

Learning of Generalized Manipulation Strategies in Service Robotics

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Rainer Jäkel

aus Schwetzingen

Tag der mündlichen Prüfung: 29.01.2013
Erster Gutachter: Prof. Dr.-Ing. Rüdiger Dillmann
Zweiter Gutachter: Prof. Michael Beetz, PhD

Ich versichere wahrheitsgemäß, die Dissertation bis auf die dort angegebenen Hilfen selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und als kenntlich gemacht zu haben, was aus Arbeiten anderer und eigenen Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, Dezember 2012

Preface

Service robots are on the verge to revolutionize our daily life promising a longterm increase in productivity and quality of life. The automation of manipulation tasks in the variable human environment requires a high flexibility of the robot. Programming a robot in an efficient way to achieve this kind of flexibility is far from being technically solved. The robot has to look ahead and predict its effects on the environment to guarantee that the task is executed properly. This kind of prediction leads to a planning process, which enables the robot to solve a task in a goal-directed way while dealing with boundary conditions of the task and the limitations of its manipulation capabilities.

The first contribution of this thesis is a novel representation of manipulation tasks and its integration into a planning system leading to a deeper understanding of the modeling and planning challenges involved in the automation of everyday manipulation tasks. The second contribution centers around the question how the robot can be equipped with planning knowledge for a large variety of tasks. Learning from natural human demonstrations of the task is combined with computer-aided simulation to derive highly efficient and flexible planning knowledge. Focus is on deepening the understanding on how to derive planning knowledge from human demonstrations although the human and robot have a different morphology. The developed methods are implemented in software and are fully integrated. Experiments with different manipulation tasks in a dedicated sensor environment as well as on multiple, partially anthropomorphic service robots show the versatility of the developed methods.

Hopefully, the discussions and insights in this thesis prove helpful to other researchers on the road to intelligent robots proactively shaping their environment and constantly learning from observation as well as their own experience.

Karlsruhe,
December 2012

Rainer Jäkel
Karlsruhe Institute of Technology (KIT)

Zusammenfassung

In der vorliegenden Arbeit wird die Manipulation von Objekten durch einen Serviceroboter unter Verwendung von Planungsmethoden untersucht. Der Einsatz im menschlichen Umfeld erfordert dabei ein hohes Maß an Flexibilität um mit der Vielzahl an Aufgaben, Objekten sowie Umgebungen zurecht zu kommen. Zusätzlich müssen die Fähigkeiten des Roboters sowie Rand- und Nebenbedingungen einer Aufgabe berücksichtigt werden, um eine gültige Ausführung zu garantieren. Dazu werden Methoden zur automatischen Planung von Roboterbewegungen unter Rand- und Nebenbedingungen eingesetzt.

Die Abbildung von Manipulationsaufgaben auf eine operationale Beschreibung, die zur zielgerichteten Planung eingesetzt werden kann, ist herausfordernd. Einerseits müssen alle relevanten Rand- und Nebenbedingungen einer Aufgabe erfasst und notwendige Teilziele definiert werden. Andererseits muss der Suchraum für Roboterbewegungen eingeschränkt werden, um effizient koordinierte Finger- und Armbewegungen erzeugen zu können. In der vorliegenden Arbeit wird ein Lern- und Planungsprozess vorgestellt, um das notwendige Planungswissen aus der Beobachtung des Menschen zu lernen, was als Programmieren durch Vormachen bezeichnet wird, unter Einsatz computergestützter Simulationsverfahren zu verfeinern und automatisch auf dem Roboter auszuführen. Der vorgestellte Lern- und Planungsprozess wurde als Software in einer dedizierten Sensorumgebung sowie auf mehreren, teilweise anthropomorphen Robotern umgesetzt, integriert und anhand vielfältiger Experimente evaluiert.

Das entwickelte Konzept, damit verbundene Einsichten sowie dessen Limitationen werden im Folgenden im Detail diskutiert.

Acknowledgments

I would like to thank Professor Rüdiger Dillmann for his support and inspiration in exciting discussions as well as providing me with a stimulating working environment. Many thanks to Professor Michael Beetz for supervising my thesis and asking interesting questions. Thanks to Professor Jürgen Beyerer and Professor Rainer Stiefelhagen for their interest in my work and a fair examination.

My deepest thanks to Sven R. Schmidt-Rohr, my mentor from the beginning and role model in dedication and determination. He set the impulse for my work and it has been a pleasure to work with such a rich intellect for so many years. Thanks to Pascal Meißner for interesting discussions broadening my understanding of robotics as well as fruitful collaboration in writing so many project proposals. Many thanks to Alexander Kasper for teaching me the basics of 3D modeling and sharing the fun of creating a realistic 3D model of my hand. My thanks to Martin Lösch for his tremendous help and good advise.

Thanks to my students Yi Xie and Alexander Bachmeier for helping me with the software implementation, hardware realization and countless experiments.

I would like to thank Zhixing Xue, Steffen Rühl and Andreas Hermann for the fruitful collaboration and their tremendous amount of work in building and maintaining the infrastructure of Adero. Many thanks to Sebastian Brechtel and Tobias Gindele for the helpful, mathematical discussions and the caffeinated fruits of their striving for perfection. Thanks to Christine Brand, Diana Kreidler and Isabelle Wappler for taking care of so many different administrative tasks and being always helpful and gentle.

Thanks to my parents Ingrid and Werner for teaching me and inspiring me of the right things. I will always be grateful for their support and advice in all aspects of my life. Many thanks to my sister Carolin who has always been my idol and showed me how to be independent and adventurous. Thanks to Markus Knopf for long-standing, profound friendship and support - not only in advanced math.

Finally, my biggest thanks to Sabine Sayler for her uncompromising support and remarkable ambition, which inspires and motivates me anew every day. By entering my life she completed me.

Karlsruhe, February 2013

Rainer Jäkel

Contents

Preface	i
Zusammenfassung	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	2
1.2 Thesis Statement and Preliminary Work	5
1.3 Concept Overview	6
1.4 Thesis Contribution	10
1.5 Document Outline	12
2 Theoretic Background and Related Work	13
2.1 Planning of Manipulation Tasks	14
2.1.1 Probabilistic Roadmaps	15
2.1.2 Rapidly-Exploring Random Trees	17
2.1.3 Constraints	18
2.1.4 Constraint Projection	21
2.1.5 Constraint-based Motion Planning	22
2.2 Efficient Search Heuristics for Motion Planning	23
2.2.1 Workspace Decomposition	24
2.2.2 Workspace Biasing	24
2.3 Execution of Compliant Motion Tasks	26
2.4 Programming by Demonstration	28
2.4.1 Probabilistic Imitation Learning	30
2.4.2 Dynamic Imitation Learning	32
2.4.3 Symbolic PbD	34
2.4.4 Feature Selection	36
2.5 Analysis	39
2.5.1 Requirements	40
2.5.2 Evaluation	40
2.5.3 Conclusion	43

3	Modelling and Planning of Manipulation Tasks	45
3.1	Constraints	45
3.1.1	Constraint Definition	47
3.1.2	Constraint Types	51
3.1.3	Constraint Regions	58
3.1.4	Discussion	62
3.2	Planning Models	62
3.2.1	Strategy Graph	63
3.2.2	Constraint Satisfaction Problem	64
3.2.3	Planning Model Linearization	70
3.2.4	Discussion	70
3.3	Mapping of Planning Models	70
3.3.1	Morphing of Geometric Object Models	72
3.3.2	Coordinate Frame Mapping	73
3.3.3	Discussion	73
3.4	Constraint-based Motion Planning	74
3.4.1	Projection Techniques	75
3.4.2	Sampling of Goal Configurations	77
3.4.3	Goal-directed Planning	80
3.4.4	Planning with Dynamics Simulation	80
3.4.5	Discussion	82
3.5	Summary and Conclusion	83
4	Programming by Demonstration of Planning Models for Manipulation Tasks	85
4.1	Observation of Explicit Human Demonstrations of Manipulation Tasks	87
4.1.1	Sensor environment	88
4.1.2	Sensor data	90
4.1.3	Explicit Demonstrations	91
4.1.4	Discussion	93
4.2	Generation of Preliminary Planning Model	93
4.2.1	Structure of the Preliminary Planning Model	95
4.2.2	Automatic Generation of Contact Coordinate Systems	95
4.2.3	Automatic Deduction of Constraint Sets	96
4.2.4	Learning of Constraint Parameters	101
4.2.5	Discussion	102
4.3	Consideration of the Correspondence Problem	103
4.3.1	Workspace Analysis	104
4.3.2	Constraint Relaxation	105

4.3.3	Discussion	106
4.4	Demonstration-based Generalization of Planning Models	107
4.4.1	Removing Inconsistent Constraints	108
4.4.2	Discussion	108
4.5	Robot-Test-based Generalization of Planning Models	109
4.5.1	Robot-Test Specification	111
4.5.2	Evolutionary Algorithm	112
4.5.3	Discussion	119
4.6	Efficient Execution of Planning Models	121
4.6.1	Overview	123
4.6.2	Definition of Search Heuristics	124
4.6.3	Selection of Search Heuristics	124
4.6.4	Temporal Alignment and Clustering	125
4.6.5	Control Algorithm for Reverse Execution of Search Heuristics	126
4.6.6	Extension of Constraint-based Motion Planner	128
4.6.7	Discussion	129
4.7	Constraint Tightening to Solve the Correspondence Problem	130
4.8	Summary and Conclusion	131
5	Evaluation	135
5.1	Experiment Setup Overview	136
5.1.1	Robots	136
5.1.2	Object Database	137
5.2	Evaluation of Planning Algorithm	137
5.2.1	Grasp Strategies	138
5.2.2	Planning without Physics Simulation	142
5.3	Evaluation of Planning Model Generalization	146
5.3.1	Demonstration-based Generalization	146
5.3.2	Robot Tests	151
5.4	Evaluation of Planning Model Specialization	156
5.4.1	Bottle in Crate	157
5.4.2	Bottle in Fridge	162
5.4.3	Chips Can to Rack	166
5.4.4	Comparison with Control-Approach	170
5.4.5	Conclusion	171
5.5	Evaluation of Learning and Planning of Dexterous Manipulation Tasks	171
5.5.1	Opening a Bottle	172
5.5.2	Lifting a Spoon	174
5.5.3	Conclusion	179

- 5.6 Evaluation of Constraint Specialization to Consider the Correspondence Problem 179
- 5.7 Evaluation of Scalability to Different Robot Systems 183
- 5.8 Conclusion 183

- 6 Summary 189**
 - 6.1 Limitations and Outlook 193
 - 6.2 Conclusion 194

- A Glossary and Notation 197**

- B Strategy Graph: Operators and Metrics 203**
 - B.1 Strategy Graph Distance Metrics 203
 - B.2 Strategy Graph Operators 204
 - B.3 Strategy Graph Linearization 205

- C Software Implementation 207**

- D Prior Publications and Research Collaboration 209**

- E List of Figures 211**

- F List of Tables 215**

- G Bibliography 217**

1. Introduction

Our daily life is shaped by a large number of manual, repetitive tasks. Inventions like washing machines or dish washers are highly effective and allow us to spend less time on a subset of these tasks while maintaining a high quality of the result. Nevertheless, a large number of everyday tasks exist which cannot be automated yet. The automation of everyday tasks with a service robot is an emerging technology with the goal to fill this gap, promising a longterm increase in productivity and quality of life.

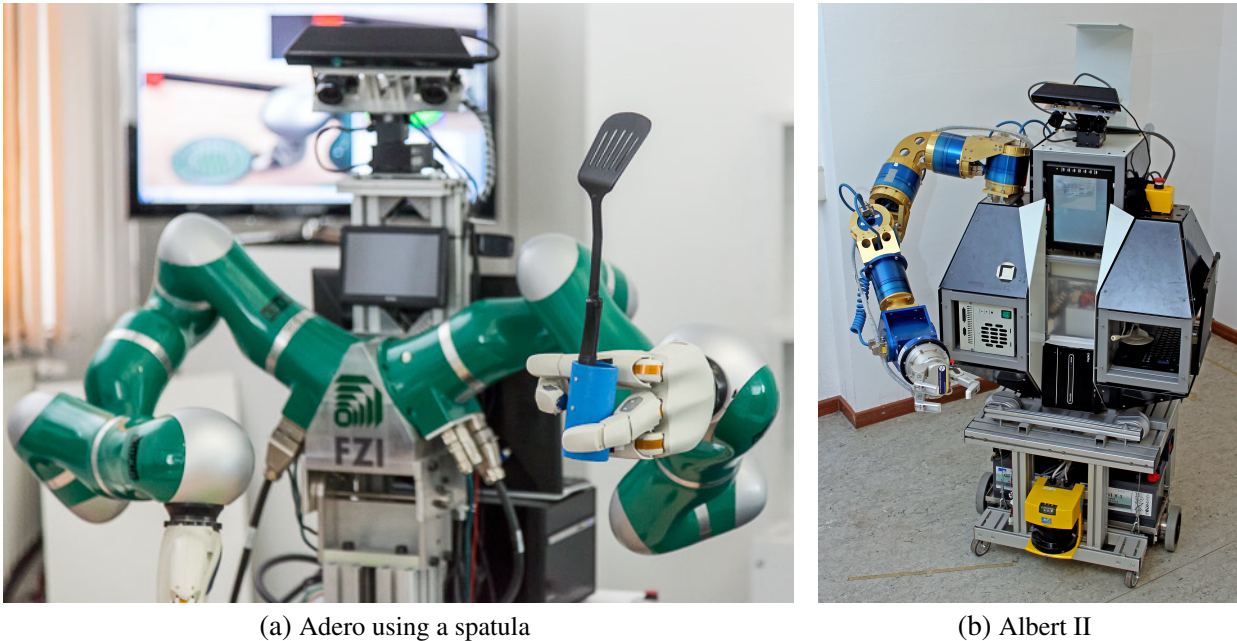
Sophisticated manipulation capabilities are required to automate everyday tasks in human-centered environments. A service robot has to be able to grasp objects of different sizes, move objects without collisions in the environment or manipulate objects with its fingertips. Due to this observation, service robots are modeled physically after humanoids, leading to the notation of *anthropomorphic robots* with two robot arms and multi-fingered robot hands with opposing thumbs. Figure 1.1 shows the service robot Albert II and the anthropomorphic robot Adero. A service robot is a complex, physical machine with a high number of degrees of freedom (DOF). The latter refers to the number of joints of the robot, which can either be passive or controlled actively. Due to this complexity, service robots are physically capable to solve a large variety of manipulation tasks.

However, programming a service robot in an efficient way to achieve this goal is far from being technically solved. The large variety of objects and obstacles as well as the restricted workspace in human-centered environments demands high flexibility of the robot. In this context, we define flexibility as the ability to execute a manipulation task in different environments with different objects, obstacles and start configurations.

In industrial robotics, standard tools exist to program a small set of robot trajectories efficiently. Such robot trajectories, either represented in the high-dimensional joint space or the 6D-Cartesian space, cannot be adapted efficiently to large changes in the environment. Therefore, the flexibility of the resulting robot program is limited.

In order to achieve a higher flexibility the robot has to look ahead and predict its effects on the environment. This kind of prediction leads to a motion planning process, which is one of the key abilities to automate everyday manipulation tasks. Motion planning enables the robot to solve a task in a goal-directed way while dealing with boundary conditions of the task and the limitations of its manipulation capabilities.

Representing complex manipulation tasks in a way suitable for motion planning is still an open research question. The representation has to cover all relevant aspects of manipulation tasks, e.g. forces between the robot fingers and objects, object poses as well as contacts between objects, potentially leading to a high complexity of the motion planning process. At the same time, the



(a) Adero using a spatula

(b) Albert II

Figure 1.1.: Anthropomorphic robot Adero (a) and service robot Albert II (b).

representation has to allow efficient planning of the task in a large number of environments to guarantee high productivity. The high efficiency and flexibility of human manipulation capabilities show that these apparently contradicting challenges can be solved at the same time.

Humans are able to solve repetitive tasks efficiently by adapting reference motions, e.g. grasping a large number of identical objects on a table with the same approach motion. Additionally, humans possess the required knowledge to generate new solutions on the fly to handle situations, in which adaptation is insufficient.

The representation and acquisition of manipulation knowledge, which enables a service robot to solve complex everyday manipulation tasks and become gradually faster by learning from its own experience, is fundamental to achieve human-like productivity and flexibility.

1.1. Motivation

The foundation to automate everyday manipulation tasks is their representation. In literature, we can distinguish symbolic and subsymbolic representations. Figure 1.2 gives a short overview of manipulation task representations, which are used in this section.

Symbolic approaches represent manipulation tasks on the basis of atomic actions or elementary operators, e.g. like grasp, approach or screw. In STRIPS-like representations [40], actions also contain pre- and post-conditions, which allow to plan a sequence of actions to reach a certain goal. Pre- and post-conditions are symbolic, e.g. a relation like *Above(Cup, Saucer)* may represent the goal of the action *PutOnTop*. The symbolic information alone is insufficient to generate a robot motion. We assume that the execution of the action will result in a state of the world, in which the relation holds. The mapping of the symbol *Above(Cup, Saucer)* to a set of adequate states in the

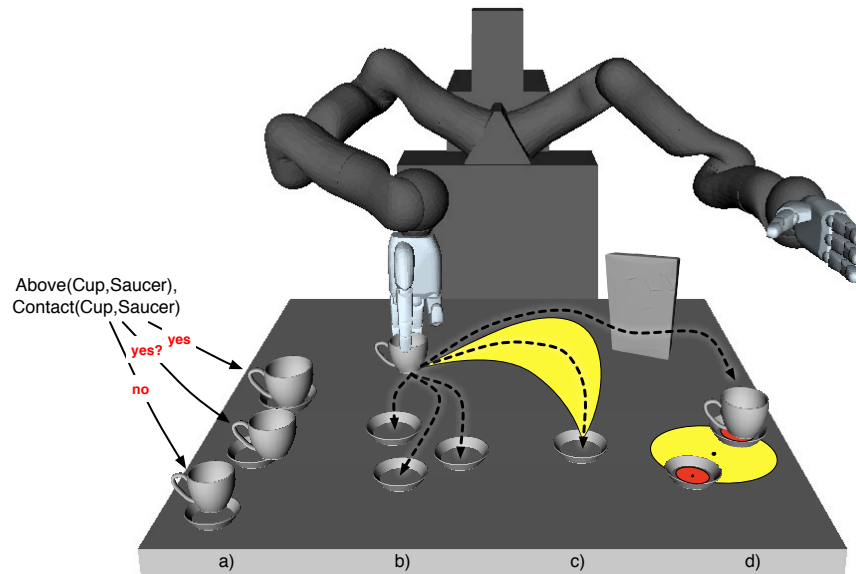


Figure 1.2.: Task representations to place a cup on a saucer. Symbolic representation (a), Dynamic Movement Primitives (b), Probabilistic representation (c) and explicit representation of goals and constraints (d).

real-world refers to the problem of *symbol grounding* and is currently unsolved. Although the generalization capabilities of symbolic approaches are enormous, the lack of subsymbolic information to generate robot motions is a severe disadvantage.

In contrast, subsymbolic approaches represent manipulation knowledge in a way that can be used directly to generate a robot motion. An example are Dynamic Movement Primitives (DMP) which can be used to define a set of differential equations for each joint of the robot. By integrating the equations a reference trajectory for each joint is obtained. In principle, generalization is more critical than in symbolic representation. A joint-based representation does not allow to adapt the motion to different object poses. Representing robot trajectories in Cartesian space, e.g. as a reference trajectory for the Tool-Center-Point (TCP) of the robot, does not solve the problem. Since the Cartesian trajectory adapts to different object poses, calculation of inverse kinematics is necessary to generate a joint-vector trajectory. Due to limited joint ranges, singularities, self-collisions as well as collisions with the environment, this simple adaptation is insufficient to generalize to large changes in the environment. The representation of a continuous set of (Cartesian) trajectories increases the probability that a valid solution is contained. An example are Gaussian Mixture Models (GMM). A GMM represents a probability distribution covering different task aspects in Cartesian and joint space at the same time. In order to generate the robot motion a simple controller is used, which limits generalization also to minor changes in the environment.

A more general idea is to represent goals and constraints of a manipulation task explicitly, e.g. using GMMs to model individual constraints. Each goal or constraint refers only to a small number of objects, similar to symbolic relations. The advantages are that goals and constraints can be instantiated in different environments and by using motion planning generalization to large changes

in the environment is possible. The disadvantage is that it is more difficult to define manipulation tasks. A lot of ambiguities exist, which have to be resolved: what are the relevant goals, which objects should be constrained in what way to reach a goal and how to constrain the robot motion to be able find a successful robot motion efficiently? Programming this way is time-consuming, error-prone and requires expert knowledge. The advantages outweigh the disadvantage and we follow this idea to represent manipulation tasks on the basis of an explicit representation of goals and constraints. In order to counteract the disadvantage, we have to look into programming techniques which do not require to specify goals and constraints manually.

In human-centered environments, knowledge in robot programming is lacking but a great deal of manipulation knowledge exists: humans are experts in solving everyday manipulation tasks. In Programming by Demonstration (PbD) the goal is to observe a human teacher while performing the manipulation task, e.g. with his own hands or by teleoperating the robot, and deduce a representation of the manipulation task step-by-step. The latter usually relies on machine learning methods to determine the free parameters, e.g. in our case the parameters of goals and constraints, based on the set of human demonstrations. Programming the robot by demonstrating the task with our own hands is very intuitive and promises a severe reduction in programming effort and (textual) programming experience. In this thesis, we apply the PbD paradigm and learn manipulation tasks based on a set of human demonstrations.

Important requirements have to be considered. The demonstration of a single example usually takes several seconds to minutes, which makes it difficult to produce a large number of demonstrations. Similar to the human, a robot has to be able to learn novel tasks using only a small number of demonstrations. In general, user-interaction should be limited and the learning process should be as automated as possible, e.g. the user should not be asked to clarify hundreds of ambiguities.

The final step is to generate a robot motion based on the explicit representation of goals and constraints. We have to ensure that all goals are reached and all constraints are obeyed. Controllers are fast but insufficient to cope with the large variety of objects, obstacles and start configurations in human-centered environments. Recent results in the field of constraint-based motion planning allow to plan robot motions online considering constraints, self-collisions and collisions with the environment. The cost of this flexibility is high planning time. Search heuristics to reduce planning time exist but often depend on the task or environment. Nevertheless, we expect a robot to learn from its experience and become faster gradually, which is one of the key abilities of humans. Based on this observation, we investigate incremental learning of search heuristics to speed up the planning process on the basis of prior experience in planning and executing a manipulation task.

Even with anthropomorphic robots, we have to consider the differences between the human and the robot in the learning and planning process, what is called the *correspondence problem*. State-of-the-art robot hands differ in geometry, kinematics and dynamics drastically from human hands, e.g. the Schunk Anthropomorphic Hand (SAH), see Figure 1.1a, is 1.5 times the size of an average human hand. Joint-based mappings, e.g. using a linear transformation of joint angles between human and robot, allow to learn full-body motions efficiently. In the context of everyday

manipulation tasks, the reproduction of contacts on the object surface as well as hand postures relative to objects is more promising. The consequence is that a Cartesian-mapping is required, which shares the same disadvantages as learning Cartesian trajectories. Modifying the learned goals and constraints of a manipulation task seems to have a clear advantage in order to solve the corresponding problem.

1.2. Thesis Statement and Preliminary Work

The thesis statement is:

“Flexible autonomous manipulation in human-centered environments is made possible by goal-directed planning based on sophisticated task knowledge, which can be efficiently obtained by learning from human demonstrations. ”

We support the statement by discussing different representations of manipulation tasks and defining a novel representation on the basis of a sophisticated set of constraints. Additionally, we present techniques to learn manipulation tasks and generalize them to increase flexibility in a (semi-) automatic way. We define and combine techniques to learn search heuristics to speed-up planning with constraint relaxation to consider the differences between human and robot in an automatic way. Scalability to different manipulation tasks, without reconfiguration of the learning or execution system, and the comparison with different execution mechanisms is evaluated.

The approach is investigated using multiple everyday manipulation tasks. Due to the limitations of sensor systems as well as dynamics simulations, we make several important assumptions:

1. Models, i.e. geometry, kinematics and (optionally) dynamics, of the robot, objects and the environment are available. All models have to be suitable for dynamics simulation.
2. Objects are assumed rigid bodies, i.e. no cloth or fluids.
3. Anthropomorphic robot hands are used for dexterous manipulation tasks that require force interaction with the fingertips.

In prior work, a sensor environment consisting of stereo cameras, magnetic field trackers and datagloves was developed. The sensor environment was extended with two robot heads, each with pan-tilt-unit, stereo and RGB-D (depth) camera, see Figure 1.3. In order to detect and localize objects in the learning environment, the 2D-vision library IVT [4] and 2D/3D object localization [74] were incorporated. Additionally, tactile sensors to measure forces applied with the fingertips were developed, which will be described in Section 4.1.1.

Different robot systems were available to evaluate the approach. The main robot systems were Albert II and Adero, see Figure 1.1. Since perception, mobility as well as mobile manipulation is not in the scope of this thesis, we restrict explanations to manipulation relevant components. Albert II has a 6-DOF robot arm with a 6D-force sensing device and a 4-DOF robot hand. In contrast to



Figure 1.3.: Sensor environment with magnetic field trackers, datagloves, tactile sensors and two robot heads featuring a stereo camera, a RGB-D camera and a pan-tilt-unit.

Albert II, Adero has an anthropomorphic upper body. Adero has two 7-DOF Kuka Lightweight Arms and two 13-DOF Schunk Anthropomorphic Hands. Kinematic and dynamic models have been defined. 3D models of robots and objects were available. The latter can be found in the KIT Object Models database [59].

