that repeats symbolic planning and motion planning to receive better solutions. Both methods are not proven to be locally or globally optimal. For planning problems with piece-wise analytic constraints Vega-Brown and Roy [38] propose the asymptotically optimal FOBT planner. In our previous work [40] (discussed in Chapter 3) we present the RMR* algorithm, which is an asymptotically optimal planner for manipulation problems including those with redundant manipulators, but limited to geometric planning with one object. Both FOBT [38] and RMR* [40] require a factorization of the state space, which makes modeling diverse sets of actions difficult. We have attempted to implement the assembly use case of this chapter with a factored model for manipulation planning. This attempt failed due to the complexity of maintaining and extending the resulting models. A possible explanation is that some constraints of a task can be modeled more efficiently as constraints resulting from a system state, e. g., constraints due to contact. Other constraints are more easily modeled as constraints resulting from actions, e. g., the assembly process in this chapter. Modeling the constraints of actions as constraints of states becomes prohibitively complex when one considers a diverse set of actions.

Another approach to optimal manipulation planning is the logic geometric programming by Toussaint [39]. However, this approach requires to model tasks exclusively via differentiable constraints. This type of model is incompatible with the black box model of collisions and other constraints employed in most of the literature and within this chapter.

5.5 Discussion

This chapter introduced an innovative model for robotic skills that enables manipulation planning for complex, industrial assembly tasks. For this model we proposed an asymptotically optimal, sampling-based manipulation planner. To speed up planning and convergence we developed a generic, informed search strategy.

The skill model we proposed includes methods to sample random parameters for skill execution and to simulate the outcome of this execution. These methods are then used by a kinodynamic planner to put skills in a sequence and to perform the intermediate motion planning. We showed how the resulting planning problem can be reformulated as a multi-modal planning problem. Based on this multi-modal formulation we proved probabilistic completeness and global, asymptotic optimality under a novel set of robustness conditions.

We implemented our approach for a realistic assembly scenario involving two robot arms. This assembly scenario extends classical manipulation planning by a diverse set of actions that includes force controlled assembly and localizing parts with a movable camera. This experiment showed that our approach scales to problems of practical relevance and that the planner is efficient enough to allow repeated, online re-planning. Extensive simulated experiments showed that our approach computes plans with significantly lower cost than an existing planner.

The main advantage of our skill model is that it couples the necessary models for planning with the control strategies for execution. These two aspects of manipulation planning are inevitably coupled and a change in one requires a change in the other. Within our model these two components are part of the same skill resulting in a modular composition of planning domains.

A limitation of this chapter is that a systematic treatment of uncertainties is missing. Furthermore, no connection to a reactive motion control was made. This is a promising area for future research. Potentially, the methods of this chapter could be integrated with a constraint-based model for belief space planning as proposed by Phiquepal and Toussaint [70] and the methods for feedback planning of Chapter 4. This would allow us to compute feedback plans that are simultaneously reactive on the level of motions and on the level of symbolic actions.

Chapter 6

Conclusions

In this thesis we presented innovative models and algorithms that enable robots to effectively and efficiently manipulate objects. Manipulation consists of an interdependent sequence of robot motions and interactions with objects. As a consequence, an autonomous robot must reason about the entire sequence of motions in order to compute a reliable and cost effective solution to an automation task. To this end we introduced asymptotically optimal manipulation planners. When the environments of robots change dynamically, e. g., when humans are present, motions must be adapted in real-time. For this purpose we proposed feedback planners for manipulation that enable a reactive execution of plans. Finally, we presented extensible models that are used to specify the underlying manipulation problems for our planners.

Manipulation typically involves a relatively large number of degrees of freedom, which results in high-dimensional state spaces during planning. To address this, all proposed approaches are built on the idea of sampling-based planning to efficiently explore the combined state space of robots and objects.

All presented approaches have been implemented and evaluated in extensive simulations and on real robots. Our experiments demonstrate that the presented methods enable autonomous manipulation in a variety of tasks and that they scale to the complexity of real world applications.

With these results we now revisit and answer the research questions we posed in Chapter 1. We believe that these answers, together with the methods proposed in this thesis, are a valuable contribution towards autonomous manipulation robots that will assist us in the future.

Research Question 1: How can a robot compute optimal sequences of motions and interactions with objects?