1.3. Concept Overview

The representation of a manipulation task has to capture all quantitative and qualitative aspects that are necessary to reproduce the task successfully on a robot system. In contrast to coherent representations, which represent a suitable set of robot trajectories explicitly, we represent quantitative and qualitative task aspects directly. Each task aspect is modeled based on an extensive set of constraints: position, orientation, direction, force, moment, contact, configuration, temporal and arbitrary constraints. The set of constraints is motivated by recent work in constraint-based motion planning. The result is an implicit representation of the manipulation task. We identify each

constraint with a continuous set of robot configurations, in which the constraint will be obeyed, which is called constraint manifold. We discuss characteristics of the representation, which enable efficient sampling of configurations on the constraint manifold as well as calculating the distance to the constraint manifold. We answer the important question how to adapt constraints to changes in the environment, e.g. different object poses. Most constraints depend on coordinate frames, which adapt automatically to changes in object pose. In order to instantiate constraints with novel objects, we use morphing techniques.

The analysis of real-world manipulation tasks shows that constraint manifolds usually change over time. A manipulation task does not consist of a fixed set of constraints but multiple goals and constraint sets. We define a novel representation of manipulation tasks on the basis of constraints, called strategy graph, to consider multiple goals and constraint sets. Goals and constraints are defined using the same representation leading to simpler algorithms, in general. The temporal constraints define a partial order, in which the (sub-)goals have to be reached. In order to execute the task, a concrete time point has to be assigned to each (sub-)goal. A-priori, this assignment is unknown. The consequence is that the set of constraints, which have to be obeyed at a given time point is also unknown, and state-of-the-art constraint-based motion planners cannot be applied. We discuss the connection to Constraint Satisfaction Problem (CSP) theory and integrate CSP-standard tools into Rapidly Exploring Random Trees (RRT) to solve the resulting CSP. Since the complexity is very high we restrict our work in the remaining part of the thesis to strategy graphs with a linear structure. In general, the planning problem is still intractable. The reason is that even simple constraints, e.g. to keep a fingertip in contact with a table surface, correspond to constraint manifolds, which cannot be computed explicitly. In this case, the constraint manifold contains all robot configurations, in which the fingertip is in contact with the table surface. State-of-the-art constraint-based motion planners do not require an explicit representation of the constraint manifold but apply rejection sampling and projection techniques to project arbitrary configurations onto the constraint manifold. We integrate projection techniques from Computer Vision into the state-of-the-art constraint-based motion planner CBiRRT to plan on complex constraint manifolds defined by the strategy graph. By using constraint-based motion planning, manipulation tasks can be executed in environments with different objects, obstacles and start configurations.

In order to model everyday manipulation tasks, we have to define a large number of constraints, e.g. for the fingers, hand and arm as well as relevant objects, choose correct reference frames and parameterize each constraint considering the kinematics, geometry and dynamics of the robot and objects. The manual definition is time-consuming, error-prone and requires expert knowledge.

In this work, we present a novel PbD-approach, see Figure 1.4, to learn strategy graphs based on a set of human demonstrations.

The first step in the PbD process, see Figure 1.4, is the observation of the human during manipulation. Based on prior work datagloves as well as magnetic field position trackers are used to observe finger and hand motion without occlusions. Forces and contacts between objects cannot be measured with sufficient precision but are fundamental aspects of a large number of manipulation

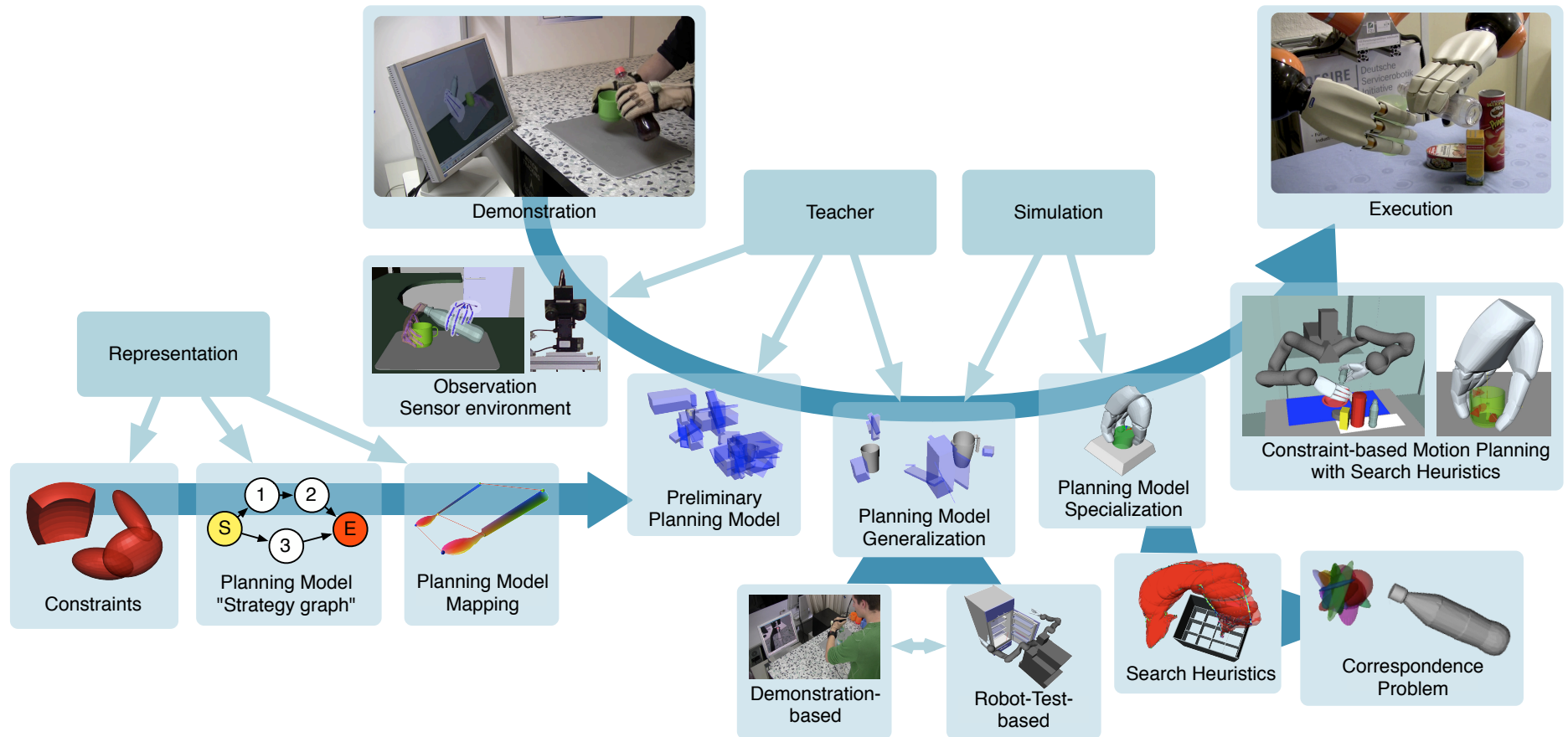


Figure 1.4.: Overview of developed PbD process. The representation of constraints and manipulation tasks, the mapping to different objects and the execution using constraint-based motion planning will be discussed in Chapter 3. Chapter 4 contains the remaining topics: observation of the human teacher, the learning of the preliminary planning model, the generalization of planning models, the specialization to the robot and object models to decrease planning time and consideration of the correspondence problem.

tasks. In order to calculate contacts between the human fingers and objects in simulation, we rely on accurate hand models, which are obtained using a high-precision laser scanner. Additionally, tactile sensors are employed to estimate forces in contact points.

We gather multiple examples of each task as the basis to learn a preliminary planning model. Two important problems are solved: how to build the structure of the strategy graph, e.g. the set of nodes and arcs, and how to define and parameterize the set of constraints. The structure is built using prior work on velocity-based segmentation. The set of constraints has a large influence on flexibility. In general, we prefer a small number of constraints to induce less restrictions on the environments, in which the task can be executed. This refers to a fundamental problem in pattern recognition: how to find the best set of features to train a classifier. In related work, the set of constraints is often predefined. We make only weak assumptions on the constraint set, e.g. consider only detected objects in the environment, and start with a large number of automatically generated constraints.

Constraints, which restrict the hand and finger motion, encode aspects which can be accounted to human morphology, e.g. the distance of the hand relative to the object. These differences in morphology of the human and robot hand have to be considered explicitly in the learning process, which is called the correspondence problem. We follow a novel approach and relax learned constraints, which restrict finger and hand motion. The basis is a workspace comparison of the robot and human hand. Constraints, which restrict object motion and contacts between objects, are not relaxed. In the planning process, fingertip and hand motion can be adapted more freely to reproduce aspects of the task, which we assume independent of human morphology.

The preliminary strategy graph contains a large number of irrelevant constraints, which limits flexibility to different environments. Therefore, the next step in the PbD process is the generalization of the planning model by identifying irrelevant constraints to increase flexibility. We present two complementary approaches to consider the differences in experience with the learning system and number of available demonstrations. In the first approach, additional demonstrations by the human teacher are used to identify irrelevant constraints. In the second approach, the human teacher defines a number of robot-tests, e.g. object arrangements, which the robot has to solve using the learned planning model. We assume that irrelevant constraints prevent a successful solution of the robot-test and generate statistics in the planning process to identify inconsistent constraints. Based on this feedback, we define an optimization problem to find a maximum subset of constraints, which admits a solution to all robot-tests, automatically.

The generalized strategy graph can be executed in a flexible way in different environments on an anthropomorphic robot. Due to the constraint relaxation and reduced number of constraints, the search space is large (and also high-dimensional). The consequence is high planning time. In the next step, we specialize the planning model to the robot and objects in the environment to decrease planning time. In the state-of-the-art, different search heuristics exist to speed-up the planning process. A common limitation is that search heuristics are fixed or learned prior to the execution. Incremental learning, e.g. to learn new search heuristics for situations, in which all other search

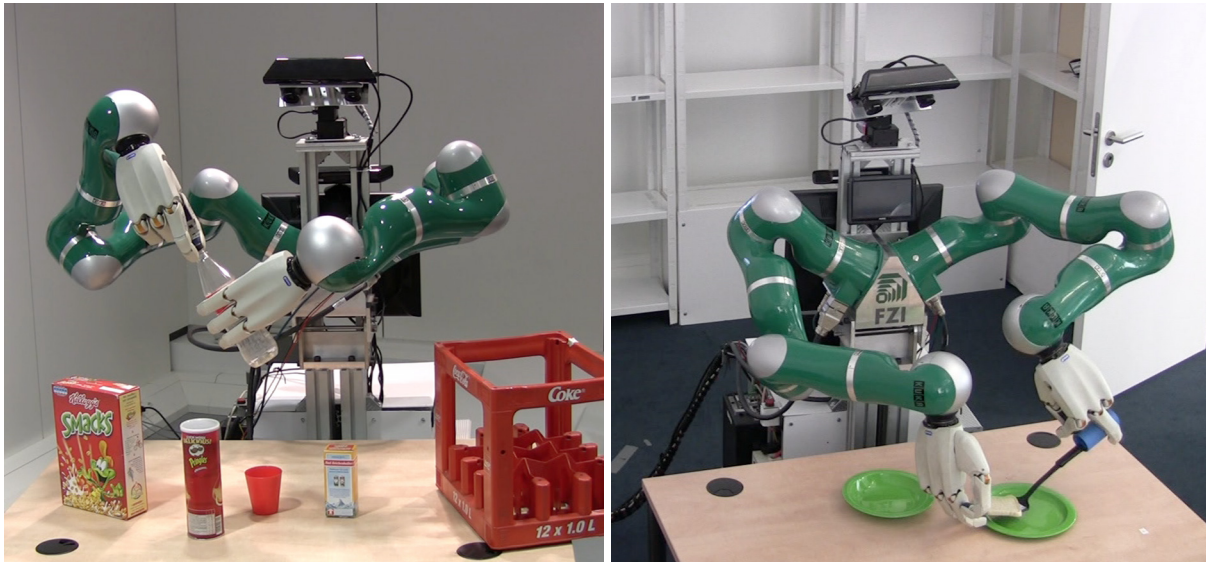


Figure 1.5.: Planning and execution of learned manipulation tasks on Adero: opening a bottle with the fingertips (left) and tool usage (right). - [53]

heuristics fail, is not supported. In this thesis, we learn search heuristics incrementally based on previously planned robot trajectories. Each search heuristic encodes a number of robot trajectories. In the vicinity of an object, a robot trajectory often depends on the geometry of the object, e.g. the motion of the finger to push a slider. Farther away from the object, the object geometry becomes less important and the influence of the start configuration and obstacles increases. Based on this assumption, we integrate a fast-control algorithm into the developed planning algorithm. The control algorithm generates the object-dependent part in an efficient way based on the search heuristics. The remaining part is planned using the original planning model. In the best case, the remaining part is negligible and planning time depends only on the control algorithm. In the worst case, the object-dependent part vanishes and the algorithm reduces to the standard motion planner.

Search heuristics incorporate information about robot morphology, e.g. where to place the robot hand in order to grasp an object, which was not available in the learning process. We relaxed the learned finger and hand constraints to compensate this lack of information. Finally, the search heuristics are projected to the constraints to counteract the relaxation and derive a strategy graph specialized to the robot system.

We evaluate the developed planning algorithm, individual components of the presented PbD-approach as well as the overall system. Focus is on everyday manipulation tasks like pour-in, placing objects in the fridge or opening a bottle. We rely on planning results in simulation, either with fixed or random object arrangements, to get a sufficient data basis. The applicability to real-world environments is shown in multiple experiments on the real robots Albert II and Adero, see Figure 1.5. Qualitative results with the robots Armar IIIb and PR-2 in simulation as well as real execution on the robot Justin show the scalability to different robots.

1.4. Thesis Contribution

In this thesis, we develop a representation of manipulation tasks on the basis of an extensive set of constraints, which cover a large variety of different everyday manipulation tasks. We extend state-of-the-art constraint-based motion planners to generate robot motions online in different situations, e.g. with different objects, obstacles and start configurations. The resulting flexibility is a major advantage compared to related work in the state-of-the-art, which employ control-based mechanisms to execute manipulation tasks.

In order to learn novel manipulation tasks, we enhance PbD-methods to extract relevant goals and constraints of a manipulation task in a (semi-)automatic way based on a small number of human demonstrations. By removing irrelevant goals and constraints, the learned task model, which we call *strategy graph*, is more general and can be applied more flexibly in different situations compared to the related work, which employ simpler generalization techniques.

Additionally, we consider explicitly the differences in morphology of the human, who demonstrates the task, and the robot, who has to execute it autonomously. In contrast to related work in the state-of-the-art, no manually defined task-dependent mapping is required, which leads to less human intervention and therefore more autonomy in the learning process.

The disadvantage of motion planning is high execution time. We decrease planning time by learning search heuristics automatically based on prior planning results. In contrast to related work, search heuristics are learned incrementally and allow to speed-up the execution in situations, in which previously learned search heuristic fail.

The main contributions are summarized as:

1. Manipulation task representation on the basis of goals and constraints to support flexible online motion planning.
2. (Semi-)automatic extraction of relevant constraints and goals of a manipulation task, which limits user interaction and leads to higher flexibility in the planning process.
3. Consideration of the correspondence problem by relaxation of constraints, which refer to human morphology, and automatic tightening based on planning results to incorporate robot morphology.
4. Incremental learning of search heuristics based on prior planning results to restrict the search space for motion planning leading to shorter execution times in real world applications.

1.5. Document Outline

The thesis is composed of six chapters, which group the contributions according to Figure 1.4.

Chapter 2 presents and analyzes related work and state-of-the-art in service robotics with focus on manipulation.

Chapter 3 discusses a concept to represent, map and plan manipulation tasks on the basis of constraints.

Chapter 4 explains the developed approach to learn and post-process manipulation tasks using PbD to increase flexibility and decrease planning time.

Chapter 5 contains the experimental evaluation of the overall system as well as individual components using different robot systems.

Chapter 6 summarizes the thesis, highlights the contribution and discusses open research questions and potential future work.

2. Theoretic Background and Related Work

The first patent for an industrial robot was filed in 1954 by George Devol [31] in the United States of America. Since then, industrial robots revolutionized different applications, e.g. building cars or microcontrollers, and “in 2011, about 165,000 industrial robots were sold worldwide, by far the highest level ever recorded” [6]. Industrial environments are particularly well suited for robotics since they can be partially structured. The environment of the robot, e.g. the work cell and conveyor belt, is fixed and an a-priori known small set of objects is manipulated. If the pose of the manipulated object is constant, a larger number of tasks can be solved using manually defined robot trajectories. In novel applications, e.g. grasping and sorting objects from a conveyor belt, this is insufficient and the robot requires flexibility to move its tool-center-point on a collision-free path to different goal positions. The calculation of such a trajectory is called motion planning, which represents a fundamental skill for robots manipulating their environment, see Section 2.1. For service robots, the manipulation tasks and environments are considerably more complex. One important aspect is the representation of constraints, see Section 2.1.3, which have to be obeyed during manipulation, e.g. maintain a contact with the finger. Constraints form the basis of the definition of planning models, which enables us to plan manipulation tasks in an automatic, flexible and goal-directed way.

The cost of flexibility is high planning time, usually ten seconds or more for simple Pick&Place tasks. The reasons are narrow passages in the search space leading to a large number of costly collision calculations or a high-dimensional search space. In order to reduce planning time, search heuristics exist, which restrict the search space. In Section 2.2, we define state-of-the-art search heuristics, which approximate the collision-free portion of the search space online to reduce the number of collision calculations or use multiple planner queries to generate a sampling distribution allowing to generate useful configurations with higher probability.

The planned robot trajectory is executed on the real robot under the assumption that robot and object models are highly precise and the current configuration, e.g. robot and object poses, is fully observable. The first assumption can be maintained by using a high precision laser scanner to obtain accurate object models [59]. The latter usually does not hold in service robot applications since objects are localized relatively to the robot using onboard sensors like stereo or depth cameras. In this case, control algorithms are applied to adapt the robot motion to follow the planned trajectory. Constraints define correlations between the robot and object motion, which are exploited to adapt the motion in a goal-directed way. In Section 2.3, we discuss task frames and constraint-based programming, which offer a well understood tool set to derive closed-form controllers to execute manipulation tasks with constraints.

The remaining question is how the planning model, i.e. constraints and goals, of a manipulation task are defined in the first place. In Chapter 1, we motivated that explicit demonstrations by a human teacher, who is an expert in manipulation, are viable to generate the skills of the robot. Multiple learning paradigms with different representations and execution mechanisms exist in the field of Programming by Demonstration (PbD) or Imitation Learning. In Section 2.4, we give an overview of probabilistic, dynamic, bio-inspired, contact-based and symbolic PbD approaches.

Finally, we analyze the related work in respect to requirements arising naturally from the thesis statement. The requirements and evaluation are summarized in Section 2.5.

2.1. Planning of Manipulation Tasks

Planning enables an autonomous robot to adapt its execution to changes in the environment, e.g. different objects and obstacles, and reach a certain goal while obeying a number of constraints. Russel et al. describe planning as “the task of coming up with a sequence of actions that will achieve a goal” [88, page 375]. In order to determine this sequence, a robot has to predict the effects of its actions on the environment. The prediction of complex effects requires general purpose kinematic or dynamic simulations, which are grounded in geometric, kinematic and (optionally) dynamic models of objects, obstacles and the robot. This dependency on accurate and precise models represents a major limitation of planning.

Depending on different characteristics of the planning problem, e.g. a discrete or continuous state space, different planning paradigms exist, e.g. symbolic planning, probabilistic planning or motion planning. In this thesis, we concentrate on the latter. The goal of motion planning is to generate a collision-free continuous path in the configuration space of the robot starting in the current configuration and ending in a predefined goal configuration. The planning problem is in PSPACE and we cannot expect to find a solution for arbitrary problems efficiently. If the environment is fixed, e.g. in industrial applications, a map of collision-free paths can be computed offline and used online to plan a path from a given start to a goal configuration in a short amount of time. We give a short overview of these Roadmap-planners in Section 2.1.1.

Service robots operate in the human environment, where object, obstacle and robot poses change with in each planner query. In this situation, motion planners have to be used, which generate a collision-free path online. We discuss the Rapidly Exploring Random Tree planner in Section 2.1.2. For service robots, manipulation tasks are often more complex, e.g. they are subject to different constraints like holding an object upright, contain more complex goal descriptions and require coordinated motion of robot fingers and hands. We summarize recent developments on the representation of constraints and goals, which form the basis of the definition of planning models, in Section 2.1.3. The set of configurations, which obeys a set of constraints, is called the constraint manifold. In the planner, random configurations have to be projected to a constraint manifold efficiently. We discuss different projection techniques in Section 2.1.4. In Section 2.1.5,

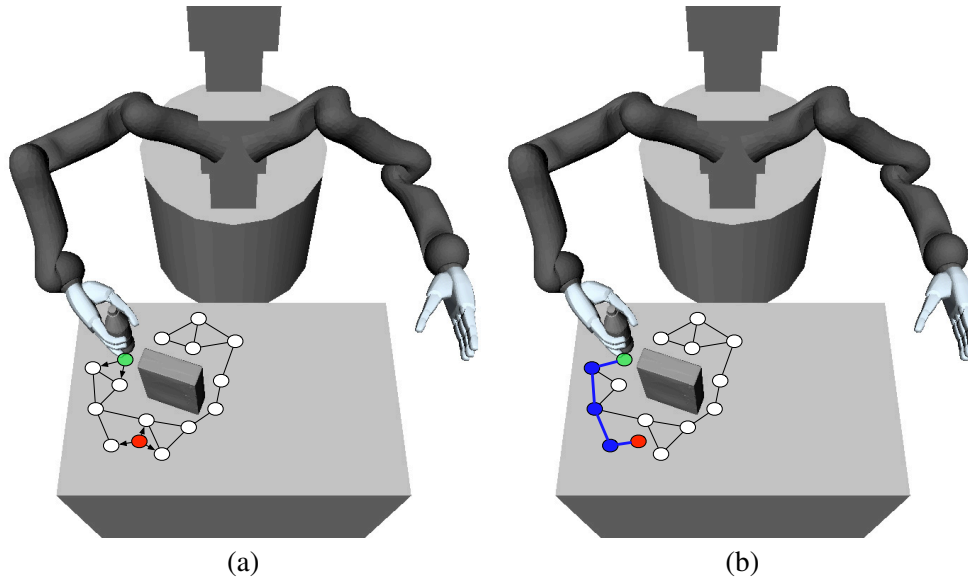


Figure 2.1.: Probabilistic roadmap: nodes (white), collision-free connections (lines), start configuration (green), goal configuration (red). (a) start and goal configuration are connected by a collision-free path to the roadmap (arrows). (b) graph search is applied to find a path between the connected nodes resulting in the solution (blue).

we discuss how the RRT and PRM motion planner can be adapted to consider constraints in the planning process based on the defined projection techniques.

2.1.1. Probabilistic Roadmaps

Sorting loose material, e.g. screws, from one conveyor belt to a box is a standard problem in robotics. A screw has to be grasped at a random position, moved on a collision-free path above a crate and released. In this situation, the environment is usually fixed, i.e. the robot and conveyor belts are the only moving objects. The task is repetitive but since the conveyor belts move, the grasp and release poses vary. Based on the assumption that the environment is fixed, a probabilistic roadmap (PRM) [60] of collision-free paths can be computed prior to the planning process to speed up the generation of a collision-free path. In this Preprocessing phase, the goal is to generate a roadmap, which can be efficiently used to answer planner queries. The roadmap approximates the free space in the sense that a collision-free path from an arbitrary point in the free space to a point in the roadmap can be found. In the Query phase, a start and goal configuration is given and the problem is to find a collision-free path between both configurations using the roadmap. In PRMs, both configurations are connected with the roadmap, i.e. a fast local planner is used to generate a collision-free path to a nearby configuration in the roadmap, and graph search, e.g. A^* , is applied to find a path in the roadmap. The whole process is visualized in Figure 2.1.

Algorithm 1 shows the basic principle [69, p. 237ff] to construct a roadmap. $\alpha(i)$ is a dense sequence of random configurations, e.g. the uniform distribution on configuration space $\alpha(i) \sim \mathcal{U}(\mathcal{C})$. If $\alpha(i)$ is collision-free (line 4) it will be added to the roadmap (line 5). We calculate nearby

Algorithm 1 PRM algorithm to construct a roadmap, [69, p. 237ff]

```

1:  $\mathcal{G}$ .init();
2:  $i \leftarrow 0$ 
3: while  $i < N$  do
4:   if  $\alpha(i) \in \mathcal{C}_{free}$  then
5:      $\mathcal{G}$ .add_vertex( $\alpha(i)$ );
6:     for  $q \in \text{NEIGHBORHOOD}(\alpha(i), \mathcal{G})$  do
7:       if ( not  $\mathcal{G}$ .same_component( $\alpha(i), q$ ) ) and  $\text{CONNECT}(\alpha(i), q)$  then
8:          $\mathcal{G}$ .add_edge( $\alpha(i), q$ );
9:       end if
10:    end for
11:   end if
12:    $i \leftarrow i + 1$ 
13: end while

```

configurations in the roadmap (line 6). If a nearby configuration q is in a different component and we can generate a collision-free path from $\alpha(i)$ to q (line 7), we add an edge from $\alpha(i)$ to q (line 8). It is true that between all vertices in a component a collision-free path exists.