Answer: Manipulation planning can be modeled as motion planning with a multi-modal structure. By making this structure explicit, it is possible to reformulate optimal, sampling-based motion planners for manipulation planning. In this thesis we proposed two such planners.

The first planner, RMR*, extends the PRM* algorithm to manipulation planning with a single object and was presented in Chapter 3. It extends the PRM* by using a factored sampling strategy. Sampling distinguishes between robot configurations, contact parameters for the object, and explicit transition configurations to change between contacts. This approach enables to retain useful features of roadmap-based, optimal motion planners, such as the preprocessing of planning problems and lazy collision checking.

The second optimal planner, MM-EST*, was presented in Chapter 5. It is based on the kinodynamic motion planner EST. We extended the EST to multi-modal planning problems and integrated it with the AO-x meta algorithm for optimal planning. This approach enables to model, plan, and execute complex manipulation tasks, which we demonstrated with a dual-arm robot in an assembly task.

Research Question 2: How can a robot reason about sequences of dynamic motions and interactions with objects in real-time?

Answer: Manipulation planning is a challenging problem that requires substantial computational resources. Sampling-based approaches result in random planning times. For these two reasons it is not feasible to continuously re-plan in real-time. We therefore approach reactive manipulation hierarchically with a separation between deliberate planning and controlled execution.

An important requirement for such an approach is that the controlled execution must adhere to the constraints that constitute the original planning problem. We achieved this with a constraint-based model that is built on the robot dynamics. This model, presented in Chapter 4, forms a common basis for manipulation planning and motion control. We automatically derived constraint-based motion controllers from the model and used them in simulation as local planners within manipulation planners. During execution the same motion controllers can be used to react in real-time to disturbances. As the controllers are derived from the planning domain, the reactive execution adheres both to the decisions of the planner and to the constraints of the planning problem.

Research Question 3: What models are suitable to describe, plan, and control manipulation?

Answer: Models for sequential manipulation tasks must include the constraints on motions of robots and objects as well as the constraints between sequential actions in a task, e. g., between perception and assembly processes. Furthermore, models must be suitable for different "stakeholders" in manipulation. A human task programmer must be able to easily modify and extend models to program new tasks. These models must then be suitable for manipulation planners and finally encode how to control the physical robot.

In Chapter 4 we presented a model for the constraints on motions of robots and objects, the dynamic constraint graph. This model describes each mode of a multi-modal planning problem as a set of constraint functions that are grounded in the robot dynamics. To model a task, a user simply composes constraint functions. This is surprisingly intuitive and computationally efficient as the modeling is done using an expression graph. A key feature of this model is that the constraint functions have meaning to all "stakeholders" in manipulation. They can be interpreted by human task programmers and are suitable for the computations of manipulation planners and motion controllers. As they are grounded in robot dynamics they result in control inputs that are executable on the real robot.

To model the sequential motions and actions in a manipulation task, Chapter 5 proposed a new model for robotic skills. This model is designed to enable task and motion planning based on a set of skills. A set of skills implicitly defines the multi-modal structure of the planning problem and can be used in optimal task and motion planners. To extend planning problems a user can add new robotic skills in a modular way. As we use a kinodynamic formulation of a planning problem the simulation of skills and their execution can be coupled within the skill model. This enables to model, plan, and execute complex assembly processes with a multi-robot system.

6.1 Future Work

The presented work results in several directions for future research. These include optimal feedback planning for manipulation as well as the integration of our methods with physical and probabilistic reasoning.

Combining optimal planning with optimal control: In Chapter 4 we presented methods that enable reactive manipulation based on a constraint-based problem specification. The feedback planners that we presented are based on a linear, second-order dynamic for the constraint functions. This linear dynamic under-utilizes the dynamic limits of the robot and as a result produces trajectories that are smooth but suboptimal with respect to execution time. A promising avenue for future research is to integrate the models of Chapter 4 and planners of Chapter 5 with model predictive control [65]. This could enable a robot to plan time-optimal trajectories and to execute them with real-time reactions to disturbances.

Combining feedback planning with physical reasoning: Throughout this thesis we focus on prehensile manipulation, where a robot rigidly grasps objects. Prehensile manipulation covers a wide array of applications, especially in industrial contexts. However, non-prehensile actions, such as pushing or throwing, are useful for efficient motions in cluttered environments.