The implementation of the function NEIGHBORHOOD has high influence on the overall planning time. If we set the neighborhood to all nodes in the roadmap, planning time will be maximal. Lavalley [69, p. 237ff] lists four different implementations, which will be summarized here.

Nearest k The k nearest neighbors to $\alpha(i)$ are returned, which can be implemented efficiently using, e.g., kd-trees.

Component k The connected components of the roadmap are calculated and for each component, the Nearest k -step is applied.

Radius r, k Choose up to k vertices, which are closer than r to $\alpha(i)$, i.e. which lie in the sphere $\mathcal{B}(\alpha(i), r)$.

Visibility In [96], the concept of guard and connector vertices was introduced. A guard is a vertex, which cannot be connected to other guards, i.e. CONNECT fails. A connector is defined as a vertex q , which can be connected to at least two guards q_1, q_2 , i.e. $\text{CONNECT}(q, q_1)$ and $\text{CONNECT}(q, q_2)$ is valid. The algorithm is adapted to create a roadmap with small number of vertices, where each vertex is a guard or connector. $\alpha(i)$ will be added if it cannot be connected to one of the guards (it will become a guard itself) or if it can be connected to at least two different connected components of the roadmap (it will become a connector).

Visibility is one example to generate roadmaps, which can be queried more efficiently and therefore allow shorter planning times online. Different approaches generate only vertices near the boundary of the space of collision-free configurations [2] and [14], in narrow passages [44], on the

medial axes [110], [43] and [72], i.e. a maximum distance to the boundary, or use statistics about connection failures of the generated vertices in [60].

2.1.2. Rapidly-Exploring Random Trees

Service robots operate in the human environment, in which objects, obstacles and the robot pose relative to the environment change in each planner query. The consequence is that we cannot guarantee that a previously calculated roadmap holds any collision-free paths and we have to generate a roadmap in each planner query. Since we do not need information about the whole free space but only the relevant subspace for the current planner query, computational effort and planning time is wasted by generating a roadmap. In this section, we discuss the Rapidly-Exploring Random Tree (RRT) [68] sampling-based motion planner, which operates online without the need to generate a roadmap. We focus the explanations on an uni-directional RRT, which grows a single tree in the start configuration. The bidirectional variant will be discussed in Section 2.1.5.

In the unidirectional RRT, a tree is grown from the start to the goal configuration. The tree explores new areas of the search space very efficiently. With increasing planning time, the tree approximates the free space, which can be reached from the start configuration. The direction of growth is controlled by sampling random configurations in the search space. Algorithm 2 based on [68] shows the implementation details.

Algorithm 2 RRT($\theta_{start}, \theta_{target}$)

```

1: AddNode( $\theta_{start}, t_{start}$ )
2: while true do
3:    $\theta_{random} \leftarrow$  RandomConfiguration( $\theta_{target}, \delta$ )
4:    $\theta_{nn} \leftarrow$  NearestNeighbor( $\theta_{random}$ )
5:    $\theta_{extend} \leftarrow$  Extend( $\theta_{nn}, \theta_{random}$ )
6:   if not HasCollision( $\theta_{nn}, \theta_{extend}$ ) then
7:     AddNode( $\theta_{extend}$ )
8:     AddParent( $\theta_{extend}, \theta_{nn}$ )
9:     if GoalTest( $\theta_{extend}$ ) then
10:      return Path from  $\theta_{start}$  to  $\theta_{extend}$ 
11:    end if
12:  end if
13: end while

```

The algorithm starts by initializing the tree root with the start configuration (line 1). In each iteration, a random configuration is sampled uniformly or, with probability δ , the goal configuration (line 3). The nearest neighbor in the tree is calculated (line 4), e.g. efficiently using kd-Trees. We take an ϵ -step from the nearest neighbor to the random configuration (line 5) and add it to the tree (line 7,8) if it is collision-free (line 6). Finally, we test if the goal configuration was reached (line 9) and return the solution by traversing the tree in reverse order (line 10).

The planning process is visualized in Figure 2.2. The tree rapidly explores large regions of the configuration space, in which no tree nodes reside. This was analyzed by Kuffner et al. [65] by looking at the voronoi regions of each node and is repeated here. In Figure 2.3, the voronoi regions after the third and fourth iteration are shown. Since we generate nodes in a small distance of the root node in the beginning, voronoi regions will be larger the farther away from the root node. The voronoi regions define a partition of the configuration space. A voronoi region of a node contains all configurations, which are closer to the node than to any other node in the tree. If we draw a random sample in the configuration space, the probability that a specific node will be extended is proportional to the area of its voronoi region. The conclusion is that nodes will be extended in the beginning, which are farther away from the root. In each extension, new nodes are added resulting in a new set of voronoi regions and the average size of voronoi regions decreases. The tree converges to a dense set, i.e. it gets arbitrarily close to any point in the configuration space.

The standard RRT planner is insufficient due to the rapid exploration, when the solution passes a narrow passage. Different extensions exist to focus search on problematic parts of the configuration space. Kuffner et al. [66] recognized the symmetry of the planning problem, i.e. planning from start to goal or vice versa is identical. They grow one tree for each goal configuration and one tree in the start configuration alternately and balance the growth of both trees with the goal to have the same number of nodes. In each iteration, both trees grow toward each other and the planning problem is solved, when both trees are connected. Since it has less parameters than the standard RRT planner (no δ), it is used widely. In dynamic domain RRTs (dd-RRT) [113], a different sampling distribution is defined. A sphere with radius d_{near} is created around each configuration in the tree. A sample is drawn from the intersection of the search space with the union of spheres, i.e. only configurations close to the tree are generated. The approach was extended in [48] to adapt the threshold automatically during motion planning. Strandberg [102] introduces a RRT planner, which spawns a new tree in a random configuration with certain probability when the following condition applies: the configuration is collision-free but cannot be connected to the tree. The concept generalizes the idea of bidirectional search and allows to solve planning problems with multiple traps. In [114], random configurations, which are in collision, are retracted to generate configurations in narrow passages or close to obstacles.

2.1.3. Constraints

In the example in the previous section, the implicit assumption was that the bottle had to be moved on the table and could not be lifted. The search space in this case would be the position x, y and orientation θ of the bottle on the table. The planning solution would be a path for the bottle. By using inverse kinematics for the robot arm, a trajectory in configuration space could be generated. The latter is problematic due to singularities of the arm, collisions of the arm with the environment and generalizes poorly to more complex tasks, which involve coordinated motion of hands and fingers. In motion planning, the problem is solved by planning in the joint space of the robot, i.e.

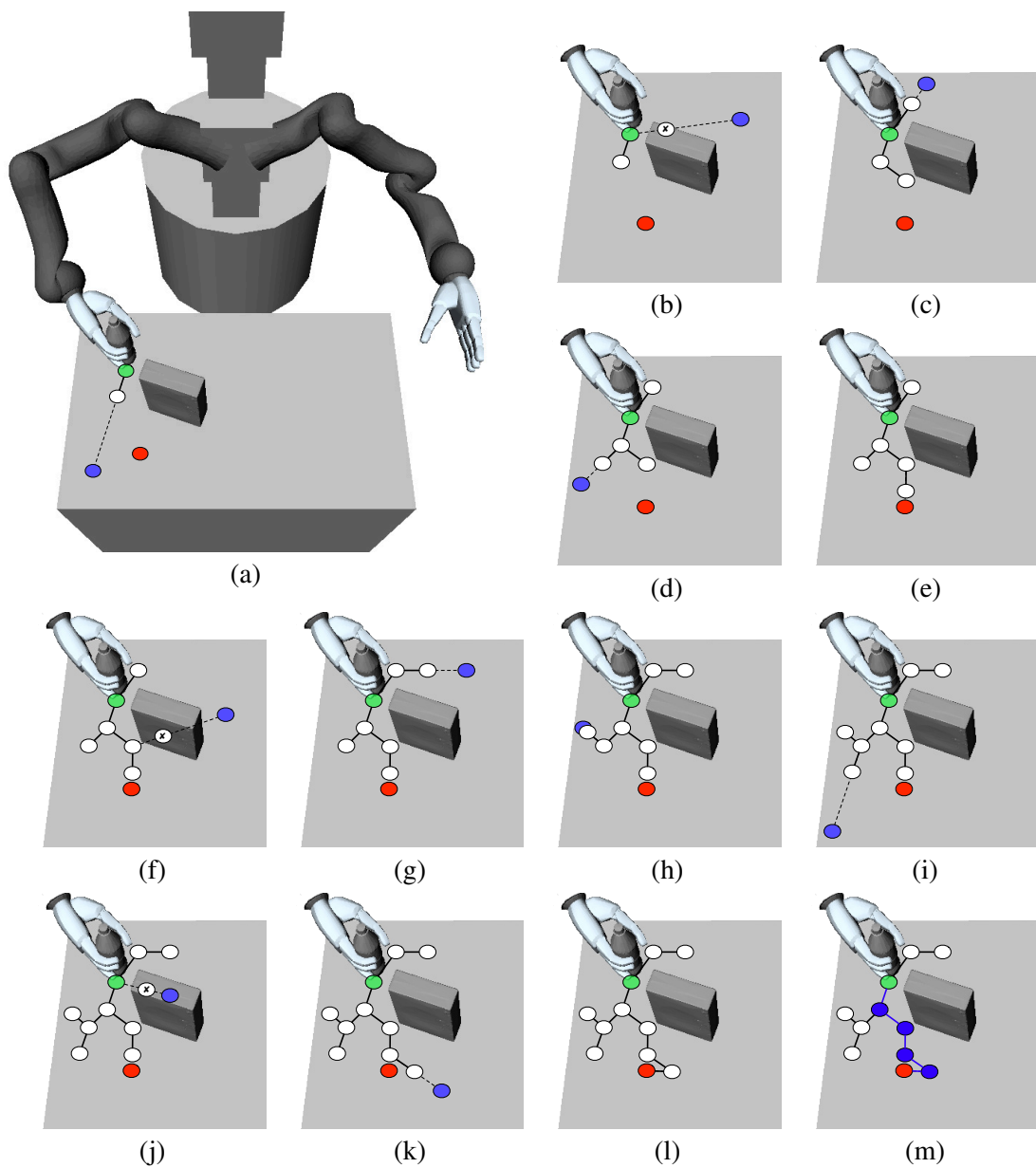


Figure 2.2.: Unidirectional RRT: nodes (white), collision-free connections (lines), start configuration (green), goal configuration (red), random configuration (blue). In order to extend the tree, a random configuration is chosen (blue) and an ϵ -step is taken from the nearest neighbor in the tree (dotted line). Nine extensions are shown (a - l). With small probability the goal configuration is chosen instead of a random configuration (e, l). The result (blue path) is a collision-free path from start to goal (m).

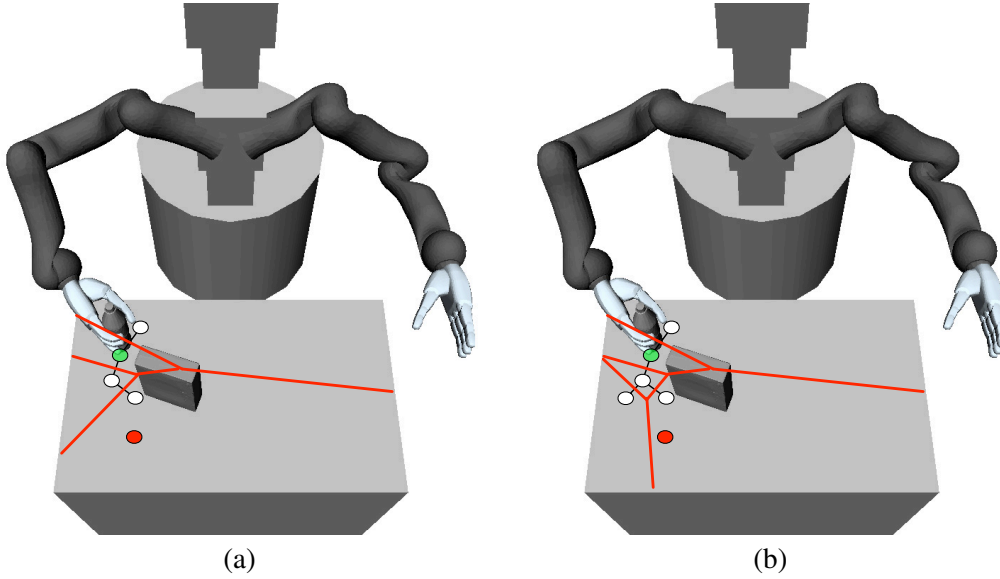


Figure 2.3.: The probability to extend a node in the tree is proportional to the area of its voronoi region. (a) shows the voronoi regions after the third iteration and (b) after the fourth iteration.

we plan directly how the joints of the robot have to move. An important question arises: how can we restrict that the generated arm motion leads to a motion of the bottle, in which it will not be lifted from the table?

The representation of such restrictions in a way compatible to a sampling-based motion planning leads to the definition of a constraint. The joint limits $l_j \leq \theta_j \leq u_j$ represent an example of a joint space constraint, which can be represented as a hypercube $[l_j, u_j]^n$. More interesting are Cartesian constraints, since they refer to objects in the environment and adapt to different object poses. Stilman et al. [101] and Berenson et al. [9], [10] define Cartesian constraints based on two coordinate frames 0T_1 and 0T_2 , where the first coordinate frame moves relative to the second coordinate frame. In our example, 0T_1 is a coordinate frame in the center of mass of the bottle and 0T_2 is a coordinate frame in the center of the table surface. If we want to restrict the motion of 0T_1 relative to 0T_2 , first we calculate the relative 6D homogenous transformation matrix 2T_1 . We transform into a 6D vector using $\Delta x = (x, y, z, roll, pitch, yaw)^T$, where x, y, z is the translation and $roll, pitch, yaw$ the roll-pitch-yaw representation of the rotation part. Stilman et al. [101] use a selection matrix C with $c_i \in \{0, 1\}$:

$$\begin{pmatrix} c_1 & & \\ & \dots & \\ & & c_6 \end{pmatrix} \quad (2.1)$$

to choose dimensions, in which the coordinate frame might move. The constraint is defined as

$$0 = C \Delta x \quad (2.2)$$

If the constraint is violated, the distance to the constraint manifold is given by $\|C\Delta x\|$. Berenson et al. [9] extended this representation to consider lower l_j and upper bounds u_j . The constraint is obeyed, if Δx lies in the hypercube defined by the boundaries

$$\begin{pmatrix} l_1 \\ \dots \\ l_6 \end{pmatrix} \leq C\Delta x \leq \begin{pmatrix} u_1 \\ \dots \\ u_6 \end{pmatrix} \quad (2.3)$$

or $l_j \leq c_j\Delta x_j \leq u_j$. The representation allows to consider constraints in the motion planning process in an efficient way.

2.1.4. Constraint Projection

The constraint definition in the previous section allows to compute the distance from a given configuration to the constraint manifold. In our previous example, let Δx be the relative transformation between the bottle center of mass and the table surface. The constraint to move the bottle on the table is defined as

$$\begin{pmatrix} -500 \\ -300 \\ 800 \\ 0 \\ 0 \\ -180 \end{pmatrix} \leq \begin{pmatrix} x \\ y \\ z \\ roll \\ pitch \\ yaw \end{pmatrix} \leq \begin{pmatrix} 500 \\ 300 \\ 800 \\ 0 \\ 0 \\ 180 \end{pmatrix} \quad (2.4)$$

The distance is defined as the maximum of the distance in each dimension $\max(l_i - x_i, \max(x_i - u_i, 0))$. If the bottle is raised 10cm above the table but held upright, the distance will be 10. Even in this simple example, the constraint manifold is infinitely thin, i.e. Lebesgue-measure 0, and drawing a random configuration, in which the constraint is obeyed, has probability 0. If we want to extend the RRT to plan on constraint manifolds, we need an algorithm to generate random configuration on the constraint manifold efficiently. Stilman et al. [101] define projection techniques, which project a given random configuration to the constraint manifold by using the distance information. We describe informally two example projection techniques: Randomized Gradient Descent (RGD) and Fraction Retraction (FR). Since projection techniques are mainly used in the Extend function, we show how it can be extended to consider configurations on a constraint manifold. A (random) configuration θ_{random} and nearest neighbor θ_{nn} are given. We calculate the configuration in distance ε from θ_{nn} to θ_{random} . Finally, the resulting configuration θ_s is projected to the constraint manifold.

RGD, see Figure 2.4a, requires only a scalar value representing the distance from the current configuration to the constraint manifold. It samples a random direction to generate a configuration in distance $\varepsilon' \ll \varepsilon$ to θ_s . If the distance to the constraint manifold is smaller than in θ_s , θ_s will be replaced by the generated configuration. If the distance is smaller than a threshold δ , the iteration

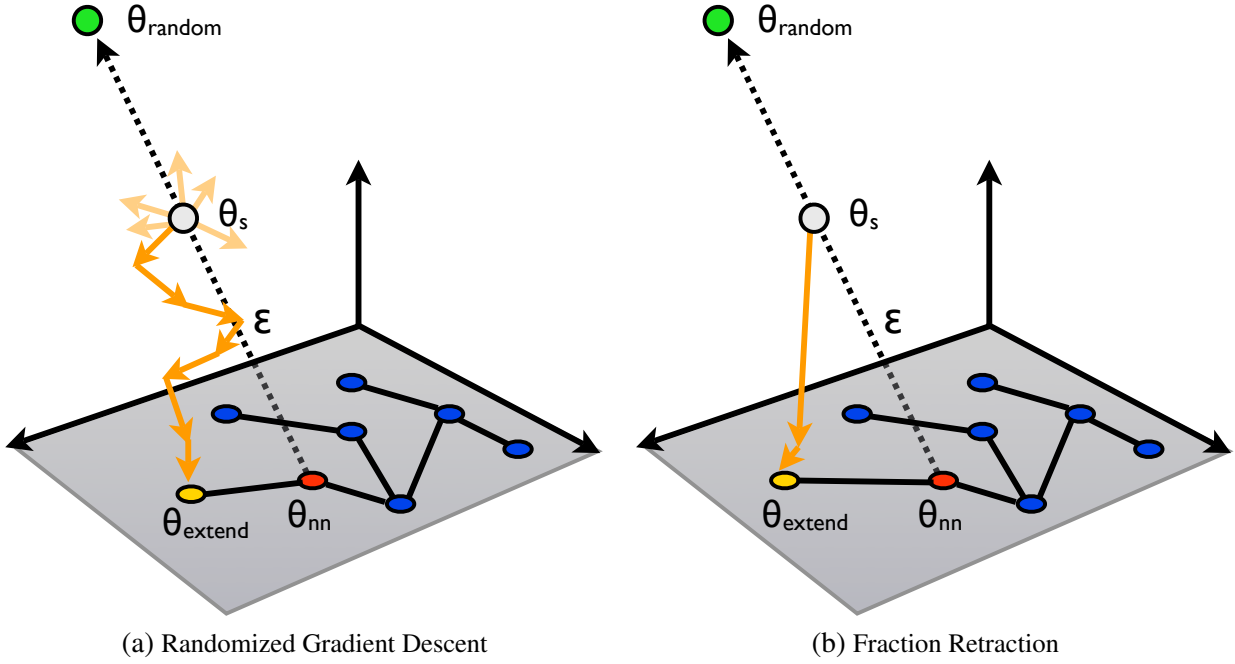


Figure 2.4.: Projection techniques to project a random configuration θ_s to a configuration θ_{extend} on the constraint manifold, [101].

stops. If the distance of θ_s to θ_{nn} is less than ϵ , the projection was successful. RGD approximates the gradient of the distance to the constraint manifold. The main advantage is numerical stability and simplicity, i.e. only a scalar distance value is required.

If a distance vector Δx to the constraint manifold is available, e.g. in the example it would be $(0, 0, -10, 0, 0, 0)^T$, we do not have to approximate the gradient resulting in a potential speed-up. The problem is that the configuration has to be adapted but the constraints define a distance function in the Cartesian space. In FR, see Figure 2.4b, the Jacobian of the manipulator $J(\theta)$ is used to project the Cartesian distance to the configuration space. First, the pseudo-inverse $J^\dagger(\theta)$ of the Jacobian is calculated. θ_s is iteratively replaced by $\theta_s - J^\dagger(\theta_s)\Delta x$. In each iteration, the distance Δx of θ_s to the constraint manifold is updated. The main advantage of Jacobian-based projection techniques is less evaluations of the constraint distance resulting in shorter computation times. Problematic are singular configurations and configurations close to the joint limits.

2.1.5. Constraint-based Motion Planning

The constraint representation and projection techniques are general and can be applied to plan on constraint manifolds with different planning techniques. In this subsection, we discuss state-of-the-art techniques to extend the RRT and PRM to plan on constraint manifolds.

Since the RRT is extended by ϵ -steps, it can be easily adapted to plan on constraint manifolds. Stilman et al. [101] replace the function $\text{Extend}(\theta_{nn}, \theta_{random})$ in line 5 in Algorithm 2 with ConstrainedExtend . $\text{ConstrainedExtend}(\theta_{nn}, \theta_{random})$ is implemented as a constraint projection tech-

nique, see Section 2.1.4, and returns a configuration in maximum distance ε to θ_{nn} on the constraint manifold.

The PRM planner is extended in a similar way. During the generation of the roadmap, the function $\text{CONNECT}(\alpha(i), q)$ is applied to connect the random configuration $\alpha(i)$ with the node q , see line 7 in Algorithm 1. q is already on the constraint manifold but not $\alpha(i)$ (and it cannot be sampled efficiently). First, we project $\alpha(i)$ on the constraint manifold resulting in $\alpha(i)'$. The CONNECT function has to generate a path on the constraint manifold from $\alpha(i)'$ to q . A simple implementation is shown in Algorithm 3. We apply the function ConstrainedExtend iteratively to make ε -steps on the constraint manifold from $\alpha(i)'$ to q .

Algorithm 3 $\text{CONNECT}(\theta_{\text{start}}, \theta_{\text{target}})$

```

1:  $\theta_{\text{result}} \leftarrow \emptyset$ 
2: while  $\theta_{\text{result}} \neq \theta_{\text{target}}$  do
3:    $\theta_{\text{current}} \leftarrow \text{ConstrainedExtend}(\theta_{\text{result}}, \theta_{\text{target}})$ 
4:   if  $\theta_{\text{current}} = \emptyset$  then
5:     return  $\theta_{\text{result}}$ 
6:   else
7:      $\theta_{\text{result}} \leftarrow \theta_{\text{current}}$ 
8:   end if
9: end while
10: return  $\theta_{\text{result}}$ 

```

By integrating projection techniques into the planning algorithms, motions, which are subject to collision as well as simple Cartesian constraints, can be planned efficiently.

2.2. Efficient Search Heuristics for Motion Planning

In constraint-based motion planning, the search space is defined implicitly as the intersection of the constraint manifolds created by individual constraints. In our example, the no collision constraint creates a constraint manifold consisting of all configurations, which are collision-free. The bottle constraint induces a thin constraint manifold with configurations, where the bottle stands on the table surface. Computation time for the constraint distances differ by a magnitude, i.e. with PQP [106] a collision query usually takes a few milliseconds while the distance of the bottle constraint is calculated in microseconds. Constraints with high computation time have a significant impact on the overall planning time, which is usually in the order of 10 seconds. If we are able to reduce the number of time consuming constraint evaluations, a significant speed-up can be achieved.

We discuss two approaches to reduce planning time for constraint-based motion planning. In Section 2.2.1, we discuss the decomposition of the workspace into more simple representations, which allow to reduce the number of collision queries during planning. In Section 2.2.2, workspace biasing approaches are presented. The idea is to learn a non-uniform sampling distribution assign-

ing a higher probability to configurations, which may be useful in the planning process, potentially leading to a decrease in planning time.

2.2.1. Workspace Decomposition

Brock et al. [16] decompose the planning problem into two subproblems P_1 and P_2 . In P_1 the goal is to generate a tunnel T , i.e. a subset of the workspace, which includes the swept volume of at least one solution to the planning problem. P_2 is the subproblem to generate a solution to the planning problem using the tunnel T to reduce planning time.

The first subproblem is solved using wavefront expansion, which is explained in more detail in [16] and is summarized here. Let s be the start and g the goal configuration. The algorithm starts by calculating the distance r of the robot to the closest object in s and generating a sphere centered at s with radius r . The sphere is added to a priority queue, where the priority is calculated as the minimum distance to the goal: $|s - g| - r$.

In each iteration, the sphere with highest priority is removed from the queue and added to a tree structure, which represents the current state of the free space approximation. Samples on the surface of the sphere are drawn uniform randomly. If a sample is not contained in one of the spheres in the tree, the distance and priority are calculated and a new sphere is added to the priority queue. The algorithm stops if the priority queue is empty or the start and goal configuration are connected by a tunnel (with sufficient diameter). The result is an approximation of the free space as a tree of spheres. The tunnel is the sequence of spheres from the start to the goal configuration. P_2 is solved by defining a potential field based on the tunnel, i.e. the parent-child relationship between spheres and the distances of the sphere centers, and distance to local obstacles.

The priority queue implements a best-first search. In Figure 2.5, the tunnel for the task to place a bottle in a crate is shown. Since the direct path from the start to goal configuration is blocked by the crate, only a small number of spheres is generated in direction of the solution path.

2.2.2. Workspace Biasing

Zucker et al. [116] describe a general approach how different features on a discretized workspace can be weighted based on previous planner queries to generate a sampling distribution, which allows to reduce planning time. In the experiments, features are Gaussian convolutions of voxel occupancy, visibility of voxels and elliptical path distance. Let $d(x, y)$ be the optimal distance from voxel x to voxel y . The elliptical path distance is $d(x, x_{start}) + d(x, x_{goal}) - d(x_{start}, x_{goal})$, where x is the current voxel, x_{start} is the voxel with the start configuration and x_{goal} the voxel with the goal configuration. Each component in the feature vector $f(x)$ corresponds to the scalar value of a feature. Let w be a weight vector. The sampling distribution $P(w, x)$ is defined as

$$P(w, x) \sim \exp\left(w^T f(x)\right). \quad (2.5)$$

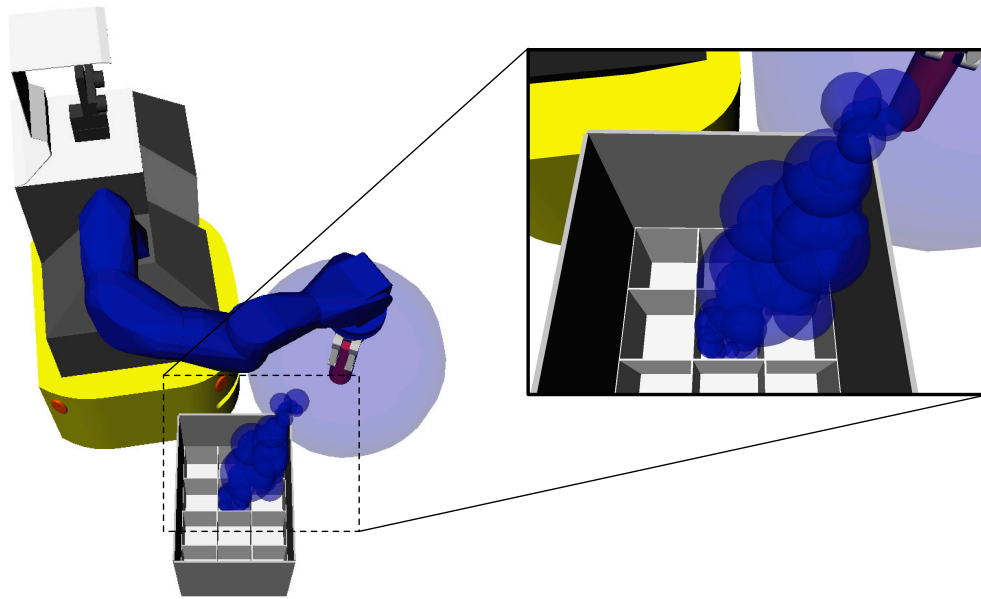


Figure 2.5.: Calculated tunnel in workspace using wavefront expansion.

$P(w, x)$ is a Gibbs distribution. If $w = 0$, $P(w, x)$ is uniform. If $w_i \rightarrow \infty$, $P(w, x)$ will only be greater zero, where $f_i(x)$ is maximal, see [116]. In order to sample a random configuration, Zucker et al. draw a random voxel according to $P(w, x)$ and apply a manually defined transformation to calculate a matching configuration, e.g. for the planar robot in Figure 2.6, a configuration is sampled, which leaves the tool center point in the voxel. They interpret the sampling distribution as a stochastic policy with the goal to find a distribution, which maximizes a reward function. In this case, the reward function is the number of planner queries solved per time frame. Based on multiple planner queries, the weight vector is adapted to find a distribution leading to short planning times. The optimal weight vector depends on the environment, e.g. if a lot of obstacles are present, voxel occupancy might be weighted higher, and thus the approach is suited best for static environments. Zucker et al. did not analyze the reduction in planning time.

Van den Berg et al. [107] analyze the workspace for narrow passages using cell decomposition. The cells are clustered based on the "watershed transform" to form regions, see Figure 2.7a. A weight is assigned to each region to reflect its role as a narrow passage for the robot. In the planning process, a region is drawn randomly proportional to its weight. In the region, a 3D vector is drawn with a uniform distribution. Since the 3D vector does not specify all degrees of freedom of the robot, the other DOF are chosen uniformly randomly. Figure 2.7b shows a set of random samples, with higher concentration in narrow passages. Depending on the objects in the environment, cell decomposition is time consuming and should be done in a Preprocessing phase, e.g. in a PRM planner. Since only the translational degrees of freedom are considered, the approach is suited for free-floating robots, which can be approximated by a sphere, e.g. mobile platforms.

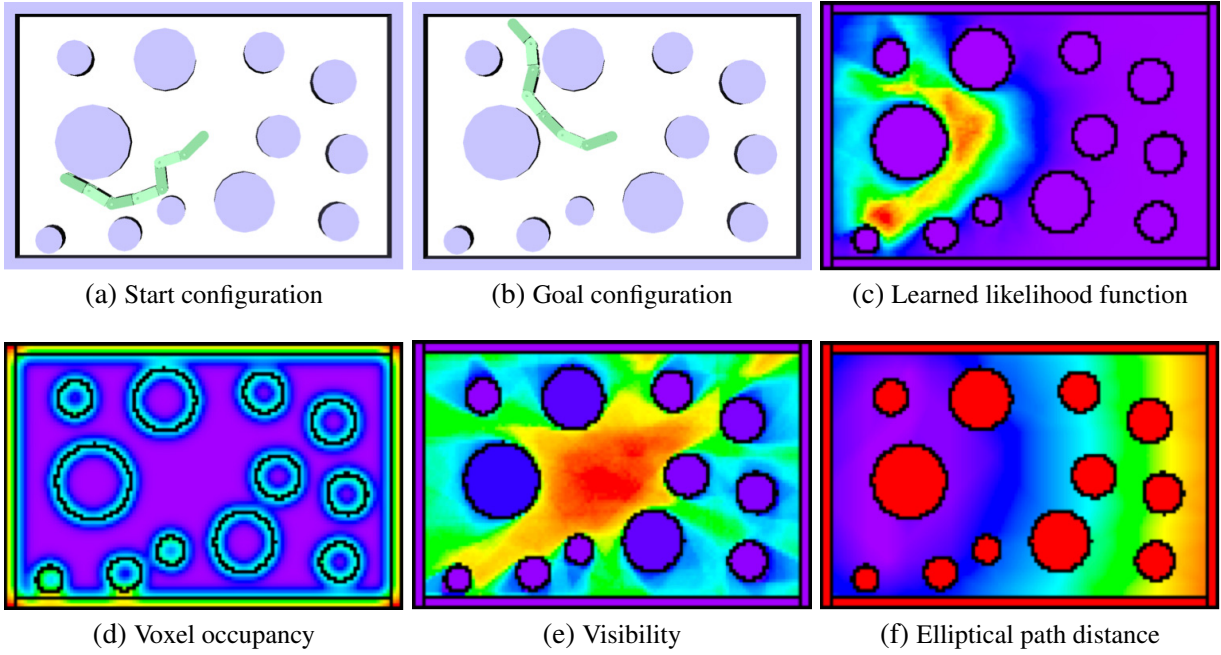


Figure 2.6.: Workspace Biasing. A planar robot with 7 DOF has to move from start (a) to goal (b). The likelihood function in (c) is learned based on a weighting of features (d, e, f), [116].

2.3. Execution of Compliant Motion Tasks

Most manipulation tasks require more or less complex compliant motions, i.e. during manipulation the robot has to cope with a number of contacts, e.g. between a grasped tool and the environment, in a safe and predictable way. In motion planning, constraints represent an efficient way to represent such restrictions but the assumption that the environment is fully observable and robot and object models are highly accurate, is often invalid. In real-world scenarios, the robot has to react constantly to measured forces while adapting a planned robot trajectory to cope with uncertainty.

A framework to define such compliant motions is the Task Frame Formalism (TFF) introduced by Mason [73] and later formalized by Bruyninckx et al. [17], which is summarized here. The basis to define contact situations is the twist vector $\mathbf{t} = (v^T \ w^T)^T$, consisting of linear velocities v and angular velocities w , and wrench vector $\mathbf{w} = (f^T \ m^T)^T$, consisting of the linear forces f and moments m . In a contact situation with the frictionless reaction forces w , the (rigid) object in contact has to execute an instantaneous motion defined by \mathbf{t} , which is reciprocal to \mathbf{w} , i.e.

$$\mathbf{w}^T \mathbf{t} = w^T m + v^T f = 0 \quad (2.6)$$

In a contact situation, the set of twists and the set of ideal contact wrenches are reciprocal vector spaces. The sum of the dimensions is always 6.

A Task Frame (TF) is defined as an orthogonal basis to represent all possible \mathbf{t} and \mathbf{w} satisfying Equation 2.6. Each basis vector is used once as an axial vector $(x_t, y_t, z_t)^T$ to specify force and linear velocity and once as a polar vector $(a_{xt}, a_{yt}, a_{zt})^T$, which allows to define angular velocity and moment. The user specifies the (task-dependent) orthogonal basis and chooses n velocity

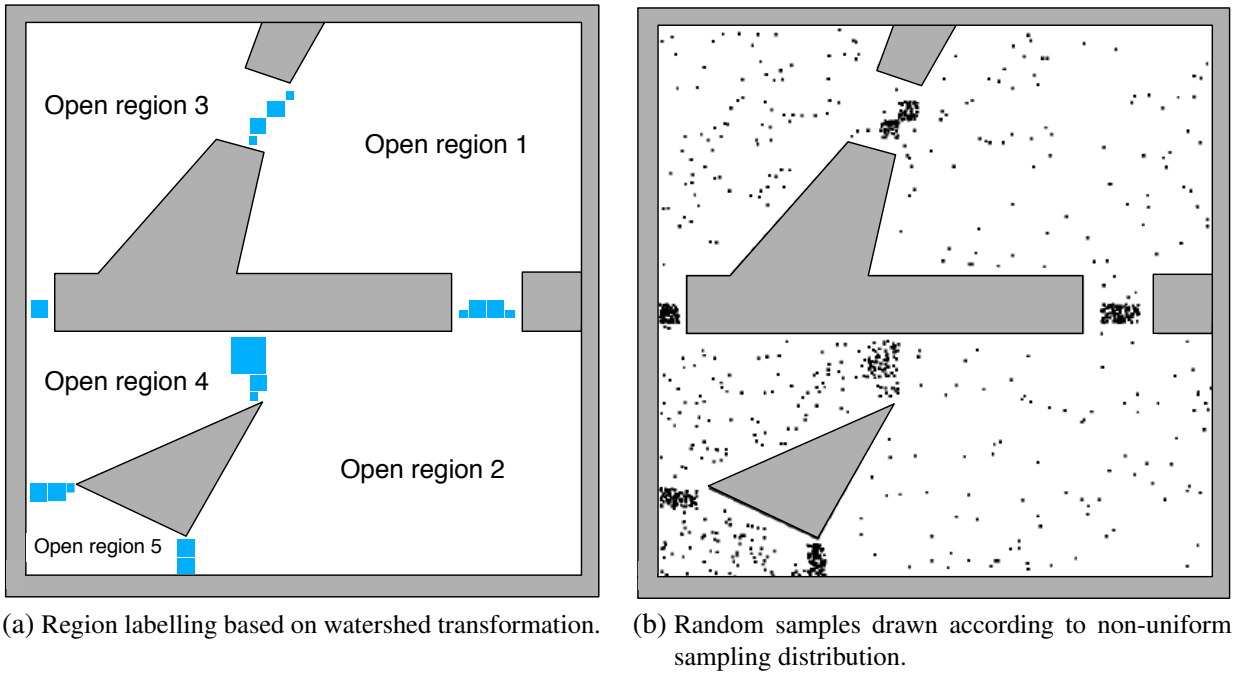


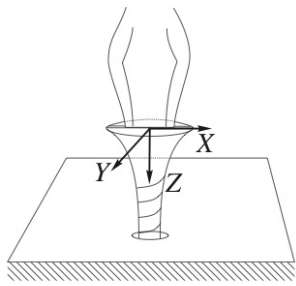
Figure 2.7.: Calculation of non-uniform sampling distribution using the watershed transformation, [107].

controlled directions, which form a basis for the twist vector space. The remaining $6 - n$ force controlled directions represent a basis for the wrench vector space. In each direction, a set-point or desired value has to be defined, which is used in the controller to adapt the robot motion in the contact situation. In each direction, different control strategies can be chosen by the user. The controller adapts the motion constantly to reach the set-points until a stop condition is fulfilled, e.g. when we pull a drawer a force indicates that the limit of the slider is reached.

The authors define different requirements to specify a TFF: geometric compatibility, causal compatibility and time-invariance. Geometric compatibility restricts that TF describes the contact situation completely, i.e. the user chooses the correct orthogonal basis and velocity controlled directions. The set-points and stop condition have to be compatible with the contact situation (causal compatibility) and the TF adapts automatically to changes in the geometry of the contact situation (time-invariance). Time-invariance is usually maintained by tracking the manipulated objects, e.g. the current orientation of a lever, and adapting the TF.

The TFF can be used to define a large variety of tasks, e.g. opening a door or turning a screw, see Figure 2.8. Counterexamples exist, which cannot be modeled as a TF [17].

Prats et al. [84] combine the TFF with a knowledge-based approach to grasping, in which grasps are defined based on predefined hand configuration (hand preshape), a hand pose relative to a grasp frame and a selection matrix to specify, in which dimensions of the grasp frame the hand pose is unconstrained, see Figure 2.9. In order to execute the task, force set-points are transformed into velocity set-points using a generalized spring. After transforming the velocities into the TCP coordinate frame, the Jacobian is applied to receive joint velocities to control the manipulator.



```

move compliantly {
  with task frame directions
  xt: velocity 0 mm/sec
  yt: force 0 N
  zt: force -f N
  axt: velocity 0 rad/sec
  ayt: force 0 Nmm
  azt: velocity  $\omega$  rad/sec
} until azt force < -m Nmm or time > t sec
    
```

Figure 2.8.: Task Frame specification to turn a screw with velocity controlled directions x_t , a_{x_t} , a_{z_t} and force controlled directions y_t , z_t , a_{y_t} , [17].

De Schutter et al. [28] extend the TFF to the more general constraint-based programming formalism. In their iTASC system, a general control scheme is introduced to define a compliant motion task with a set of feature coordinates, similar to the directions in TFF, and a set of equality constraints, similar to the set-points for different directions in the TFF. They derive a controller for the task and discuss how constraints can be weighted and a smooth transition between different subtasks, i.e. constraints, can be achieved. Constraint-based programming has been successfully applied to complex tasks like simultaneous manipulation of the same workpiece by two robots, e.g. one robot picks up an object, moves it to a conveyor belt and places it down while a second robot paints it with an airbrush.

Decre et al. [30] extend the iTASC framework with a more general objective function and support for inequality constraints with the disadvantage that a controller cannot be derived efficiently.

2.4. Programming by Demonstration

Constraint-based motion planning, see Section 2.1, and Constraint-based programming, see Section 2.3, are powerful techniques to execute manipulation tasks in a flexible and reliable way. The major drawback of these techniques is complexity. The representation of a manipulation task as the intersection of a number of constraints is challenging, e.g. the number of constraints is large and semantics of a constraint depend on the parameterization and referenced objects. The consequence is that the robot has to be programmed by a robotics expert with sufficient domain knowledge.

For simple tasks and less complex robot systems, e.g. a single industrial robot, the use of simpler programming techniques, e.g. Teach-In or Playback, has a lot of advantages. The user is usually a domain expert and knows how to move a tool to accomplish the task goal, e.g. a metal worker knows where to place a welding tool to get optimal welding points. In Teach-In [79, p. 339ff] the robot is controlled remotely using a Teach Box to specify key points in joint space to reach important positions, e.g. the welding points, or move around obstacles. The user chooses interpolation techniques, e.g. joint, linear or circular motions [79, p. 225-226], to generate a continuous robot trajectory based on the key points. If the robot supports zero-force control, the user is able to move the robot with his own hands, e.g. with a special lever attached to the TCP.

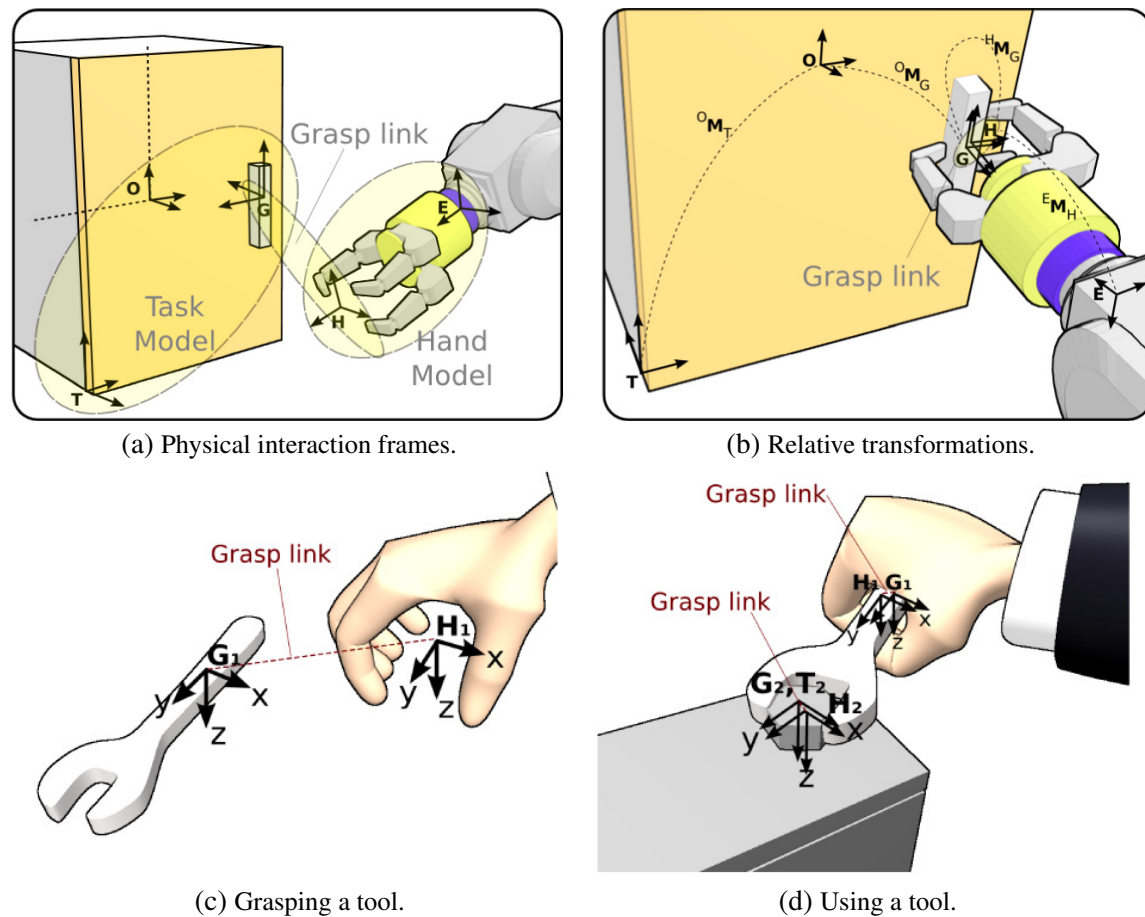


Figure 2.9.: Combination of Task Frame Formalism with knowledge-based approach to grasping, [84].

In this case, the robot trajectory is stored and can be executed directly on the robot system, which is called Playback [79, p. 347ff]. Both approaches make perfect use of the domain knowledge of the user and require limited experience with the robot. Programming is efficient, since humans are able to generate a predictive model of the robot behavior. The program is represented in joint space, which limits flexibility since the trajectory does not adapt to changes in robot posture, object poses or obstacles. In order to react to such changes, the robot requires precise models about itself, objects, obstacles and the environment and sensing capabilities. In the industrial context, reactive robot behavior is programmed using robot programming languages, e.g. KRL for KUKA robots and RAPID for ABB. In service robotics, the Robot Operation System (ROS) [85] or robot simulation environments, e.g. OpenRAVE [32], offer functionality to program robots using standard programming languages like C++ or Python. Textual programming, even assisted by simulation, requires expert knowledge and is inefficient.

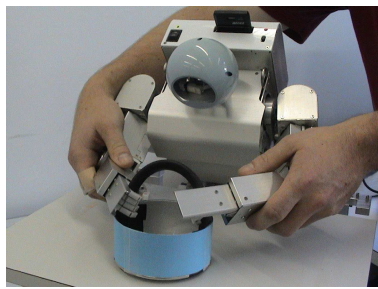
Complex manipulation tasks require a sophisticated task model, e.g. based on constraints, see Section 2.1, which is difficult to define manually. Since humans perform these tasks on a daily basis and are able to provide a sufficient number of examples, it is natural to investigate machine learning to enable a robot to learn such tasks based on the observation of a human teacher. Dillmann et al. [34] define this process as Programming by Demonstration. Billard et al. summarize

different approaches in [11]. A large body of work exists differing by the demonstration setup, e.g. using online robot sensors to demonstrate table tennis strokes [77] or dedicated sensor systems to observe human hand motions [81], by the task representation, e.g. probabilistic [21], dynamic [46], symbolic [78], and by the execution paradigm, e.g. closed-loop control [83] or policy execution [94]. We concentrate explanations on probabilistic, see Section 2.4.1, dynamic, see Section 2.4.2, and symbolic, see Section 2.4.3, PbD approaches. In most approaches, the set of learning features, e.g. to learn a 3D motion of the TCP relative to an object, is manually defined. Automatic approaches are required to enable a service robot to decide, which features in the huge amount of sensor data are relevant to the task and learn new tasks autonomously. In Section 2.4.4, we summarize different approaches, e.g. gazing or taking the human perspective, to reduce dimensionality in the large amount of sensor readings.

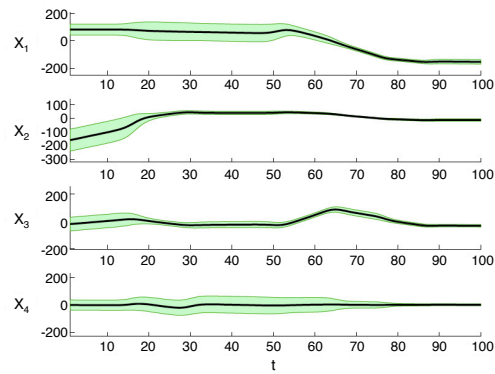
2.4.1. Probabilistic Imitation Learning

Probabilistic PbD focuses on learning skills. A skill is an efficient encoding of a set of similar, low-level manipulation motions that can be executed by the robot in closed-loop control. A human teacher demonstrates the manipulation motions by moving the robot arm [23], similar to Playback programming, by using a sensor arm with kinematics identical to the robot [82], by using haptic interfaces [92] or by moving objects tracked by a vision system [38]. Each demonstration represents a trajectory in a real-valued vector space. In Probabilistic PbD, a learned skill encodes a number of demonstrations as a probability density function (PDF). In order to execute the skill in a new environment, a robot trajectory is generated based on the PDF, which minimizes a metric of imitation. Due to this nomenclature the approaches are referred to as Imitation Learning.

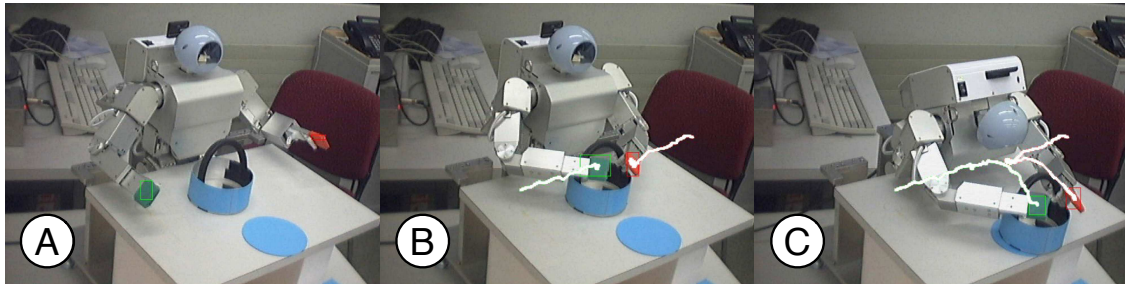
Billard et al. [12] investigate the use of Hidden Markov Models (HMM) to represent skills. The goal is to deduce a controller, which generates a trajectory minimizing a specific cost function, the so-called metric of imitation. The metric of imitation is a linear combination of cost functions for specific constraints, e.g. to reproduce the relative hand-object motion. The idea of a metric of imitation was elaborated further by Calinon et al. [23]. The learning data consists of time, joint angles of the robot, hand poses relative to robot, hand poses relative to object and a discrete activity for each hand. Each constraint forms a separate vector space, in which a PDF is learned using the Expectation-Maximization (EM) algorithm. Discrete activities are encoded as a Bernoulli Mixture Model (BMM), the other constraints as Gaussian Mixture Models (GMM), see Figure 2.10. Since time is known during execution, Calinon et al. calculate the PDF conditioned on time using Gaussian Mixture Regression (GMR) resulting in the final skill representation. Based on this representation, they deduce an optimal controller, which enables the robot to generate a trajectory minimizing the metric of imitation. In this case, the metric of imitation is the weighted Euclidean distance to the mean of the data, where the weight matrix is set to the inverse covariance matrix of the data. In other words, the generated trajectory has to be reproduced the more precise the smaller the variance in a specific dimension. In [21], Calinon et al. deduce a controller enabling the robot



(a) Observation of human demonstrations.



(b) Learned GMM (left) and deduced skill representation using GMR (right).



(c) Reproduced robot motion (white) on HOAP-2.