This can be enabled by planning manipulation within a physics simulator. Toussaint *et al.* [111] present a variation of Logic-Geometric-Programming [39] that incorporates contact dynamics. Todorov [112] shows how one can tightly integrate goal directed motions and physics simulation with contact. A promising area for future research is to combine these two approaches with the feedback planners in Chapter 4. This could enable integrated planning and controlled, non-prehensile interaction in manipulation tasks.

Combining planning and control with physical, probabilistic reasoning: The methods in this thesis assume that the state of a system is known and evolves deterministically. In unpredictable domains with occlusions, e. g., super markets, the resulting uncertainties must be addressed.

Planning manipulation in a belief space has already been addressed, e. g., by Kaelbling and Lozano-Pérez [113]. However, this approach relies on probabilistic models that make strong assumptions on the structure of the manipulation problem. General, probabilistic models for manipulation, i. e., motion with contacts, are difficult to obtain. The work of Wirnshofer *et al.* [2, 80] shows that probabilistic methods for manipulation benefit from incorporating contact dynamics and the low-level torque controllers of the robot into the probabilistic models.

For this reason, we belief that in order to systematically address uncertainty in manipulation planning one must consider contact dynamics and motion control as well. A potential avenue for future research is to create probabilistic models that build on the dynamic constraint graph presented in this thesis. The reason for this is that the dynamic constraint graph is a combined model for planning and motion control and is potentially suited for constraint-based state estimation as shown in [41].

List of Figures

List of Tables

Bibliography

[1] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[2] Florian Wirnshofer, Philipp S Schmitt, Philine Meister, Georg v Wichert, and Wolfram Burgard. State estimation in contact-rich manipulation. In *Int. Conf. on Robotics and Automation*. IEEE, 2019.

[3] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *The Int. Journal of Robotics Research*, 25(7):627–643, 2006.

[4] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[5] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Department, Iowa State University, 1998.

[6] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The Int. Journal of Robotics Research*, 21(3):233–255, 2002.

[7] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:159–185, 2018.

[8] Rachid Alami, Thierry Simeon, and Jean-Paul Laumond. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In *The fifth Int. Symposium on Robotics Research*, pages 453–463. MIT Press, 1990.

[9] Joseph Mirabel and Florent Lamiraux. Manipulation planning: addressing the crossed foliation issue. In *Int. Conf. on Robotics and Automation*, pages 4032–4037. IEEE, 2017.

[10] Thierry Simeon, Juan Cortes, Anis Sahbani, and Jean-Paul Laumond. A manipulation planner for pick and place operations under continuous grasps and placements. In *Int. Conf. on Robotics and Automation*, volume 2, pages 2022–2027. IEEE, 2002.

[11] Kris Hauser. Task planning with continuous actions and nondeterministic motion planning queries. In *AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 2010.

[12] Shuai D Han, Nicholas M Stiffler, Athanasios Krontiris, Kostas E Bekris, and Jingjin Yu. Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps. *The Int. Journal of Robotics Research*, pages 1775–1795, 2018.

[13] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The Int. Journal of Robotics Research*, 30(7):846–894, 2011.

[14] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[15] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Int. Conf. on Robotics and Automation*, volume 1, pages 521–528. IEEE, 2000.

[16] Kurt Mehlhorn and Peter Sanders. *Algorithms and data structures: The basic toolbox*. Springer Science & Business Media, 2008.

[17] Roland Geraerts and Mark H Overmars. Creating high-quality paths for motion planning. *The Int. Journal of Robotics Research*, 26(8):845–863, 2007.

[18] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *Int. Conf. on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[19] Ioan A. Sucan and Sachin Chitta. Moveit! URL `http://moveit.ros.org`.

[20] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *Conf. on Computer Vision and Pattern Recognition, 2008*, pages 1–8. IEEE, 2008.

[21] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Transactions on Pattern Analysis and Machine Intelligence*, 36 (11):2227–2240, 2014.

[22] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science and Engineering*, 5(1):46–55, 1998.

[23] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Int. Conf. on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[24] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *Int. Conf. on Robotics and Automation*, pages 2951–2957. IEEE, 2015.

[25] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Conf. on Decision and Control*, pages 7681–7687. IEEE, 2010.

[26] Léonard Jaillet and Josep M Porta. Asymptotically-optimal path planning on manifolds. *Robotics: Science and Systems VIII*, page 145, 2013.

[27] Kris Hauser and Yilun Zhou. Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *Transactions on Robotics*, 32(6):1431–1443, 2016.