Figure 2.10.: Learning a Pick&Place-task using Gaussian Mixture Models, [23].

to mix joint space and Cartesian constraints using the manipulator Jacobian to project Cartesian constraints into joint space. Since skills are represented on the motion level, high variance in the environment and start configuration may require multiple encodings of the same skill. In [22], the framework is extended to automatically choose between motion alternatives, i.e. different encodings of the same skill. Rozo et al. [87] follow the same goal but train a HMM to reflect the temporal information in the data. In the reproduction phase, the GMR controller is extended to consider not only the current configuration but also the previously measured configurations to choose between motion alternatives, which is done implicitly by using the transition probabilities in the HMM.

Eppner et al. [38] generalize the approach by using a Dynamic Bayesian Networks (DBN) to model the imitation learning process. Similar to the previous approach, they consider joint constraints and constraints, which restrict the motion of an object relative to the TCP. Both types of constraints are represented as Gaussian distributions. In the learning phase, Eppner et al. apply the Parzen window kernel to estimate the parameters of each Gaussian based on the limited number of training examples. In the reproduction phase, see Figure 2.11, the displacement vector for each constraints is calculated and Cartesian constraints are projected into joint space using the manipulator Jacobian. They combine all constraints using the projected covariance matrices resulting in a Gaussian distribution describing the displacement from the current configuration to the mean of all learned constraints. Eppner et al. define a simple controller by setting the goal configuration to the sum of the current configuration and the mean of the combined Gaussian distribution. Small obsta-

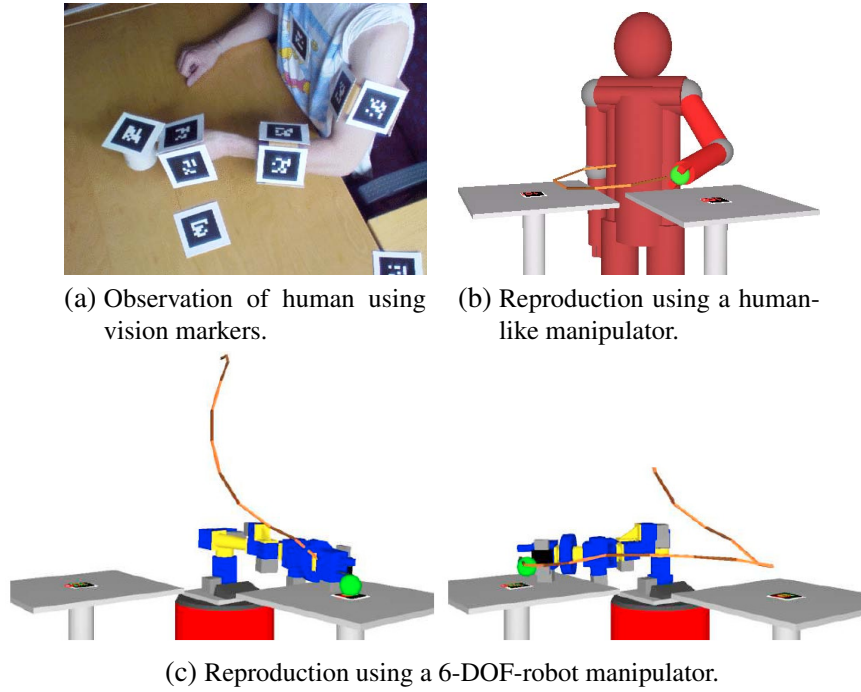


Figure 2.11.: Learning a Pick&Place-task using Gaussian constraints, [38].

cles can be avoided locally by adding a repellent force in the controller formula. Eppner et al. also discuss how large or difficult obstacles, e.g. U-shaped, can be considered by using planning-based approaches but no experimental validation is given.

2.4.2. Dynamic Imitation Learning

A different approach to encode a set of low-level trajectories are (second-order) differential equations, which were first introduced by Ijspeert et al. [46] to describe the motion of a humanoid robot. Schaal et al. [91] extended the approach to the Dynamic Movement Primitives (DMP) framework. A DMP is a motion primitive consisting of a set of nonlinear differential equations. Schaal et al. define multiple variants of DMPs. Following the notation in Pastor et al. [82], an important variant is defined by the following differential equations for a single degree of freedom:

$$\tau \dot{v} = K(g - x) - Dv + (g - x_0)f \quad (2.7)$$

$$\tau \dot{x} = v \quad (2.8)$$

with position x , velocity v , start position x_0 , goal position g , temporal scaling τ , spring constant K and damping D . The non-linear function f is

$$f(s) = \frac{\sum w_i \phi_i(s) s}{\sum \phi_i(s)} \quad (2.9)$$

with phase variable s , weights w_i , Gaussian basis functions

$$\phi_i(s) = \exp(-h_i(s - c_i)^2) \quad (2.10)$$

with center c_i , and width h_i . The phase variable s ranges from 0 to 1 and is defined by $\tau\dot{s} = -\alpha s$ with constant α . Except w_i all variables are predefined. In order to encode a set of trajectories $x(t)$ the parameters w_i have to be calculated. First, the $s(t)$ and the derivatives $v(t)$ and $\dot{v}(t)$ are calculated (using numerical differentiation if necessary). Second, Pastor et al. calculate goal values of $f(s)$ using Equation 2.8:

$$f_{target}(s(t)) = \frac{-K(g - x(t)) + Dv(t) + \tau\dot{v}(t)}{g - x_0} \quad (2.11)$$

They calculate the parameters w_i with linear regression to minimize $\sum_t (f_{target}(s(t)) - f(s(t)))$. In a new situation, i.e. with different g and x_0 , a trajectory is calculated by starting with $s = 1$ and integrating the differential equations. For robots with multiple degrees of freedom, a set of differential equations, i.e. a set of weights, is learned in each dimension separately.

A smooth transition between subsequent DMPs is obtained by using the position and velocity at the end of the first DMP as the start values of the second. Obstacles can be avoided locally by adding a velocity term, which pushes the TCP away from an obstacle in the workspace.

The DMP variant above has its limitations due to the weighting of f with $g - x_0$. If $g = x_0$, e.g. when learning periodic movements returning to the start configuration, no motion will be generated. If $g - x_0$ is close to 0, high accelerations may damage the robot. If $g - x_0$ has a different sign than during learning, the motion will be mirrored. Pastor et al. [82] define a new variant to overcome these limitations:

$$\tau\dot{v} = K(g - x) - Dv + K(g - x_0)s + Kf \quad (2.12)$$

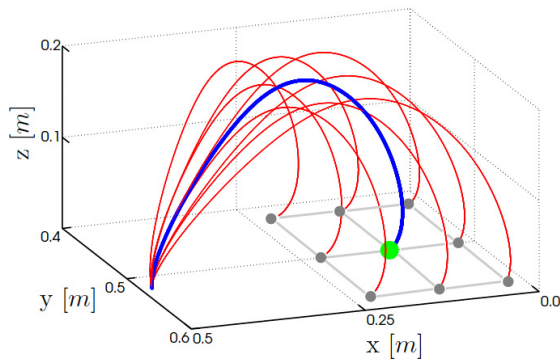
$$\tau\dot{x} = v \quad (2.13)$$

Similar to the controller defined in the previous section, trajectories adapt to different start and goal configurations of the robot, see Figure 2.12a.

In [83], Pastor et al. learned DMPs to perform a pool stroke and to manipulate a box with two chopsticks. The learned DMPs could not reproduce the task since the dynamics could not be reproduced accurately. They defined a cost function manually, which defines the goal of the task, and ran a Reinforcement Learning algorithm PI^2 to modify the DMP parameters until the task was reproduced accurately.

DMPs were learned successfully for different tasks, e.g. tennis strokes [46], ball on string [63], darts [63] or pan cake flipping [64].

The goal of probabilistic and dynamic PbD is to encode a number of low-level trajectories in an efficient way to reproduce the way how the human moved the robot. The assumption is that a



(a) Adaptation to different goal configurations.



(b) Human demonstrations with Sarcos slave arm.



(c) Reproduction using a 6-DOF Sarcos master arm with identical kinematics.

Figure 2.12.: Learning the pour-in task using DMPs, [82].

similar motion achieves the same task goal, which limits generalization to minor changes in the positions and orientations of relevant objects.

2.4.3. Symbolic PbD

In contrast to probabilistic and dynamic PbD approaches, symbolic PbD tries to extract the abstract goals of a task using a symbolic description, e.g. based on predicates, and abstract actions necessary to achieve the goals.

In early work by Dillmann et al. [33] low-level skills are learned based on multiple demonstrations with the focus on segmentation, which is highly relevant to symbolic PbD.

In the simplest case, primitive actions and their semantics are predefined. Ekvall et al. [37] use a graphical interface to define a behavior by combining a set of primitive actions. Since the primitive actions are predefined, pre- and post-conditions are known and only valid combinations are built. They combine the approach with low-level PbD by letting the teacher demonstrate a number of trajectories for each primitive action to cope with the symbol grounding problem, i.e. how to generate a robot motion, which reflects a symbol like move until contact in the given context. The consequence is that the same limitations apply as for probabilistic and dynamic PbD. Mühlig et al. [75] follow a similar approach and combine probabilistic PbD to learn skills and symbolic PbD to learn abstract sequences of the learned, primitive skills.

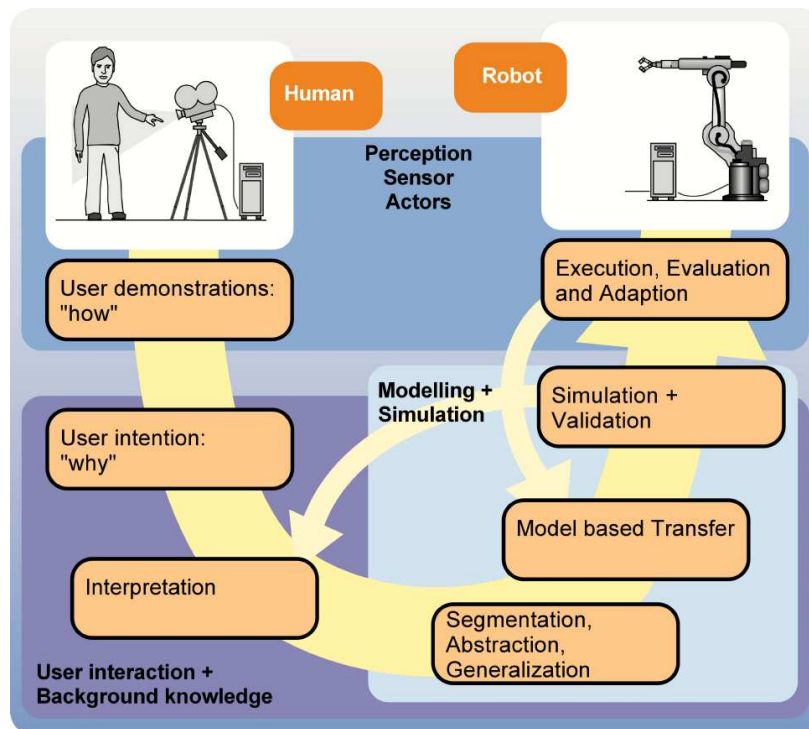


Figure 2.13.: Formalization of PbD process by Dillmann et al. [34].

Dillmann et al. [34] classify different PbD approaches and give an overview of the mapping process between human and robot skills, see Figure 2.13. In their PbD system, the human teacher demonstrates tasks naturally with his own hands in a sensory environment. Data gloves, magnetic field-based motion trackers and computer vision is used to track the human hand and finger tips in simple tasks, e.g. without occlusions. They use grasp classification and trajectory analysis to map a human motion to a sequence of primitive actions, e.g. approach, grasp or linear motion, which are called elementary operators (EOP). Dillmann et al. do not consider the symbol grounding problem but assume that each EOP can be executed directly.

Zöllner et al. [115] extended the approach and use predefined hierarchies of EOPs, so called Macro Operators, to deduce a hierarchical representation of the task. The temporal dependencies of both human hands during bimanual manipulation tasks are represented as a petri net. A learned Macro Operator to unscrew a jam jar was executed on the humanoid robot ARMAR but it is not described how the symbol grounding problem was solved.

The approach was further refined by Pardowitz et al. [81] who introduce Task Precedence Graphs (TPG), which define a partial order on a set of EOPs. They apply machine learning to deduce a TPG based on multiple sequences of EOPs. Based on the TPG, a robot decides which EOPs can be executed in parallel potentially decreasing execution time.

In the previous approaches, the human teacher clarifies ambiguities and resolves false classifications manually. Rybski et al. [89] add spoken dialog to the PbD process to resolve these problems in a more natural way.

Veeraraghavan et al. [109] also use speech and visual perception to allow more efficient programming. Tasks are represented as programs, where the building blocks are primitive actions, loops, when-conditions and repeat-until-loops. They analyze demonstrated sequences of primitive actions to detect loops, when-conditions and repeat-until-loops.

Schmidt-Rohr [93] developed a sophisticated PbD system to learn decision making policies based on human observation and interaction with the human. One or two human teachers are observed with the robot sensors considering different skill domains, e.g. mobility, object manipulation and human-robot interaction, see Figure 2.14. The sensor data is automatically segmented and clustered to deduce a discrete set of states. The transition between states is automatically mapped to a discrete set of actions. In addition to predefined actions, e.g. speak, Schmidt-Rohr uses actions, which were learned in this thesis, e.g. a planning model to grasp and lift a chair, see Figure 2.14b. In order to map the human motion, which is detected less accurately, to a learned planning model, a classifier for the left and right wrist motion is learned based on the training data of the planning model. Based on (smoothed) sequences of states and actions, Schmidt-Rohr calculates an initial Partially Observable Markov Decision Problem (POMDP) model. The model is analyzed and the robot requests additional demonstrations to generalize the model with additional state transitions. The POMDP model is refined using background knowledge to include robot specific parameters, e.g. action costs. Schmidt-Rohr uses geometric analysis to refine transition probabilities for mobility, e.g. due to collisions and path deviations. Finally, he uses dynamics simulation to refine transition probabilities for object manipulation by considering noise in the localization of errors, see Figure 2.14d. Based on the POMDP model, a policy is computed and executed autonomously on the service robot Albert II.

2.4.4. Feature Selection

In Probabilistic and Dynamic PbD approaches, (demonstrated) motions are encoded in an efficient way by learning probability density functions or learning parameters of a set of differential equations. The basis is a representation of the motion as sequence of real-valued vectors. An example is given in [23]. The chess knight move is represented by time, joint values, 6D hand pose relative to robot, 6D hand pose relative to object and an activity identifier. The motion is independent of the pose of the table, the individual poses of each chess board, among others. In [83], Pastor et al. defined a pool stroke, see Figure 2.15a based on the “translational offset from the right gripper to the bridge, the roll, pitch, and yaw angles of the cue around the bridge, and the redundant degree-of-freedom in the arm” [83, page 3]. In these examples, the machine learning expert selected the features, which have to be considered in the learning process, manually. The disadvantage is that expert knowledge in the learning algorithm is required, e.g. Pastor et al. pointed out that the cost function, which represents the goal of the task, requires “careful tuning” [83, page 3].

In realistic learning settings, a large number of potential features exist, e.g. coordinate frames defined in each recognized object, coordinate frames in the robot, joint values of the robot as well

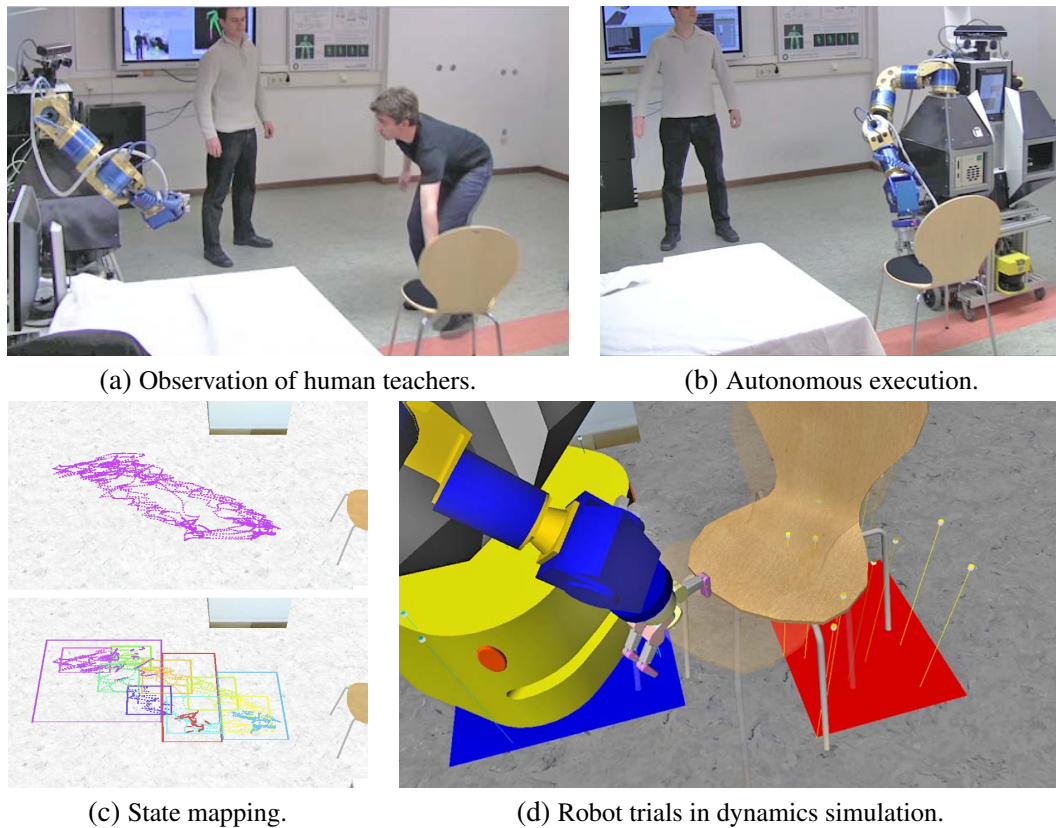


Figure 2.14.: PbD system by Schmidt-Rohr et al. [93].

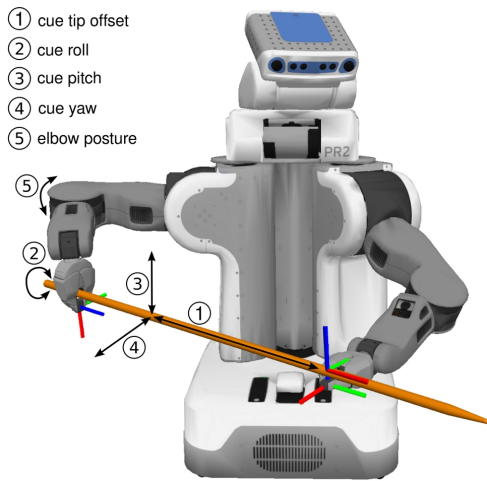
as task-specific features, e.g. the water level in a pot. Different approaches exist to transform and reduce dimensionality of the feature space or to select a subset of features.

Principal Component Analysis (PCA) is a standard tool in mathematics. It calculates the d -dimensional affine subspace of the feature space, which minimizes the mean-squared-error between a feature vector and its representation in the subspace. The calculation is simple: compute stochastic mean μ and covariance matrix Σ of the data points, calculate eigenvalues and eigenvectors of the covariance matrix, sort the eigenvectors by decreasing eigenvalue and use the first d eigenvectors as the columns of a transformation matrix A . A data point x is projected in the d -dimensional subspace by applying the transpose of the transformation matrix: $(\mu_1, \dots, \mu_d)^T + A^T(x - \mu)$. Usually, the dimension of the subspace is not important and the goal is to find the subspace with smallest dimension, which preserves $1 - \epsilon$, e.g. 98%, of the energy of the data. The energy of the data is the sum of all eigenvalues. Therefore, we calculate d as the number of sorted eigenvalues, which sum up to more than $1 - \epsilon$.

Calinon et al. [23] apply PCA to reduce dimensionality in a small, predefined feature space. For the chess move, dimensionality was reduced from 23 to 14. In [19], they showed that Independent Component Analysis (ICA) has a slight advantage but the decorrelation of PCA is sufficient.

In Figure 2.16, PCA was applied to the finger and hand motion to open a bottle with $\epsilon = 0.02$. The result shows that the hand variance dominates the finger variances although the latter is more important to the task.

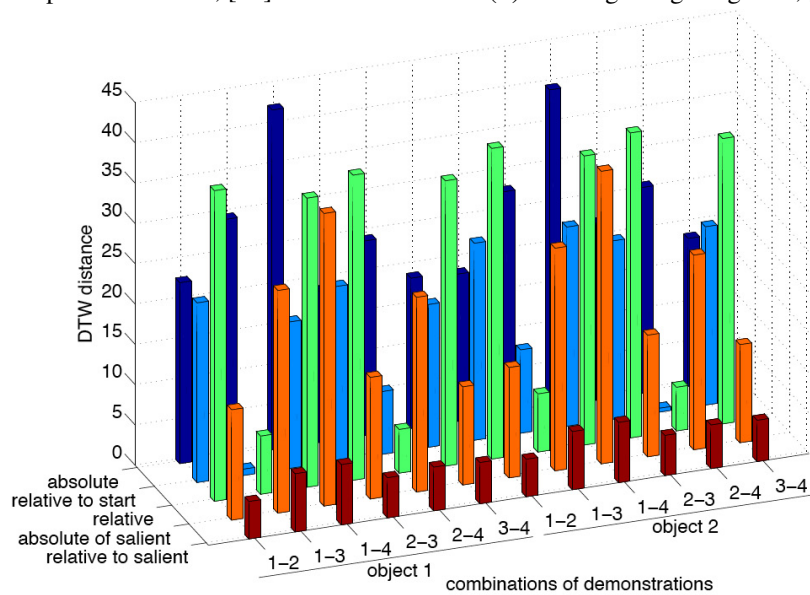
2. Theoretic Background and Related Work



(a) Features in pool stroke task, [83].



(b) Pointing and gazing cues, [20].



(c) Variance-based criterion to select features, [76].

Figure 2.15.: Features and feature selection in PbD.

In [20], a different approach is followed. Pointing and gazing cues are used to identify relevant objects, see Figure 2.15b. The approach was used in a game scenario. Calinon et al. performed a user study to analyze the differences between a simple keyboard interface and using pointing and gazing cues to select the relevant object. The results show that the latter has a higher score for attentive, friendly, enjoyable and natural. Appropriate, efficient and easy show similar scores.

Breazeal et al. [15] follow a similar approach to learn simple tasks. Since demonstrations can be ambiguous, e.g. it is unclear, which object the human points to, or flawed, the robot interacts with the human to deduce the goals. The process is called perspective taking. The approach was tested in a simple scenario with three on/off-buttons, which can be activated in different patterns. Some buttons are occluded from the view of the robot resulting in different schemas for the human and robot, e.g. the intention of the human is to *press any buttons ON* but the hypothesis of the

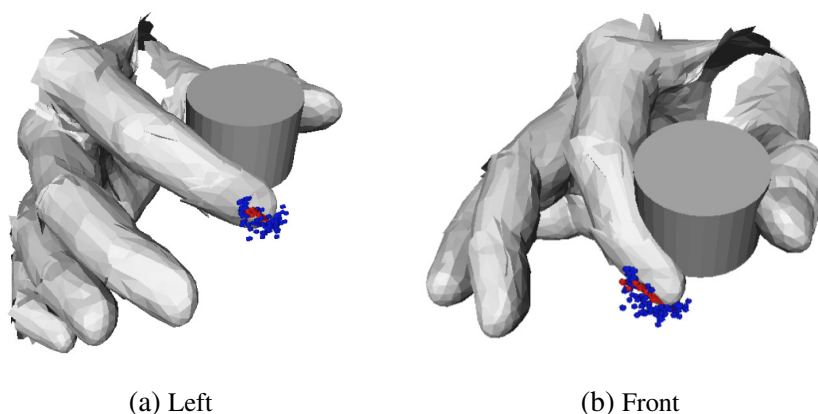


Figure 2.16.: Principal Component Analysis was applied to the finger (blue) and hand motion to open a bottle. 98% of the energy of the data was preserved, but information about the finger motion was lost (red).

robot is *press the buttons we can both see ON*. The robot uses social cues, e.g. pointing and facial expressions, to resolve ambiguities and determine a single schema for human and robot.

Mühlig et al. [76] investigate a more direct representation of the feature space, which they call task space pool. The task space pool contains predefined features, e.g. the relative position of the TCP to an object. They provide different criteria to weight the features: a variance-based, see Figure 2.15c, an attention-based and a kinematic criteria. The goal is to select a number of features based on the weights but only a single example was evaluated and the integration into a learning framework is not described.

2.5. Analysis

In Section 2.5.1, the requirements for autonomous learning and flexible execution of manipulation tasks are defined, which serve as a basis to analyze and compare the relevant work in the state-of-the-art. The evaluation of the presented work is described in Section 2.5.2 by grouping relevant work according to the sections and assigning a representative value according to the defined features. Based on the evaluation, the combination of planning-based approaches with machine learning techniques is motivated in Section 2.5.3 to overcome the limitations of the state-of-the-art.

2.5.1. Requirements

The requirements for autonomous learning and flexible execution of manipulation tasks are defined by four distinct properties:

1. Explicit representation of continuous subspaces of the configuration space describing relevant manipulation aspects
2. Autonomous extraction of relevant learning features
3. Flexibility to varying object poses, obstacles and start configurations

4. Efficient execution

Requirement 1 ensures that the representation of manipulation tasks captures ranges of values for important manipulation aspects, e.g. object positions and orientations, joint values, force and torque measurements. This property is necessary to describe goals and constraints of the manipulation task in a way suitable for state-of-the-art planning and control algorithms. In order to monitor the execution of manipulation tasks, to react to failures and to allow autonomous planning, the representation has to allow to test if a given configuration is consistent with the representation or not. The approximative distance of an inconsistent configuration to the nearest consistent configuration has to be computable in order to efficiently search the continuous configuration space.