[28] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The Int. Journal of Robotics Research*, 2009.

[29] Kris Hauser and Victor Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *The Int. Journal of Robotics Research*, 30 (6):678–698, 2011.

[30] Stéphane Cambon, Fabien Gravot, and Rachid Alami. Overview of asymov: Integrating motion, manipulation and task planning. In *Int. Conf. on Automated Planning and Scheduling Doctoral Consortium*, 2003.

[31] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Towards service robots for everyday environments*, pages 99–115. Springer, 2012.

[32] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

[33] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.

[34] Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57, 2001.

[35] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Backward-forward search for manipulation planning. In *Int. Conf. on Intelligent Robots and Systems*, pages 6366–6373. IEEE, 2015.

[36] Kensuke Harada, Tokuo Tsuji, and Jean-Paul Laumond. A manipulation motion planner for dual-arm industrial manipulators. In *Int. Conf. on Robotics and Automation*, pages 928–934. IEEE, 2014.

[37] Chongjie Zhang and Julie A Shah. Co-optimizing task and motion planning. In *Int. Conf. on Intelligent Robots and Systems*, pages 4750–4756. IEEE, 2016.

[38] William Vega-Brown and Nicholas Roy. Asymptotically optimal planning under piecewise-analytic constraints. *The 12th Int. Workshop on the Algorithmic Foundations of Robotics*, 2016.

[39] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth Int. Joint Conf. on Artificial Intelligence*, 2015.

[40] Philipp S Schmitt, Werner Neubauer, Wendelin Feiten, Kai M Wurm, Georg v Wichert, and Wolfram Burgard. Optimal, sampling-based manipulation planning. In *Int. Conf. on Robotics and Automation*, pages 3426–3432. IEEE, 2017.

[41] Joris De Schutter, Tinne De Laet, Johan Rutgeerts, Wilm Decré, Ruben Smits, Erwin Aertbeliën, Kasper Claes, and Herman Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *The Int. Journal of Robotics Research*, 26(5):433–455, 2007.

[42] Joseph Mirabel and Florent Lamiraux. Handling implicit and explicit constraints in manipulation planning. In *Robotics: Science and Systems*, 2018.

[43] Erwin Aertbeliën and Joris De Schutter. eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs. In *Int. Conf. on Intelligent Robots and Systems*, pages 1540–1546. IEEE, 2014.

[44] Tinne De Laet and Joris De Schutter. Constraint-based control of sensor-based robot systems with uncertain geometry. Technical report, Department of Mechanical Engineering, KU Leuven, 2007.

[45] Nicolas Mansard, Olivier Stasse, Paul Evrard, and Abderrahmane Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *Int. Conf. on Advanced Robotics*, pages 1–6. IEEE, 2009.

[46] Ki Suh and J Hollerbach. Local versus global torque optimization of redundant manipulators. In *Int. Conf. on Robotics and Automation*, volume 4, pages 619–624. IEEE, 1987.

[47] Ruben Smits. KDL: Kinematics and Dynamics Library. `http://www.orocos.org/kdl`.

[48] Hans J Ferreau, Hans G Bock, and Moritz Diehl. An online active set strategy to overcome the limitations of explicit mpc. *Int. Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[49] Hans J Ferreau, Christian Kirches, Andreas Potschka, Hans G Bock, and Moritz Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[50] Torsten Kröger and Friedrich M Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *Transactions on Robotics*, 26(1): 94–111, 2010.

[51] Torsten Kröger. Opening the door to new sensor-based robot applications: The reflexxes motion libraries. In *Int. Conf. on Robotics and Automation*, pages 1–4. IEEE, 2011.

[52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[53] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Int. Conf. on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

[56] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *Int. Conf. on Learning Representations*, 2016.

[57] Hado Van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[58] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conf. on Artificial Intelligence*, 2016.

[59] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *Int. Conf. on Machine Learning*, pages 1995–2003, 2016.

[60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. on Learning Representations*, 2015.

[61] François Chollet et al. Keras. `https://keras.io`, 2015.

[62] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[63] Bruno Siciliano and Jean-Jaques E Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Int. Conf. on Advanced Robotics*, volume 2, pages 1211–1216, 1991.

[64] Wilm Decré, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Extending iTaSC to support inequality constraints and non-instantaneous task specification. In *Int. Conf. on Robotics and Automation*, pages 964–971. IEEE, 2009.

[65] Wilm Decré, Herman Bruyninckx, and Joris De Schutter. Extending the itasc constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons. In *Int. Conf. on Robotics and Automation*, pages 1941–1948. IEEE, 2013.

[66] Enea Scioni, Gianni Borghesan, Herman Bruyninckx, and Marcello Bonfè. Bridging the gap between discrete symbolic planning and optimization-based robot control. In *Int. Conf. on Robotics and Automation*, pages 5075–5081. IEEE, 2015.

[67] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *The Int. Journal of Robotics Research*, 23 (7-8):729–746, 2004.

[68] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. Manipulation planning on constraint manifolds. In *Int. Conf. on Robotics and Automation*, pages 625–632. IEEE, 2009.

[69] Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylene Campana, Nicolas Mansard, and Florent Lamiraux. HPP: A new software for constrained motion planning. In *Int. Conf. on Intelligent Robots and Systems*, 2016.

[70] Camille Phiquepal and Marc Toussaint. Combined task and motion planning under partial observability: An optimization-based approach. In *Int. Conf. on Robotics and Automation*. IEEE, 2019.

[71] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The Int. Journal of Robotics Research*, 18(6):534–555, 1999.

[72] H Işil Bozma and Daniel E Koditschek. Assembly as a noncooperative game of its pieces: analysis of 1d sphere assemblies. *Robotica*, 19(1):93–108, 2001.

[73] Cem Serkan Karagoz, H Isil Bozma, and Daniel E Koditschek. Feedback-based event-driven parts moving. *Transactions on Robotics*, 20(6):1012–1018, 2004.

[74] Vasileios Vasilopoulos, T Turner Topping, William Vega-Brown, Nicholas Roy, and Daniel E Koditschek. Sensor-based reactive execution of symbolic rearrangement plans by a legged mobile manipulator. In *Int. Conf. on Intelligent Robots and Systems.*, 2018.

[75] J Zachary Woodruff and Kevin M Lynch. Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks. In *Int. Conf. on Robotics and Automation*, pages 4066–4073. IEEE, 2017.

[76] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[77] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[78] Philipp S Schmitt, Florian Wirnshofer, Kai M Wurm, Georg v Wichert, and Wolfram Burgard. Modeling and planning manipulation in dynamic environments. In *Int. Conf. on Robotics and Automation*. IEEE, 2019.

[79] Philipp S Schmitt, Florian Wirnshofer, Kai M Wurm, Georg v Wichert, and Wolfram Burgard. Planning reactive manipulation in dynamic environments. In *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2019.

[80] Florian Wirnshofer, Philipp S Schmitt, Wendelin Feiten, Georg v Wichert, and Wolfram Burgard. Robust, compliant assembly via optimal belief space planning. In *Int. Conf. on Robotics and Automation*, pages 1–5. IEEE, 2018.

[81] Fabrizio Caccavale, Pasquale Chiacchio, Alessandro Marino, and Luigi Villani. Six-dof impedance control of dual-arm cooperative manipulators. *Transactions On Mechatronics*, 13(5):576–586, 2008.

[82] Robert Kindel, David Hsu, J-C Latombe, and Stephen Rock. Kinodynamic motion planning amidst moving obstacles. In *Int. Conf. on Robotics and Automation*, volume 1, pages 537–543. IEEE, 2000.

[83] Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Int. Conf. on Robotics and Automation*, pages 2493–2498. IEEE, 2010.

[84] Francesco Rovida and Volker Krüger. Design and development of a software architecture for autonomous mobile manipulators in industrial environments. In *Int. Conf. on Industrial Technology*, pages 3288–3295. IEEE, 2015.

[85] Herman Bruyninckx and Joris De Schutter. Specification of force-controlled actions in the" task frame formalism"-a synthesis. *Transactions on Robotics and Automation*, 12(4):581–589, 1996.

[86] Ruben Smits, Tinne De Laet, Kasper Claes, Herman Bruyninckx, and Joris De Schutter. itasc: a tool for multi-sensor integration in robot manipulation. In *Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, pages 426–433. IEEE, 2008.

[87] Jonathan Bohren and Steve Cousins. The smach high-level executive [ros news]. *IEEE Robotics & Automation Magazine*, 17(4):18–20, 2010.