If *Requirement 2* is satisfied the robot will be capable to extract and consider only learning features, which are relevant to the given manipulation task. Since the set of learning features has to be existent in the execution environment, e.g. relevant object properties, its size has great influence on the flexibility of the learned manipulation knowledge.

Requirement 3 describes the flexibility of the execution algorithm in regard to different object poses, obstacles. e.g. narrow passages, and start configurations.

Requirement 4 ensures that the execution of the manipulation task is efficient. Service robots possess a large number of degrees-of-freedom resulting in a potentially high complexity and computation time. This is a weaker requirement. Moore's Law indicates that computation time will decrease due to rising computation power. In the case of foot-step planning for bipeds, this observation is imminent. In 2001, computation of a foot-step sequence to walk over a set of static obstacles took 12 seconds [57]. Four years later, foot-step sequences could be computed with similar approaches online [25], i.e. for dynamic obstacles.

2.5.2. Evaluation

In this subsection, we discuss limitations of the state-of-the-art and rate related work according to the defined requirements one to four. Table 2.1 shows a summary of the ratings for selected related work.

In Probabilistic PbD, manipulation tasks are represented as probability density functions, e.g. Gaussian Mixture Models. The representation is suitable for manipulation tasks, since constraints and goals can be efficiently encoded. In the described learning systems only trajectories are encoded and no goals are extracted. The motion itself is reproduced, which limits generalization to small changes in object poses. Goal-directed planning is not supported. In the described learning systems, only simple techniques to reduce dimensionality are used, e.g. PCA. The result of PCA is a linear transformation of the input features, which mixes (potentially) all features. In order to reproduce the manipulation task in a different setting, all features have to be present. If a feature refers to an object, e.g. an object-relative coordinate frame, the object has to be present, even if it is irrelevant to the task. In order to learn manipulation tasks automatically in scenes with multiple objects and multiple object-relative coordinate frames, more advanced feature selection techniques

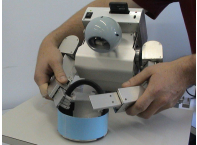
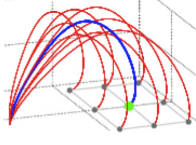

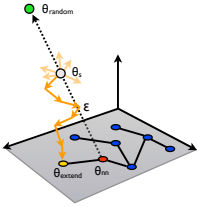
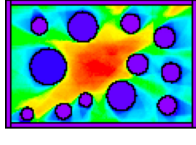
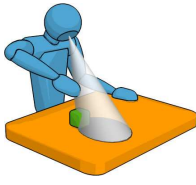
Requirement	Probabilistic PbD: Calinon et al. [21], Eppner et al. [38], Rozo et al. [87]	Dynamic PbD: Pastor et al. [82], Schaal et al. [91], Ijspeert et al. [46]	Symbolic PbD: Schmidt-Rohr [93], Ekvall et al. [37], Dillmann et al. [34]	Motion Planning: Berenson et al. [10], Stilman et al. [101]	Search Heuristics: Zucker et al. [116], Brock et al. [16]	Feature selection: Mühlig et al. [76], Breazeal et al. [15]
	 - [23]	 - [82]	 - [93]	 - [101]	 - [116]	 - [11]
Explicit representation of continuous subspaces of the configuration space describing relevant manipulation aspects	■	■	■	■	■	■
Autonomous extraction of relevant learning features	■	■	■	□	□	■
Flexibility to varying object poses, obstacles and start configurations	■	■	■	■	■	□
Efficient execution	■	■	■	■	■	□

Table 2.1.: Rating of selected related work using requirements 1 to 4. Fully fulfilled (green), fulfilled with restrictions (yellow), not fulfilled (red) and not applicable (white).

are required. A motion is reproduced efficiently using a velocity-based controller with local obstacle avoidance. Flexibility is limited since the controller cannot adapt to large changes in the environment, e.g. which require global obstacle avoidance. Additionally, learned constraints can be violated arbitrarily, e.g. a pour-in task can be executed starting with cup and bottle upside down. For autonomous robots, the latter represents a severe limitation since the execution result cannot be validated. More advanced execution techniques are required.

Dynamic PbD shares the same limitations as Probabilistic PbD in the requirements 2, 3 and 4. The representation is less suited for manipulated tasks since goals and constraints cannot be represented explicitly. Goal-directed planning and validation of the execution is not possible.

In both PbD approaches a set of features has to be defined, which serves as a basis to describe the manipulation task. For complex manipulation tasks, a large number of potentially relevant features exists. In order to learn manipulation tasks automatically a subset of these features has to be selected automatically. Mühlig et al. [76] introduced different criteria to support a machine learning expert to select a set of features based on the analysis of human demonstrations. Sadly, the approach was not integrated into a fully automatic learning system. Pointing and gazing were analyzed in different settings to deduce, which object is currently in the focus of the teacher. The scalability to manipulation tasks, where different objects or object-dependent coordinate frames may be important in different steps of the task, was not analyzed.

The main advantage of Symbolic PbD is generalization. A manipulation task is represented based on a set of primitive actions, e.g. move, approach, grasp, which abstract from the details of the real-world. The result is that tasks consisting of multiple subtasks or a larger number of primitive actions can be learned efficiently compared to probabilistic or dynamic PbD. The assumption is that the semantics of all actions are known and reflected by their implementation, e.g. a grasp action has the same result on a robot with a single arm and jaw gripper and on a humanoid-like robot. This refers to the symbol grounding problem, which is currently unsolved. A second disadvantage is that all necessary manipulation skills have to be defined beforehand but no taxonomy of manipulation tasks exists.

Constraint-based motion planning generates a robot motion in a goal-directed way by searching in the full configuration space. It offers the highest flexibility to execute a manipulation task in a new environment with different object poses, obstacles and a restricted workspace. Manipulation tasks are represented by a number of constraints, which restrict relative object motion and joint values, and a number of goals, which are represented consistently as constraints. Based on this representation, the planning and execution can be validated, which is an important precondition for autonomous robots. The main limitations are availability of precise object and robot models and high planning times, especially for coordinated finger-arm motions.

State-of-the-art search heuristics, see Section 2.2.1 and 2.2.2, produce a significant speed-up of the planning process. Learning a non-uniform sampling distribution on a discretization of the workspace does not scale to environments with varying objects and robots with workspace limitations. Deriving an explicit representation of the free space reduces the number of collision queries

significantly. In general, there will be situations, in which search heuristics fail or have arbitrarily long execution times. Incremental learning of new search heuristics, which cover situations, in which the previously learned search heuristics failed, was not considered.

2.5.3. Conclusion

In this chapter, we discussed related work in the state-of-the-art of constraint-based motion planning, search heuristics, constraint-based execution and PbD. In order to execute a manipulation task in the human environment a service robot has to be flexible to adapt to different objects, object poses and obstacles. Goal-directed motion planning has proven more flexible than the reproduction of human trajectories based on a set of encoded human demonstrations. The assumption is that an explicit representation of goals and constraints exist. Current approaches in constraint-based motion planning, which originate from constraint-based programming, are limited. Different modalities, e.g. forces, positions and orientations, as well as more complex constraint and goal representations, e.g. Gaussian Mixtures, which are used successfully in the context of Probabilistic PbD, are required. In Chapter 3, we extend constraint-based representations of manipulation tasks to overcome these limitations and adapt state-of-the-art constraint-based motion planners to plan complex manipulation tasks, e.g. using dynamics simulation to predict object motion during fingertip manipulation. Manual definition of manipulation tasks is time-demanding, error-prone and requires expert knowledge. In the PbD paradigm a human teacher demonstrates a manipulation task in a natural way and machine learning is applied to generate the representation of the manipulation task. A severe limitation is the manual definition of the set of features, i.e. what is important to the task, by the human teacher since it requires expert knowledge. Automatic selection of features is currently limited to the selection of objects using pointing or gazing, which is insufficient for (bimanual) manipulation tasks with multiple steps, e.g. relevant objects or coordinate frames change during manipulation. In Chapter 4, we present a PbD system to learn manipulation tasks based on a set of explicit human demonstrations. Features are selected automatically using either two sets of human demonstrations or an automatic optimization process with the goal to find a maximal subset of features, which admits a successful solution to a set of test problems. The main limitation of constraint-based motion planning is high planning time. State-of-the-art search heuristics do not scale well to environments with varying objects and obstacles. We introduce a novel approach to learn search heuristics incrementally in Chapter 4.

3. Modelling and Planning of Manipulation Tasks

Service robots have to be flexible to execute manipulation tasks with different objects in varying environments, where multiple obstacles restrict the workspace. Planning, i.e. “the task of coming up with a sequence of actions that will achieve a goal” [88, p. 375], enables the robot to adapt its motion to the environment in a flexible and goal-directed way. The basis is a sophisticated model of the manipulation task, which captures all qualitative and quantitative aspects that are required to plan and monitor the execution. In Section 3.1, we introduce constraints, e.g. restricting the relative position of two objects, to model different aspects of a task. Constraints represent the basis to define planning models for manipulation tasks. In Section 3.2, we define the planning model, so called *strategy graph*, to model multiple goals as well as constraints of a task and their temporal relationships. We discuss the connection to constraint satisfaction problem theory and define a planning algorithm to plan with multiple, parallel goals and constraints. Planning models adapt automatically to different object poses. In order to map a planning model to a new object, e.g. with a different 3D model, the set of constraints has to be transformed, which will be discussed in Section 3.3. Finally, we focus on planning of *strategy graphs* in real-world environments. Since planning with parallel goals and constraints is extremely hard, we adapt a state-of-the-art motion planner, see Section 3.4, to plan *strategy graphs* with a linear sequence of goals and constraints in an efficient way. Figure 3.1 shows an overview of the modeling aspects of the developed PbD process, which are discussed in this chapter.

3.1. Constraints

Autonomous service robots require a sophisticated task model in order to plan the execution in a goal-directed way and monitor the execution online. One of the key problems is to deduce the relevant aspects of a task, which have to be actively enforced during planning and validated online during the execution. For example, when pressing a button with a robot finger, the coordinate frame in the fingertip has to stay in a region above the button, a contact between the 3D model of the fingertip and the button is established and a force normal to the button surface exists, see Figure 3.2. In contrast, the relative position of the fingertip to different objects as well as the wrist pose are irrelevant. By defining only relevant aspects of a task, the planner can choose irrelevant aspects, e.g. the wrist pose, to fulfill the relevant aspects, which greatly increases flexibility. In order to learn a task model autonomously, a set of basic aspects of a task has to be defined, from which a set of relevant aspects can be deduced.

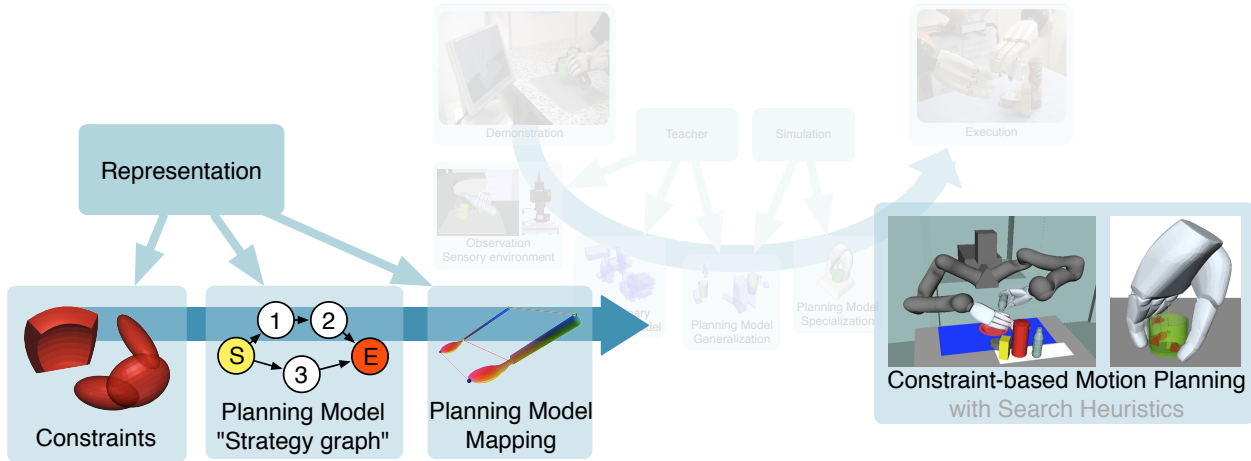


Figure 3.1.: Overview of modeling aspects of the developed PbD process in Figure 1.4. The modeling aspects are enlarged. Learning aspects are transparent and will be discussed in the next chapter.

In general, dexterous bimanual manipulation tasks require the coordinated motion of fingers and arms to apply forces and produce certain object movement and object state change. The representation of manipulation tasks has to capture finger-, arm-, object-, joint-motion as well as forces and torques between two bodies, e.g. the robot finger and an object. Multiple subgoals and their temporal ordering have to be considered, e.g. to conclude the previous example in Figure 3.2: the hand-arm system is moved to a configuration, in which the fingertip is above the button, a contact between the finger and button is established and the button is pressed down slightly.

In constraint-based programming, see Section 2.3, the main idea is to describe manipulation tasks based on the instantaneous motion between two coordinate frames, which is a 6D space described by a 3D translation and 3D rotation between both coordinate frames. The instantaneous motion is restricted by limiting the (relative) velocities $(v_x, v_y, v_z, v_\alpha, v_\beta, v_\gamma)$ and wrench $(f_x, f_y, f_z, f_\alpha, f_\beta, f_\gamma)$, i.e. force and torque, to have a certain predefined value. For example, in the task to open a drawer, which moves in $-z$ direction, velocity would be restricted to $(0, 0, -10\text{mm/s}, 0, 0, 0)$ and wrench to $(0, 0, -5\text{N}, 0, 0, 0)$. Based on this representation, an optimal velocity-based controller can be deduced by solving an optimization problem. In [84], this concept is applied to simple manipulation tasks. Since ranges for the finger-, arm- and object-pose as well as contact-points are not represented, manipulation tasks, which require coordinated hand-arm- or multi-finger-motion as well as induced object motion, e.g. how deep a button should be pressed at least, cannot be represented. In [30] the iTaSC framework is extended to represent inequality constraints and therefore (hyper-)cubical ranges for restricted values but with the limitation that no optimal controller can be deduced directly. In the human environment, objects with complex geometry, e.g. a teapot, and arbitrary obstacles have to be considered. Additionally, articulated robot hands introduce closed-loop kinematics chains during manipulation, e.g. while opening a bottle. Since the object geometry as well as closed-loop kinematics are not considered in the deduced control algorithms, constraint-based programming cannot be applied directly to the defined problem.

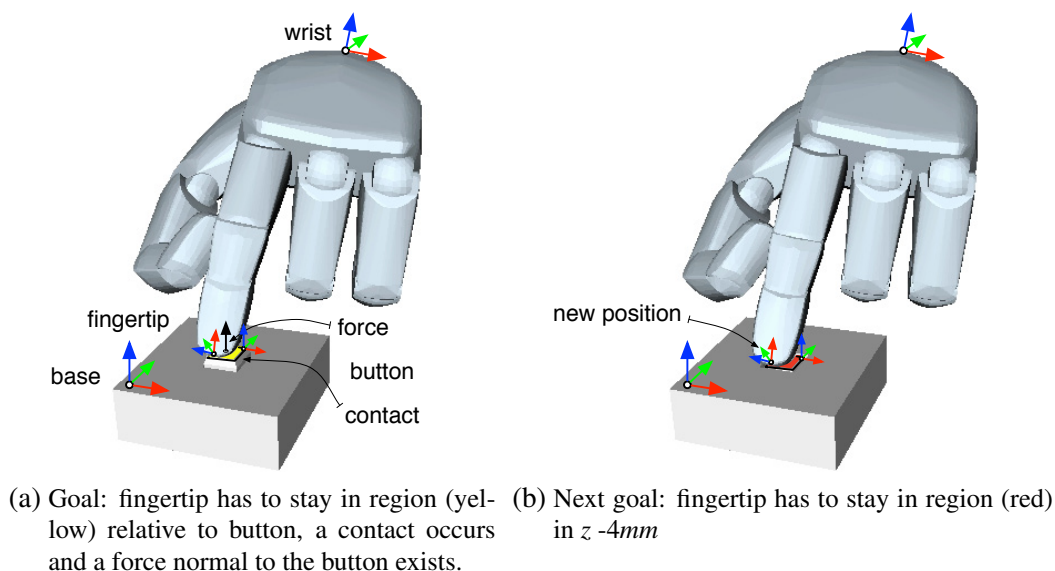


Figure 3.2.: Constraint example: pressing a button with a robot finger.

Instead, global search in the full configuration space has to be applied to consider the geometry, kinematics and dynamics of the robot, objects and obstacles. This topic is heavily researched in the field of constraint-based motion planning. In [10], the constraint representation from constraint-based programming was adapted so that the relative motion of one coordinate frame relative to a second coordinate frame has to stay in a hypercube, which translates to inequality constraints on the translation x, y, z and rotation α, β, γ . The constraints are used to describe the workspace-goals in a constraint-based motion planning process, which represents the basis of the representation of manipulation tasks in this thesis. The representation is still insufficient to cover all relevant aspects of bimanual, dexterous manipulation tasks.

In this thesis, the constraint representation from [10] is extended to consider arbitrary region types by formulating the requirements by the motion planner on the constraint representation, i.e. drawing samples and calculating the distance to the constraint manifold. The set of constraints is extended to forces and torques, which is consistent with the Task Frame concept. Additionally, configuration constraints are defined, which restrict the joint values of the robot directly, e.g. to be able to represent inactivity of one arm during loosely coupled bimanual manipulation. Contact constraints will be introduced to enforce a contact between two 3D bodies, which is necessary to describe sliding motions and environment conditions, e.g. due to gravitation, a cup will be in contact with a table while being pushed. Finally, temporal constraints are defined to represent the ordering of subgoals for complex manipulation tasks with multiple subgoals.

3.1.1. Constraint Definition

A service robot with multiple manipulators, e.g. independently movable fingers and arms, as well as objects, on which different forces like gravity act, represents an input-output system, see Figure

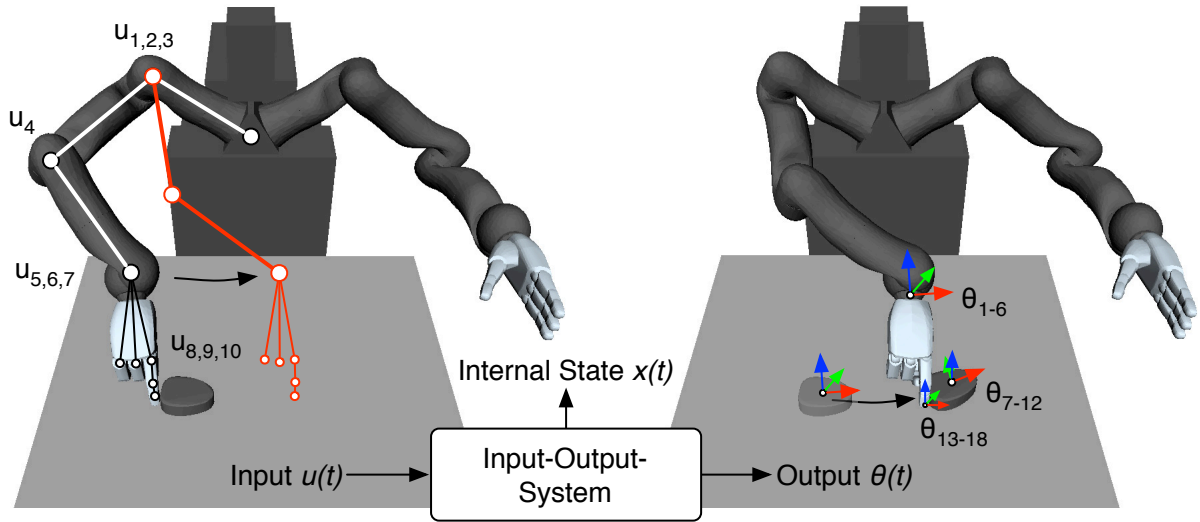
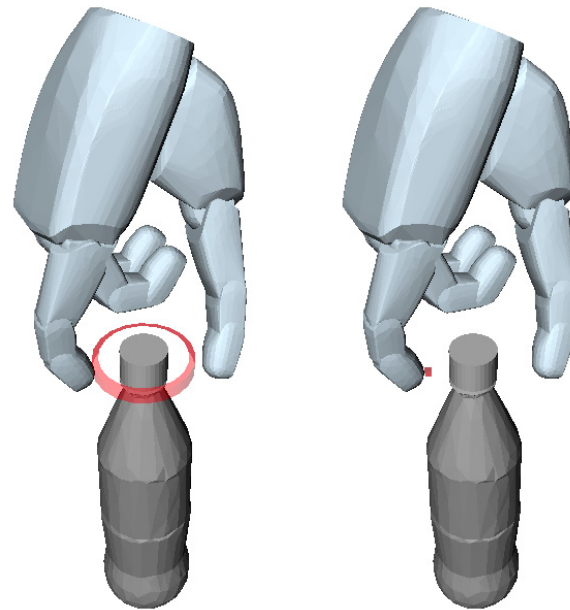


Figure 3.3.: Input-Output-System: controllable joints $u(t)$, internal state $x(t)$ and observable outputs $\theta(t)$. u_{1-7} are the joints of the robot arm, u_{8-10} the joints of the point finger. θ_{1-6} is the 6D-TCP-pose of the arm, θ_{7-12} the pose of the object and θ_{13-18} the pose of the point finger tip.

3.3. In general, an input-output system is defined by its inputs $u(t)$, the internal state $x(t)$ and the outputs $\theta(t)$ at a given time point t [71]. Given a specific input $u(t)$, the internal state $x(t)$ changes resulting in (observable) outputs $\theta(t)$. In our case, $u(t)$ represent the controllable joints of the robot system. If the robot has n degrees of freedom (DOF), the vector $u(t)$ will be represented as a n -dim. real-valued vector, $u(t) \in \mathcal{U} \subseteq \mathbb{R}^n$. The internal state $x(t) \in \mathcal{X}$ represents all relevant information, which is required to completely describe the properties of the system at a given time. In general, $x(t)$ will not be used explicitly but kinematic and dynamic simulation of the robot and environment will be used to calculate the output $\theta(t)$ for a given input $u(t)$.

The outputs $\theta(t) \in \Theta$ represent the observable effects of $u(t)$ on the system. In our case, $\theta(t)$ will be the measured joint values of the robot, the absolute and relative pose, velocity, acceleration of geometric bodies, forces and moments acting on geometric bodies and contacts between geometric bodies. A manipulation task will be defined by restricting the values of $\theta(t)$, i.e. the observable effects in the environment. In planning literature, these kinds of restrictions are referred to as *constraints* [69, p. 791].

In the simplest case, a constraint is defined as the set of values $\mathcal{M}(t)$ of $\theta(t)$. $\mathcal{M}(t)$ represents the so-called *constraint manifold*. The constraint will be obeyed in $\theta(t)$ if it lies in the constraint manifold, i.e. $\theta(t) \in \mathcal{M}(t)$. If a single constraint is used to describe a manipulation task $\mathcal{M}(t)$ will be high-dimensional, e.g. restricting the position of two robot fingers (each 3 DOF), the position and orientation of one robot arm (6 DOF) as well as one object (6 DOF) results in $\mathcal{M}(t)$ being 18-dimensional. In general, drawing samples from a high-dimensional set or calculating the closest point to a given configuration are inefficient and therefore manipulation planning is time-demanding. In order to represent even complex manipulation tasks efficiently, a conjunction of constraints will be used to define the constraint manifold. Each constraint will restrict only



(a) Constraint: fingertip position relative to bottle
 (b) Constraint: fingertip position relative to cap, cap orientation relative to bottle

Figure 3.4.: Different models to represent the screwing task: the cap is grasped with two fingers and rotated. Only constraints for the point finger are shown.

a subset of the (observable) outputs, e.g. a position constraint restricts the relative position of two objects to stay in a 3D region. Dependencies between different outputs will be considered explicitly.

Figure 3.4 shows different models to represent the screwing task. In Figure 3.4a, a cylindrical constraint restricts the fingertip position relative to the bottle. In Figure 3.4b, we use a different model. The cap orientation is restricted and the fingertip has to stay in a cube relative to the cap surface (and not the bottle). This model is more flexible than the first one. If a similar bottle with a larger cap is used, the first model will not adapt to the larger cap. The fingertip constraint is relative to the bottle and the resulting fingertip positions will be in collision with the cap. In the second model, the fingertip constraint is relative to the cap surface and will adapt automatically. In the learning process, a model containing both alternatives is learned and this ambiguity is resolved in the generalization process.

Definition 1 (Constraint) Let \mathcal{E} be a finite set of entities. A constraint $\tau = (e_1, \dots, e_m, f, R)$ with $e_i \in \mathcal{E}$ entities, the order $m \geq 0$, constraint function $f : \mathcal{E}^m \times \Theta \rightarrow \mathbb{R}^k$, $k > 0$, constraint region $R(t) \subseteq \mathbb{R}^k$ will be obeyed in $\theta \in \Theta$ if $f(e_1, \dots, e_m, \theta) \in R$.

The constraint manifold $\mathcal{M}(\tau)$ is defined as $\{\theta \in \Theta \mid f(e_1, \dots, e_m, \theta) \in R\}$. The constraint will be obeyed in $\theta \in \Theta$ if $\theta \in \mathcal{M}(\tau)$.

Definition 1 represents the most general type of a constraint. An example is the *collision constraint* $\tau_{coll} = (e_1, \dots, e_m, f_{coll}, [0, 0])$. In this case, the entities e_i are objects. The constraint function f_{coll} returns ∞ , if a collision occurs between at least one pair of objects $(e_i, e_j), 1 \leq i, j \leq m$, and 0 otherwise. The constraint region is $[0, 0]$, i.e. the constraint will be obeyed if the constraint function is 0. The set of configurations $\mathcal{M}(\tau_{coll}) = \{\theta \in \Theta \mid f_{coll}(e_1, \dots, e_m, \theta) \in [0, 0]\}$ represents the constraint manifold of all collision-free configurations of the robot.

In the motion planning process, rejection sampling [13, p. 528ff] will be applied to find a path in $\mathcal{M}(\tau_{coll})$, i.e. random configurations θ will only be considered in the planning process if $\theta \in \mathcal{M}(\tau_{coll})$. For narrow constraint manifolds, i.e. with vanishing volume, rejection sampling is insufficient since a random configuration has probability close to 0 to lie in the constraint manifold. Projection techniques will be introduced which project a random configuration to a (close) configuration on the constraint manifold. In order to apply projection techniques the constraint representation has to allow for continuous distance computations, see extension 1.

Extension 1 (Constraint-Distance) Let $\tau = (e_1, \dots, e_m, f, R)$ be a constraint. The distance of $\theta \in \Theta$ with respect to τ is defined as

$$d_{\mathcal{M}}(\tau, \theta) = \inf_{r \in R} \|f(e_1, \dots, e_m, \theta) - r\| \quad (3.1)$$

where $d_{\mathcal{M}}(\tau, \cdot)$ is continuous.

The implementation of $\inf_{r \in R} \|f(e_1, \dots, e_m, \theta) - r\|$ depends on the representation of the region R and will be discussed in Section 3.1.3.

Projection techniques represent local optimization algorithms, i.e. the configuration on which they are applied should be close to the constraint manifold. This assumption is valid during motion planning. In order to generate configurations representing the goal of a manipulation task, this assumption can be violated, e.g. the configuration before motion planning does not have to be close to the goal yet. Subsequently, projection techniques will fail and produce a failure of the planning process or longer planning times when applied to random configurations. In order to generate configurations, which are close to the constraint manifold, random samples from $\mathcal{M}(\tau)$ will be generated and inverse kinematics applied.

Extension 2 (Constraints-Sample) Let τ_1, \dots, τ_l be a set of constraints with $\tau_i = (e_{i_1}, \dots, e_{i_{m_i}}, f_i, R_i)$ and $f_{\tau_1, \dots, \tau_l}^{-1} : R_1 \times \dots \times R_l \rightarrow \Theta$ measurable. A random sample $\theta_{rnd} \in \Theta$ is obtained by

$$r_i \sim \mathcal{U}(R_i) \quad (3.2)$$

$$\theta_{rnd} = f_{\tau_1, \dots, \tau_l}^{-1}(r_1, \dots, r_l) \quad (3.3)$$

The random sample θ_{rnd} will be valid if and only if $\theta_{rnd} \in \bigcap M(\tau_i)$. It is distributed as $\theta_{rnd} \sim f_{\tau_1, \dots, \tau_l}^{-1}(\mathcal{U}(R_1) \times \dots \times \mathcal{U}(R_l))$.

The implementation of f^{-1} depends on the type of constraints. In the motion planning process, inverse kinematics will be applied to generate random configurations based on position, orientation and direction constraints, which will be discussed in Section 3.4.2. Figure 3.7 shows an example, which will be described in Section 3.1.2.

Based on this general definition of constraints, constraint-distance and constraints-sampling the learning, generalization and planning algorithms will be defined. In order to represent bimanual, dexterous manipulation tasks, the set of entities and the set of constraints has to be defined explicitly, which will be discussed in the next subsection.

3.1.2. Constraint Types

When a service robot manipulates an object, it will subsequently make contact between his fingers and the object to apply forces and moments to induce a specific object motion. Since we assume rigid bodies, the object motion is described completely by specifying the motion of a coordinate frame, which is rigidly attached to the object. The same principle applies to the robot motion as well. Based on this observation, the set of entities contains a large number of coordinate frames, which are either attached to the robot, e.g. a coordinate frame in the fingertip of the robot, or attached to an object, e.g. the coordinate frame in the center of a button surface.

The general definition of constraints allows to express dependencies between an arbitrary number of entities, which is expressed by the order m of the constraints. The number of constraints of order m , which can be represented using the entities in \mathcal{E} , is $\binom{|\mathcal{E}|}{m}$ in $O(|\mathcal{E}|^m)$ using Big O notation [26, p. 41-50]. Since no assumptions about the relevance of specific constraints to represent a manipulation task are made, all constraints up to a maximum order will be considered. In probabilistic PbD, e.g. [21], the configuration space Θ is usually divided into independent subspaces

$$\Theta = \Theta_1 \times \dots \times \Theta_n, \quad (3.4)$$

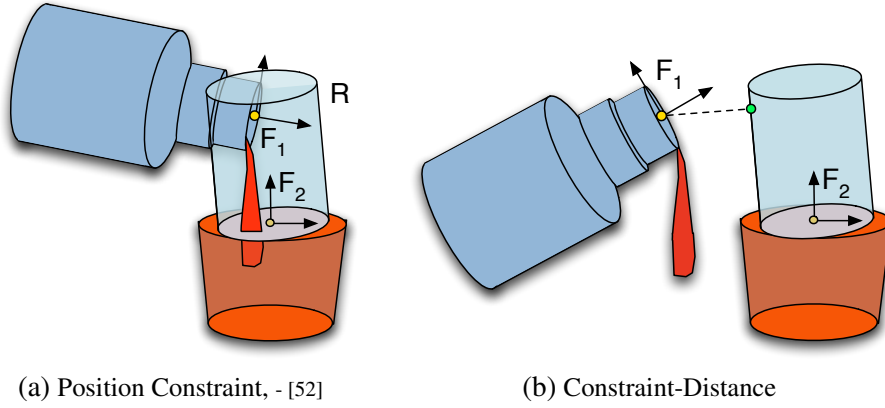
e.g. to model constraints in joint and task space. The manipulation task is represented by a joint probability density function at time point t :

$$p(\theta|t) = p(y_1, \dots, y_n) = \mathcal{N}(\mu_1, \Sigma_1) \cdot \dots \cdot \mathcal{N}(\mu_n, \Sigma_n), \quad (3.5)$$

where $\mathcal{N}(\mu_i, \Sigma_i)$ is a Gaussian distribution with mean μ_i and covariance Σ_i . Since the covariance describes pair-wise correlations, this maps to constraints of order 2, which is sufficient to represent simple manipulation tasks.

Position, Orientation, Direction, Force and Moment Constraints

In this thesis position, orientation, direction, force and moment constraints up to order 3, which will be called *cartesian constraints*, will be considered. A cartesian constraint is defined as



(a) Position Constraint, - [52]

(b) Constraint-Distance

Figure 3.5.: The position constraint $(F_1, F_2, f_{pos}, Cylinder((0, 0, 0, 5, 5, 0)^T, 0, 80, 180, 0, 20))$ will be obeyed if the origin of 2F_1 , i.e. the relative transformation of coordinate Frame F_1 to coordinate frame F_2 , stays in the slightly rotated cylinder (light blue) above the cup (red).

$\tau = (e_1, e_2, e_3, f_{cart}, R)$ with e_1, e_2, e_3 coordinate frames, $f_{cart} \in \{f_{pos}, f_{ori}, f_{dir}, f_{for}, f_{mom}\}$ constraint functions and region R . The position and orientation of each coordinate frame depends on the current configuration θ . To simplify the descriptions, the dependency on θ will be omitted. A coordinate frame a will be described using a homogenous transformation matrix. The position and orientation relative to a second coordinate frame b is defined as ${}^bH_a = ({}^bRot_a, {}^b t_a)$, where bRot_a is a 3x3 rotation matrix and ${}^b t_a$ a 3D vector. A 3x3 matrix bRot_a is a redundant description of the orientation of the coordinate frame with 9 parameters. Representing a continuous set of orientations is difficult since the 9 parameters cannot be chosen arbitrarily. The Simultaneous Orthogonal Rotations Angle (SORA) representation [99] has only 3 parameters and allows to reconstruct a rotation matrix efficiently. In order to obtain the SORA representation, we transform the rotation matrix bRot_a into a quaternion (qx, qy, qz, qw) . The quaternion can be regarded as a rotation around the 3D axis $\frac{1}{\sin \frac{\theta}{2}}(qx, qy, qz)^T$ with right-handed angle $\theta = 2 \cos^{-1} qw$. The SORA representation of the rotation matrix bRot_a is

$$\text{axis}({}^bRot_a) = \frac{\theta}{\sin \frac{\theta}{2}}(qx, qy, qz)^T. \quad (3.6)$$

The main advantages compared to a rotation matrix, a quaternion or Euler angles is that the representation is three dimensional and continuous, i.e. we can reconstruct a rotation matrix based on an arbitrary 3D vector. The latter enables us to use unified optimization and sampling algorithms to work with different constraint types.

In order to describe the dependency of e_1 on e_2 and e_3 a third reference coordinate frame e_{23} is defined with the position of e_2 and the orientation of e_3 . Formally, ${}^0H_{e_{23}} = ({}^0Rot_{e_3}, {}^0 t_{e_2})$.

Position constraints restrict the position of e_1 relative to e_{23} , i.e. ${}^{e_{23}}t_{e_1}$, to stay in the region R . In Figure 3.5a, a position constraint is shown, which restricts the opening of a bottle F_1 to stay in a cylindrical region fixed to F_2 , the opening of a cup.

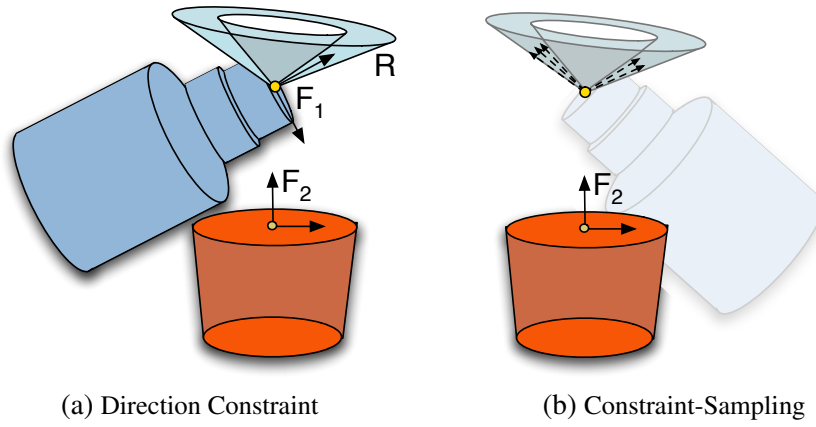


Figure 3.6.: The direction constraint $(F_1, F_2, f_{dir}, Cone((0, 0, 0, -3, 0, 0)^T, 0, 1, 45, 55, 45, 55))$ will be obeyed if the z-axis of F_1 stays in the cone (blue).

Orientation constraints will be obeyed if the SORA representation of the orientation of e_1 relative to e_{23} . i.e. ${}^{e_{23}}Rot_{e_1}$, is included in the region R .

Direction constraints are used to represent a degree of freedom around a symmetry axis. Assuming the symmetry axis is $(0, 0, 1)^T$ in e_1 , the axis is transformed into e_{23} . It will be obeyed if the rotated axis, i.e. ${}^{e_{23}}Rot_{e_1}(0, 0, 1)^T$, stays in the region R , see Figure 3.6a.

Force and Moment constraints restrict a certain force or moment represented by e_1 to stay in the region R relative to e_2 and e_3 . The main difference to a position constraint is that it will be transformed like a direction, i.e. only the rotation ${}^{e_{23}}Rot_0$ will be applied since the magnitude is invariant to coordinate transformations.

The constraint functions with values in \mathbb{R}^3 are defined [52] by

$$f_{pos}(e_1, e_2, e_3, \theta) = {}^{e_{23}}t_{e_1} \quad (3.7)$$

$$f_{ori}(e_1, e_2, e_3, \theta) = \text{axis}({}^{e_{23}}Rot_{e_1}) \quad (3.8)$$

$$f_{dir}(e_1, e_2, e_3, \theta) = {}^{e_{23}}Rot_{e_1}(0, 0, 1)^T \quad (3.9)$$

$$f_{for}(e_1, e_2, e_3, \theta) = f_{mom}(e_1, e_2, e_3, \theta) = {}^{e_{23}}Rot_0 {}^0t_{e_1} \quad (3.10)$$

In order to generate a random configuration, in which the constraint is obeyed, the constraint function has to be inverted. f_{pos} , f_{for} , f_{ori} and f_{dir} can be inverted in the sense that with fixed e_2, e_3, θ a value for e_1 can be obtained for which the constraint is obeyed, e.g. with $x \in f_{pos}^{-1}$ [52]:

$$f_{pos}(f_{pos}^{-1}(x, e_2, e_3, \theta), e_2, e_3, \theta) = x \quad (3.11)$$

In the planning process, random configurations will be generated and projection techniques applied to adapt a configuration until all constraints are obeyed. The projection techniques require to calculate the distance to the constraint manifold.

Constraint-Distance A continuous distance function is given by the Euclidean distance to the nearest point in R : $\inf_{r \in R} \|f_{cart}(e_1, e_2, e_3, \theta) - r\|$. In Figure 3.5b, the position constraint to place a bottle opening above a cup opening is violated. The constraint-distance is calculated by finding the closest point on the cylinder and calculating the distance of e_1 to it.

Constraints-Sample Learned planning models usually contain a large number of constraints Ψ . Projection techniques, see Section 3.4.1, allow to project a random configuration to the constraint manifold $\mathcal{M}(\Psi)$ using local optimization techniques. The diversity of the projected configurations is limited, i.e. most will lie on the surface of the constraint manifold and not the interior. Additionally, computation time depends on the initial configuration. We generate more suitable random configurations by sampling values in the regions of a subset of Ψ and using inverse kinematics to generate a configuration based on the sampled values.

We sample a position and an orientation or direction constraint for each object to determine the values of each DOF. First, we have to deduce the object to which a coordinate frame belongs. Each coordinate frame e_1 can be written as ${}^0H_{e_1} = {}^0H_{d_n} {}^{d_n}H_{d_{n-1}} \dots {}^{d_2}H_{d_1} {}^{d_1}H_{e_1}$, where d_i are coordinate frames. For each 3D rigid body $o \in \mathcal{O}$, there exists a unique coordinate frame o representing the object coordinate frame, in which the geometry is defined. The object assigned to a given coordinate frame e_1 is defined as the first object coordinate frame, on which e_1 depends: $o(e_1) = \arg \min_{d_i \in \mathcal{O}} i$. For the constraint $\tau = (e_1, \dots, e_m, f, R)$, we set $o(\tau, i) = o(e_i^{\tau}) = o(e_i)$.

A constraint sampling set Ψ_S for Ψ is defined as a set of constraint pairs. Each pair constrains the same object and consists of a position and a orientation or direction constraint. From the position constraint, a random position vector is drawn. The orientation or direction constraint is used to generate a random rotation matrix. Both samples are combined resulting in a random pose of the constrained object. If the object belongs to the robot, e.g. the TCP, inverse kinematics are applied based on the random pose to generate a random robot configuration.

The constraint sampling set Ψ_S is defined as

$$\Psi_S = \{(\tau, \varphi) \mid \tau, \varphi \in \Psi \quad (3.12)$$

$$\tau \text{ position constraint,} \quad (3.13)$$

$$\varphi \text{ orientation or direction constraint,} \quad (3.14)$$

$$o(\tau, 1) = o(\varphi, 1) \quad (3.15)$$

Let $\tau = (e_1^{\tau}, e_2^{\tau}, e_3^{\tau}, f^{\tau}, R^{\tau})$ and $\varphi = (e_1^{\varphi}, e_2^{\varphi}, e_3^{\varphi}, f^{\varphi}, R^{\varphi})$. A set of random object poses is generated by drawing a sample r from R^{τ} and s from R^{φ} :

$$r \sim \mathcal{U}(R^{\tau}) \quad (3.16)$$

$$s \sim \mathcal{U}(R^{\varphi}) \quad (3.17)$$

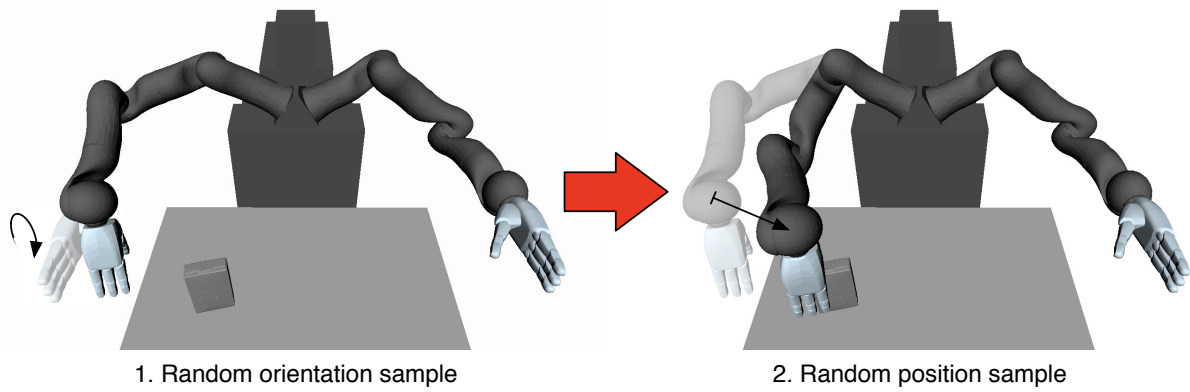


Figure 3.7.: Example of Constraint-Sampling. First, the orientation constraint (TCP, Box, f_{ori} , $\text{Cube}(-0.1, 0.1, -0.5, 0.5, -0.1, 0.1)$) is sampled to generate a random rotation matrix. Second, the position constraint (TCP, Box, f_{pos} , $\text{Cube}(-20, 20, -60, 60, -30, 30)$) is sampled to generate a random position vector. Random position vector and random rotation matrix are combined into a homogenous matrix and inverse kinematics are applied to generate a random configuration, in which the TCP pose is equal to the homogenous matrix.

and using the inverse constraint functions to compute the pose

$$\forall(\tau, \varphi) : {}^0H_{o(\tau)} = ({}^0Rot_{o(\tau)}, {}^0t_{o(\tau)}) \quad (3.18)$$

$$(\cdot, {}^0t_{o(\tau)}) = f^{\tau-1}(r, e_2^{\tau}, e_3^{\tau}, \theta) \quad (3.19)$$

$$({}^0Rot_{o(\tau)}, \cdot) = f^{\varphi-1}(s, e_2^{\varphi}, e_3^{\varphi}, \theta) \quad (3.20)$$

In the motion planning process, the position and orientation of a 3D rigid body might constrain the joint angles of the robot system, e.g. if the 3D rigid body is grasped by the robot or refers to a body part. Based on this observation, f^{-1} will be defined using inverse kinematics to generate a random configuration based on the set of random object poses.

Figure 3.7 shows an example to sample a robot configuration based on a position and orientation constraint. Each constraint restricts the TCP relative to the box on the table. First, a random rotation matrix ${}^{box}Rot_{TCP}$ for the TCP is sampled using the orientation constraint. Second, a random position ${}^{box}t_{TCP}$ is generated using the position constraint. Both results are combined into a homogenous matrix ${}^{box}H_{TCP} = ({}^{box}Rot_{TCP} {}^{box}t_{TCP})$ and inverse kinematics are applied to generate a random configuration, where the TCP pose is ${}^{box}H_{TCP}$.

Contact Constraints

Manipulation tasks in household environments, e.g. opening a bottle or pushing buttons, often require to maintain contact between the robot fingers and the surface of the manipulated objects. The representation of manipulation tasks has to be general enough to be applied to objects with similar shape, e.g. bottles with different sizes of caps. If contacts are only expressed by position constraints between the two 3D bodies, generalization to different object geometries is difficult since the constraint region has to capture local characteristics of the object surface and be auto-

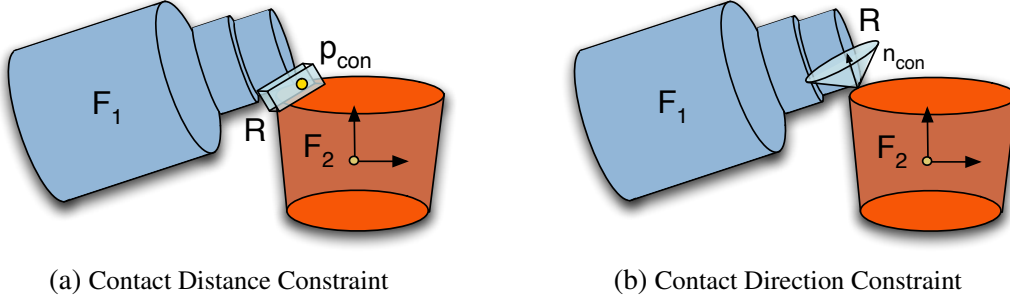


Figure 3.8.: The contact constraint will be obeyed if the distance vector of the 3D models assigned to F_1 and F_2 is inside the cube and the contact normal is inside the cone.

matically adapted to different object surfaces. In general, the combination with force constraints is not applicable since, on the robot system, not all forces between 3D bodies in the environment can be measured, e.g. if the robot has to move a staple of objects, forces between objects cannot be measured.

Contact Constraint By using the surface of the 3D body explicitly to define contact constraints, the constraint is more flexible. A *contact constraint* is defined as $\tau = (e_1, e_2, f_{con}, R)$ with objects e_1, e_2 , constraint function $f_{con} \in \{f_{cdis}, f_{cdir}\}$ and constraint region R . In state-of-the-art proximity query packages, e.g. C2A [104], the distance vector $d_{con}(e_1, e_2)$ between two 3D bodies e_1, e_2 can be computed efficiently. Additionally, the contact point $p_{con}(e_1, e_2)$, i.e. the closest point on the object surface of e_2 , as well as the contact normal $n_{con}(e_1, e_2)$, i.e. the direction vector perpendicular to the object surface in the contact point, are computed. A contact distance constraint will be obeyed if the distance vector is included in the constraint region, see Figure 3.8a. In the same way, a contact direction constraints restricts the contact normal to stay in the constraint region, see Figure 3.8b:

$$f_{cdis}(e_1, e_2, \theta) = {}^{e_2}Rot_0 \cdot d_{con}(e_1, e_2) \quad (3.21)$$

$$f_{cdir}(e_1, e_2, \theta) = {}^{e_2}Rot_0 \cdot n_{con}(e_1, e_2) \quad (3.22)$$

In order to monitor contact constraints, the constraint-distances are defined based on the Euclidean distance. In the motion planning process, a configuration will be projected to the constraint manifold $\mathcal{M}(\tau)$ using continuous collision detection (CCD) algorithms. The constraints-sample is obtained by using penetration depth (PD) algorithms, which will be explained in Section 3.4.1.

Configuration Constraints

In the previous subsections, constraints restricting the motion of 3D rigid bodies were developed. In robotics, 3D rigid bodies often form kinematic chains imposing constraints on the relative pose between two bodies, e.g. a robot arm or a cupboard with door. These degrees of freedom will be

considered explicitly in the motion planning process. In Θ , they are represented as a real-valued vector of joint angles for each robot and object. An example of a configuration constraint is the set of joint limits, i.e. the n -dimensional vector of joint angles has to stay in a n -dimensional hypercube. Additionally, configuration constraints are required to ensure that certain DOF will not change, e.g. in order to press a button, only one finger will move and the others remain static. In the learning process, configuration constraints will be automatically generated to reduce the dimensionality of the motion planning problem in a flexible way.

A *configuration constraint* is defined as $\tau = (e_1, 1_n, R)$ with $e_1 \in \mathbb{R}^n$, constraint function 1_n , i.e. the identity in \mathbb{R}^n , and constraint region $R \subseteq \mathbb{R}^n$. The constraint-distance is defined as the Euclidean distance in \mathbb{R}^n . In order to generate a random configuration, in which the configuration constraint is obeyed, f_τ^{-1} is set to 1_n .

Temporal Constraints

In general, manipulation tasks consist of multiple subgoals, e.g. first place a finger on a button and then apply force to it. The ordering of subgoals as well as the duration of the motion from one subgoal to the next has to be considered. Since the time t is represented explicitly, constraints on the temporal domain can be defined.

A *temporal constraint* is defined as $\tau = (t, 1, [l_R, u_R])$, where t is the time, 1 the identity in \mathbb{R} and $l_R \leq u_R, l_R, u_R \in \mathbb{R}$ the minimum and maximum time.

Since temporal constraints are represented using the identity in \mathbb{R} , distance calculation and sampling are defined in the same way as for configuration constraints.

Additional Constraints

In motion planning, collision checking as well as calculation of grasp quality represent standard tools to model Pick-and-Place tasks. Since collision checks as well as grasp quality heavily depend on the 3D geometry, additional constraints will be defined. By introducing additional constraints, the framework to learn, generalize and execute planning models can be efficiently extended.

The *collision constraint* is defined as $\tau_{coll} = (e_1, \dots, e_m, f_{coll}, [0, 0])$ with the objects e_i and the constraint function f_{coll} . f_{coll} returns ∞ , if a collision occurs between at least one pair of objects $(e_i, e_j), 1 \leq i, j \leq m$, and 0 otherwise.

The *grasp quality constraint* is defined as $\tau_{qual} = (e_1, f_{qual}, [l_R, u_R])$ with the object e_1 , the constraint function f_{qual} and $0 \leq l_R \leq u_R, l_R, u_R \in \mathbb{R}$. f_{qual} returns the value of the grasp quality measure on e_1 . l_R is the minimum and u_R the maximum grasp quality. Usually $u_R = \infty$, i.e. only the minimum grasp quality l_R has to be achieved.