[88] Markus Klotzbücher and Herman Bruyninckx. Coordinating robotic tasks and systems with rfsm statecharts. *Journal of Software Engineering for Robotics*, 3(1): 28–56, 2012.

[89] Bernd Finkemeyer, Torsten Kröger, and Friedrich M Wahl. Executing assembly tasks specified by manipulation primitive nets. *Advanced Robotics*, 19(5):591–611, 2005.

[90] Dominick Vanthienen, Markus Klotzbu, Joris De Schutter, Tinne De Laet, Herman Bruyninckx, et al. Rapid application development of constrained-based task modelling and execution using domain specific languages. In *Int. Conf. on Intelligent Robots and Systems*, pages 1860–1866. IEEE, 2013.

[91] Ulrike Thomas, Gerd Hirzinger, Bernhard Rumpe, Christoph Schulze, and Andreas Wortmann. A new skill based robot programming language using uml/p statecharts. In *Int. Conf. on Robotics and Automation*, pages 461–466. IEEE, 2013.

[92] Frank Nägele, Lorenz Halt, Philipp Tenbrock, and Andreas Pott. A prototype-based skill model for specifying robotic assembly tasks. In *Int. Conf. on Robotics and Automation*, pages 558–565. IEEE, 2018.

[93] Ingo Kresse and Michael Beetz. Movement-aware action control - integrating symbolic and control-theoretic action execution. In *Int. Conf. on Robotics and Automation*, pages 3245–3251. IEEE, 2012.

[94] Georg Bartels, Ingo Kresse, and Michael Beetz. Constraint-based movement representation grounded in geometric features. In *Int. Conf. on Humanoid Robots*, pages 547–554. IEEE, 2013.

[95] Moritz Tenorth and Michael Beetz. Knowrob - knowledge processing for autonomous personal robots. In *Int. Conf. on Intelligent Robots and Systems*, pages 4261–4266. IEEE, 2009.

[96] Moritz Tenorth, Georg Bartels, and Michael Beetz. Knowledge-based specification of robot motions. In *European Conf. on Artificial Intelligence*, 2014.

[97] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios S Polydoros, Bjarne Großmann, Ronald PA Petrick, and Volker Krüger. Skiros - a skill-based robot control platform on top of ros. In *Robot Operating System (ROS)*, pages 121–160. Springer, 2017.

[98] Simon Bøgh, Mads Hvilshøj, Morten Kristiansen, and Ole Madsen. Identifying and evaluating suitable tasks for autonomous industrial mobile manipulators (aimm). *The Int. Journal of Advanced Manufacturing Technology*, 61(5-8):713–726, 2012.

[99] Simon Bøgh, Oluf Skov Nielsen, Mikkel Rath Pedersen, Volker Krüger, and Ole Madsen. Does your robot have skills? In *Int. Symposium on Robotics*, page 6. Verlag, 2012.

[100] Matthew Crosby, Ronald P A Petrick, Francesco Rovida, and Volker Krueger. Integrating mission and task planning in an industrial robotics framework. In *Int. Conf. on Automated Planning and Scheduling*. AAAI, 2017.

[101] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Int. Conf. on Robotics and Automation*, volume 3, pages 2719–2726. IEEE, 1997.

[102] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The Int. Journal of Robotics Research*, 20(5):378–400, 2001.

[103] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Int. Conf. on Robotics and Automation*, pages 5041–5047. IEEE, 2013.

[104] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The Int. Journal of Robotics Research*, 35(5):528–564, 2016.

[105] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Int. Conf. on Robotics and Automation*, pages 639–646. IEEE, 2014.

[106] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Int. Conf. on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[107] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *Int. Conf. on Robotics and Automation*, pages 2810–2817. IEEE, 2017.

[108] Erion Plaku and Gregory D Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Int. Conf. on Robotics and Automation*, pages 5002–5008. IEEE, 2010.

[109] Jennifer Barry, Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation with multiple action types. In *Experimental Robotics*, pages 531–545. Springer, 2013.

[110] Stefan Zickler and Manuela Veloso. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 27–33. Int. Foundation for Autonomous Agents and Multiagent Systems, 2009.

[111] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*, 2018.

[112] Emanuel Todorov. Goal directed dynamics. In *Int. Conf. on Robotics and Automation*, pages 2994–3000. IEEE, 2018.

[113] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The Int. Journal of Robotics Research*, 32(9-10):1194–1227, 2013.