3.1.3. Constraint Regions

The representation of a constraint region R plays an important role in the motion planning and learning process. In the motion planning process, the representation has to allow different compu-

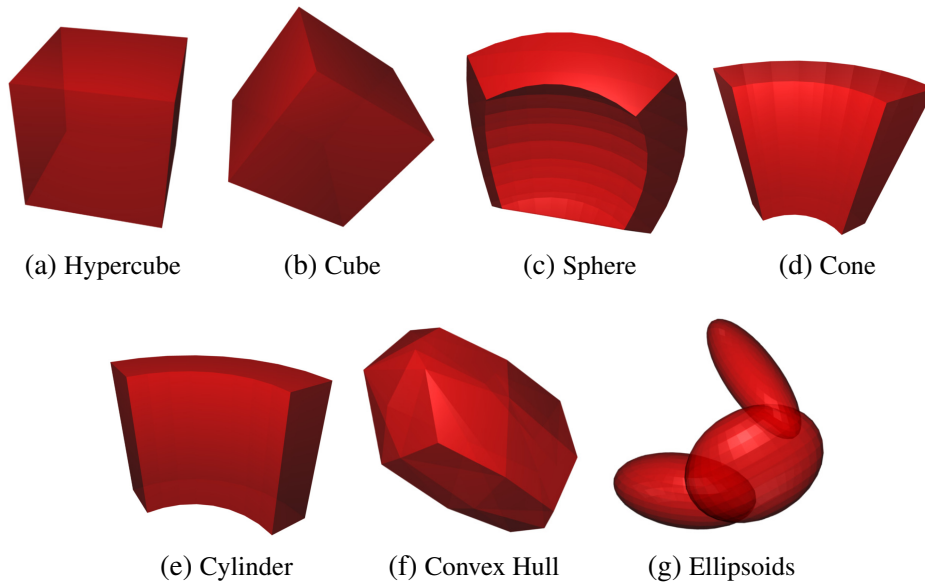


Figure 3.9.: Visualization of different region types

tations in an efficient way: inclusion, distance computation and sampling. For a given configuration, it has to be tested efficiently if the value of the constraint function is included in the constraint region. Second, the distance of the value of the constraint function to the constraint manifold has to be computed efficiently. Third, the constraint region has to allow to draw a random value in an efficient way. In the learning process, a representation has to be chosen which can be learned efficiently and approximates the training data sufficiently.

In summary, a constraint region $R \subseteq \mathbb{R}^n$ is described by the following four characteristics

1. Inclusion: $r \in R$.
2. Distance computation: $\forall x \in \mathbb{R}^n : \inf_{r \in R} \|x - r\|$.
3. Sampling: $r \sim \mathcal{U}(R)$.
4. Parameterization: $X \subseteq \mathbb{R}^n, \mu$ surface area $\mathbb{R}^n, R = \arg \min_{X \subseteq R'} \mu(R')$.

In order to use a constraint region during motion planning in an efficient way, we assume that the first three characteristics can be computed in $O(1)$.

Six different types of constraint regions will be defined [51]: hypercube, cube, cylinder-, sphere-, cone-sectors, convex hull, sequence of ellipsoids, see Figure 3.9. The parameterization and implementation of the four characteristics will be discussed in the next subsections.

Hypercube

A hypercube, see Figure 3.9a, is a generalized representation of a closed interval in a high-dimensional space. The parameters are given by $c_i, l_i, u_i \in \mathbb{R}, 1 \leq i \leq n, l_i \leq u_i$. Let $x, r \in \mathbb{R}^n, X \subseteq \mathbb{R}^n$. It follows that

$$(1) \quad \forall i : l_i \leq r_i - c_i \leq u_i \quad (3.23)$$

$$(2) \quad \|x - r\| \text{ with } \forall i : r_i = c_i + \min(u_i, \max(l_i, x_i - c_i)) \quad (3.24)$$

$$(3) \quad \forall i : r_i \sim c_i + \mathcal{U}([l_i, u_i]) \quad (3.25)$$

$$(4) \quad \forall i : c_i = \frac{1}{2}(\min_{x \in X} x_i + \max_{x \in X} x_i), \quad (3.26)$$

$$l_i = c_i - \min_{x \in X} x_i, u_i = c_i - l_i$$

Cube

In \mathbb{R}^3 , the definition of a hypercube leads to an axis-aligned boundary box of the learning data X in (4). A better approximation of X can be achieved by using an oriented boundary box, see Figure 3.9b. The parameters of a hypercube in \mathbb{R}^3 are extended by a 3x3 rotation matrix Rot , which represents the rotation of the principal axes. In the Equations 3.23ff, Rot^{-1} has to be applied, e.g.

$$(1) \quad r' = Rot^{-1}(r - c), \forall i : l_i \leq r'_i \leq u_i \quad (3.27)$$

The main difference is that in (4) a 3D optimization problem has to be solved, which can be done efficiently using Rosenbrock [86].

Sphere

A sphere, see Figure 3.9c, is parameterized by the center $c \in \mathbb{R}^3$, a 3x3 rotation matrix Rot , inner angles $l_\alpha, u_\alpha \in [0, \pi], l_\alpha < u_\alpha$, inner angles $l_\beta, u_\beta \in [0, \pi], l_\beta < u_\beta$ and radii $l_r, u_r \in \mathbb{R}_{>0}, l_r < u_r$. It holds that

$$(1) \quad r \in R \Leftrightarrow l_r \leq \|x\| \leq u_r \wedge \quad (3.28)$$

$$l_\alpha \leq \text{acos}(x_1) \leq u_\alpha \wedge \quad (3.29)$$

$$l_\beta \leq \text{acos}(x_2) \leq u_\beta \quad (3.30)$$

$$\text{with } x = Rot^{-1}(r - c) \quad (3.31)$$

It can be shown that (2, 3) can be computed in $O(1)$ and it is sufficient to solve a 3D optimization problem to compute (4).

Cone

A cone, see Figure 3.9d, is parameterized in the same way as a sphere with

$$(1) r \in R \Leftrightarrow l_r \leq x_3 \leq u_r \wedge \quad (3.32)$$

$$l_\alpha \leq \text{acos}(x_1) \leq u_\alpha \wedge \quad (3.33)$$

$$l_\beta \leq \text{acos}(x_2) \leq u_\beta \quad (3.34)$$

$$\text{with } x = \text{Rot}^{-1}(r - c) \quad (3.35)$$

It can be shown that (1-3) can be computed in $O(1)$ and it is sufficient to solve a 3D optimization problem to compute (4).

Cylinder

The parameters of a cylinder, see Figure 3.9e, are given by the center $c \in \mathbb{R}^3$, a 3x3 rotation matrix Rot , inner angles $l_\alpha, u_\alpha \in [0, \pi], l_\alpha < u_\alpha$, radii $l_r, u_r \in \mathbb{R}_{>0}, l_r < u_r$ and heights $l_h, u_h \in \mathbb{R}, l_h < u_h$. It follows that

$$(1) r \in R \Leftrightarrow l_h \leq x_3 \leq u_h \wedge \quad (3.36)$$

$$l_\alpha \leq \text{acos}(x_1) \leq u_\alpha \wedge \quad (3.37)$$

$$l_r \leq \sqrt{x_1^2 + x_2^2} \leq u_r \quad (3.38)$$

$$\text{with } x = \text{Rot}^{-1}(r - c) \quad (3.39)$$

It can be shown that (1-3) can be computed in $O(1)$ and it is sufficient to solve a 3D optimization problem to compute (4).

Convex Hull

Let $X \subseteq \mathbb{R}^n$ be a finite set of points. The convex hull, see Figure 3.9f, is defined as the minimal convex set containing X . Since the convex hull in \mathbb{R}^n is a convex polytope, which can be defined as the intersection of half-spaces or by its extreme points, the parameters are m half-spaces $(\alpha_i, a_i) \in \mathbb{R}^n \times \mathbb{R}$ and l extreme points $\beta_i \in X$. It follows that

$$(1) \quad \exists i : r \cdot \alpha_i > a_i \quad (3.40)$$

$$(2) \quad \max(0, \max_i (r \cdot \alpha_i - a_i)) \quad (3.41)$$

$$(3) \quad \sum w_i \beta_i \text{ with } w \sim \mathcal{U}(\{v \mid v_i \in [0, 1], \sum v_i = 1\}) \quad (3.42)$$

Different algorithms exist to compute (4). In this thesis, Qhull [24] is used, which implements the Quickhull-algorithm [5].

Sequence of Ellipsoids

The previously defined region types are usually time-independent. This results in a stationary constraint manifold, on which a trajectory can be generated efficiently in the motion planning process. For some manipulation tasks, the execution of a relatively fixed trajectory, e.g. to write a letter, might be necessary. In order to generate trajectories similar to a reference trajectory time-dependent constraints have to be introduced.

A sequence of ellipsoids, see Figure 3.9g, is defined by a sequence $h_i \in \mathbb{R}^n, H_i \in \mathbb{R}^{n \times n}$, positive definite, $L_i \in \mathbb{R}^{n \times n}$, lower triangular, $L_i L_i^T = H_i^{-1}, 0 \leq i \leq m$. We calculate L_i using the Cholesky-decomposition. Each (h_i, H_i) defines an ellipsoid with center h_i by

$$1 = (x - h_i)^T H_i (x - h_i) \quad (3.43)$$

with $x \in \mathbb{R}^3$. L_i transforms the unit sphere into the ellipsoid, which we use to calculate the approximate distance to the ellipsoid. First, we calculate the distance vector to the center. We project the distance to the unit sphere. If the point lies outside the sphere, we calculate the closest point on the sphere, project the closest point back to the ellipsoid and calculate its distance to the original point. Let $t \in [0, 1]$ be the normalized time and $i = \lfloor t \cdot m \rfloor$ then

$$(1) \quad L_i^T (r - h_i) \in [0, 1]^n \quad (3.44)$$

$$(2) \quad \begin{cases} 0, & \text{if } \|y\| < 1 \\ \|x - r\| & \text{otherwise} \end{cases} \quad (3.45)$$

$$\text{with } y = L_i^T (x - h_i), r = h_i + L_i \frac{y}{\|y\|} \quad (3.46)$$

$$(3) \quad r = h_i + L_i \cdot u \quad \text{with } u \sim \mathcal{U}([0, 1]^n) \quad (3.47)$$

In [35], a derivation of the exact distance algorithm can be found. In order to generate a sequence of ellipsoids for the training set $X \subseteq \mathbb{R}^n$ in (4), the EM-algorithm is applied to learn a Gaussian Mixture Model (GMM) $p(x, t)$. Gaussian Mixture Model Regression (GMM) [21] is applied to generate the conditional probability density function $p(x|t) = \mathcal{N}(\mu_t, \Sigma_t)$. The parameters of the sequence of ellipsoids are defined by $h_i = \mu_t$ and $H_i = 9\Sigma_t^{-1}$ with $i = \lfloor t \cdot m \rfloor$ representing the covariance ellipsoid with 3 standard deviations.

3.1.4. Discussion

The constraint representation is fundamental to the definition of planning models. Constraints are an efficient way to define complex constraint manifolds, which are suited to define the goals of a manipulation task as well as restrict the search space for planning. A consistent representation of goals and constraints allows to apply general learning, generalization and planning techniques, which will be discussed in the remaining part of the thesis. Motion planning in high-dimensional

configuration spaces is currently insufficient to find robot trajectories, which optimize a given objective function. Since a soft constraint is effectively an objective function, we cannot guarantee that the soft constraint is obeyed (as best as possible). Therefore, the representation is based on hard constraints, which allow to decide if the constraint is obeyed in a given configuration or not. The actual representation of the constraint region was defined implicitly by four characteristics, which allows to use arbitrary representations suitable for constraint-based motion planning. The defined set of constraint regions can be easily extended but has proven sufficient for simple manipulation tasks. By using 3D bodies, we can visualize constraints and apply state-of-the-art geometric tools to manipulate constraints. We represent constraints for the position and orientation of a 3D body separately. Couplings between position and orientation are restricted to ease the problem to identify relevant constraints, i.e. in the most complex case the position or orientation of a 3D body depends on the position of one coordinate frame and the orientation of a second coordinate frame. The intersection of a set of constraints (including contact constraints) allows to represent more complex constraint manifolds efficiently.

3.2. Planning Models

In the previous section, a manipulation task was identified with a complex, time-dependent constraint manifold. The constraint manifold was represented as the intersection of constraint manifolds defined by a set of constraints. Intuitively, a manipulation task consists of multiple subgoals, which allow to separate the task into several segments. A simple example is the Peg-in-Hole task, which can be solved by first rotating the grasped object, moving it down until contact, aligning it with the hole and moving down until the goal depth is reached.

Each segment can be described separately by a set of constraints, which have to be obeyed during the transition from the subgoal assigned to the start of the segment to the subgoal assigned to the end. In the same way, each subgoal can be identified with the set of configurations, in which it is fulfilled. The set of subgoal configurations defines a constraint manifold and consequently, each subgoal will be described by a set of constraints.

The Peg-in-Hole task is an example of a linear planning model, in which all subgoals have at most one outgoing and one ingoing segment. For bimanual manipulation tasks, e.g. in which both robot arms manipulate different objects, it might be necessary to work on (and plan) some subgoals in parallel. These complex temporal dependencies between subgoals will be represented by temporal constraints. In the next subsection, the strategy graph will be defined based on the set of constraints.

3.2.1. Strategy Graph

A planning model can be described as a directed graph, called *strategy graph* [51]. Nodes represent (sub-)goals of the task. Arcs represent segments, i.e. transitions between (sub-)goals. Each node and arc will be described by a set of constraints containing at least one temporal constraint. The

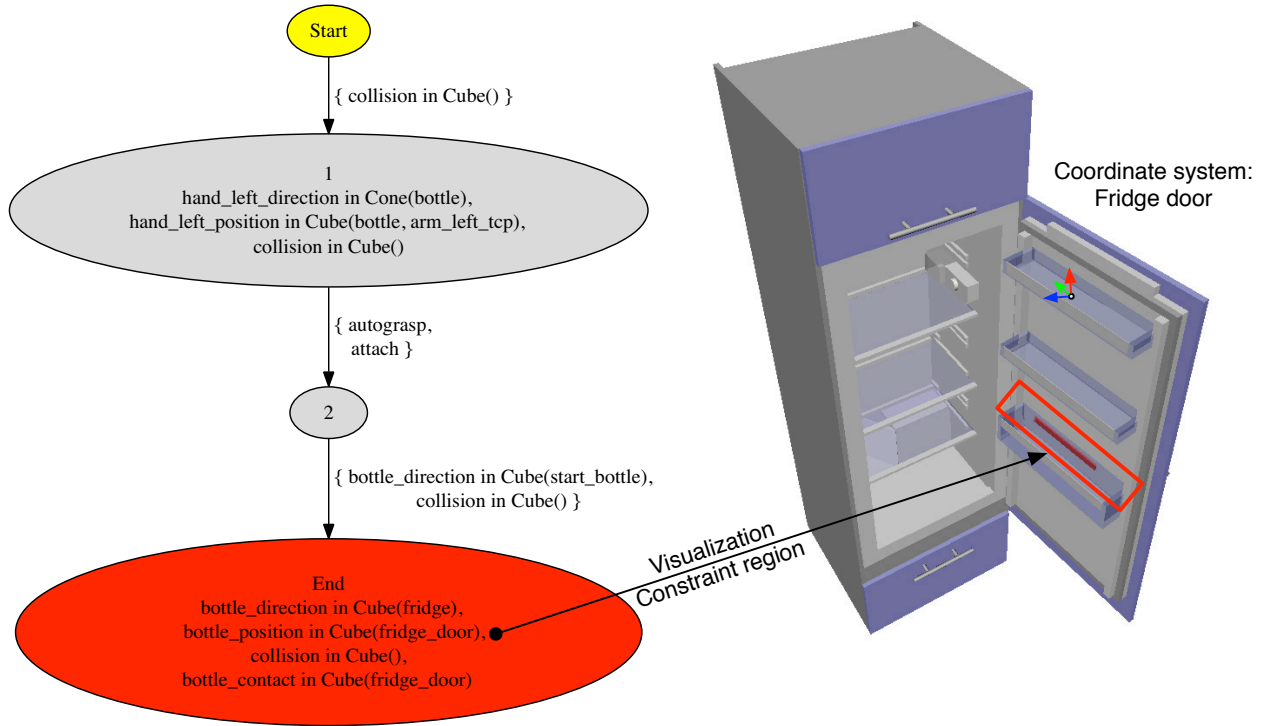


Figure 3.10.: Strategy graph (left) to grasp a bottle and place it in a fridge door. The position constraint restricts, where the bottle has to be placed in the fridge door. The constraint adapts automatically to different door angles.

temporal constraints are necessary to represent the temporal ordering and dependencies between different (sub-)goals. In Figure 3.10, a planning model to grasp a bottle and place it in the lower shelf of a fridge door is visualized.

The time-dependent constraint manifold $\mathcal{M}(t)$ describes the constraints, which have to be obeyed at time t in order to execute the manipulation task. In the motion planning process, a concrete time point t_i will be assigned to the i th node. The constraint manifold $\mathcal{M}(t_i)$ defines the set of configurations, in which the i th subgoal is fulfilled. Consequently, we identify node i with $\mathcal{M}(t_i)$ and define it by the set of node constraints $\Psi_{node}(i)$. In the same way, arc (i, j) from node i to node j with $t_i \leq t_j$ is identified with the constraint manifolds $\mathcal{M}_{|t_i < t < t_j}$, which will be defined by a single fixed set of arc constraints $\Psi_{arc}(i, j)$. Based on this similarity, (sub-)goals of a manipulation task will be treated in the same way as manipulation constraints.

Definition 2 (Goals and Constraints) *A (sub-)goal i of a manipulation task will be identified with the set of configurations, in which it is fulfilled. The set of configurations will be defined implicitly by a set of constraints $\Psi_{node}(i)$. Due to this similarity with constraints, which restrict the transition from one subgoal to the next, (sub-)goals will be referred to as constraints as well.*

Due to this unification, constraints define the atomic building blocks of the representation of planning models for manipulation tasks.

3.2.2. Constraint Satisfaction Problem

A planning model is defined as a tuple $(\mathcal{X}, \Upsilon_{node}, \Psi_{node}, \Upsilon_{arc}, \Psi_{arc}, \theta)$ with the variables \mathcal{X} , temporal node constraints Υ_{node} , domain node constraints Ψ_{node} , temporal arc constraints Υ_{arc} , domain arc constraints Ψ_{arc} and path θ . The temporal constraints were defined in Section 3.1.2. Domain constraints consist of position, orientation, direction, force, moment, configuration and arbitrary constraints, as defined in Section 3.1.2.

A preliminary version of the representation based on a different constraint representation was published by the author in [51].

$\theta : \mathbb{R}_{\geq 0} \rightarrow \Theta$, θ continuous, is a continuous path in the configuration space. In the planning process, the goal is to find a path θ , on which all constraints are obeyed. $\mathcal{X} \subset \mathbb{R}_{\geq 0}$ is a finite set of variables x_i in the temporal domain, which corresponds to the set of nodes in the graph representation. The variables describe specific time points at which a subgoal has to be reached. The time point t_i of x_i is restricted explicitly by a set of temporal constraints $\Upsilon_{node}(i)$:

$$\forall \tau = (e_1, \dots, e_m, f, R) \in \Upsilon_{node}(i) : f(e_1, \dots, e_m, \theta) \in R \quad (3.48)$$

$$\Leftrightarrow \forall \tau = (t_i, 1, [l_R, u_R]) \in \Upsilon_{node}(i) : l_R \leq t_i \leq u_R \quad (3.49)$$

The set of configurations at time point t_i is restricted by a set of domain constraints $\Psi_{node}(i)$:

$$\forall \tau = (e_1, \dots, e_m, f, R) \in \Psi_{node}(i) : f(e_1, \dots, e_m, \theta(t_i)) \in R \quad (3.50)$$

The time span between t_i and t_j , i.e. the temporal ordering of the variables x_i and x_j , is restricted by a set of temporal constraints $\Upsilon_{arc}(i, j)$:

$$\forall \tau = (t_j - t_i, 1, [l_R, u_R]) \in \Upsilon_{arc}(i, j) : l_R \leq t_j - t_i \leq u_R \quad (3.51)$$

The path from t_i to t_j , i.e. $\theta_{[t_i, t_j]}$, is restricted by a set of domain constraints Ψ_{arc} :

$$\forall t_i \leq t \leq t_j \forall \tau = (e_1, \dots, e_m, f, R) \in \Psi_{node}(i) : f(e_1, \dots, e_m, \theta(t)) \in R \quad (3.52)$$

The planning model without domain node and arc constraints $(\mathcal{X}, \Upsilon_{node}, \emptyset, \Upsilon_{arc}, \emptyset, \theta)$ defines a temporal constraint satisfaction problem (TCSP) [29]. A solution to the problem is an assignment t_i of x_i , in which all temporal constraints are obeyed and Equations 3.49 and 3.51 hold. The latter defines a consistent assignment of the variables. Consequently, finding a solution is referred to as *consistency checking* in literature. Global search algorithms exist which incrementally test different variable assignments and prune the domains of unassigned variables based on (weak) consistency conditions. The domain of a variable is defined as the set of values, in which a solution will be searched. Initially, the domain contains all possible values of the variable. Some values will be inconsistent with other variable assignments due to violation of temporal constraints. The

domains decrease in size until either a solution is found or the domain is empty indicating that no solution to the TCSP exists. An important consistency condition is *arc consistency*, which ensures that the domains of a pair of variables are consistent. For each value in the domain of the first variable at least one value in the second domain can be found, for which all constraints are obeyed. The same applies to the domain of the second variable. AC-3 [95], see Algorithm 4, is an efficient algorithm to ensure arc-consistency. The function *Remove-Inconsistent-Values(i,j)* in line 4 depends on the domain of the variables and has to be specified by the user. It will be called iteratively to remove all values of the domain of x_i , which are inconsistent with all values in the domain of x_j . In this thesis, *Remove-Inconsistent-Values(i,j)* is defined for disjunctions of closed intervals.

In general, planning models contain both temporal and domain constraints and consistency has to be extended to the Equations 3.50 and 3.52 resulting in a temporal constraint satisfaction problem with domain constraints (TDCSP). The solution to a TDCSP will be discussed in the next subsection.

Algorithm 4 AC-3 algorithm for maintaining arc consistency in a CSP

```

1: queue  $\leftarrow$  Edges of csp
2: while queue  $\neq \emptyset$  do
3:   (i,j)  $\leftarrow$  Remove first element of queue
4:   if Remove-Inconsistent-Values(i,j) then
5:     for each neighbor k of i do
6:       Add (k,i) to queue
7:     end for
8:   end if
9:   if domain of i =  $\emptyset$  then
10:    return inconsistent
11:  end if
12: end while
13: return consistent

```

Solving the Temporal Constraint Satisfaction Problem

A solution of the planning model $(\mathcal{X}, \Upsilon_{node}, \Psi_{node}, \Upsilon_{arc}, \Psi_{arc}, \theta)$ is composed of the path θ and an assignment t_i of the variables x_i . If the assignment is known, the set of constraints, which have to be obeyed at time point t can be extracted:

$$\Psi(t) = \bigcup_{t_i=t} \Psi_{node}(i) \cup \bigcup_{t_i \leq t \leq t_j} \Psi_{arc}(i,j) \quad (3.53)$$

In this case, constraint-based motion planning can be applied to generate a path θ with $\theta(t) \in \mathcal{M}(\Psi(t))$ and a solution to the TDCSP is found.

If the assignment is unknown but the path θ is known, the TDCSP can be solved using AC-3 to generate a consistent assignment. In the function *Remove-Inconsistent-Values*(i,j), a temporal value t_i will be removed from the domain of x_i if no temporal value t_j in the domain of x_j exists, so that all constraint are obeyed on the path from t_i to t_j , i.e. $\exists t : \theta(t) \notin \mathcal{M}(\Psi(t))$.

The examples show that the problem to generate a path θ and an assignment t_i cannot be decoupled and has to be solved at the same time. We introduce an extended Rapidly-Exploring Random Tree (RRT) planner, which iteratively generates a disjunction of temporal constraints, which will be called set of restrictions, for each RRT node. The set of restrictions defines the domain \mathcal{D}_i of each variable x_i . It is ensured that the domains are consistent with the partially planned path $\theta'_{0 \leq t \leq t'}$, i.e. $\forall 0 \leq t \leq t' : \exists t_i \in \mathcal{D}_i : \theta'_{0 \leq t \leq t'}(t) \in \Psi(t)$. In the planning process, the set of restrictions will be used to generate random sets of constraints $\Psi(t)$, which are then used to extend partially planned paths towards the goal.

The following algorithms 5, 6, 7 and 8 were published by the author in a preliminary form with different representation in [51].

Algorithm 5 RRT($\theta_{start}, \theta_{target}$)

```

1: AddNode( $\theta_{start}, t_{start}, \Upsilon_{start}$ )
2: while true do
3:    $\theta_{random} \leftarrow$  RandomConfiguration( $\theta_{target}, 0.01$ )
4:   ( $\theta_{nn}, t_{nn}, \Upsilon_{nn}$ )  $\leftarrow$  NearestNeighbor( $\theta_{random}$ )
5:   ( $\theta_{extend}, t_{extend}$ )  $\leftarrow$  Extend( $\theta_{nn}, \theta_{random},$  Constraints( $\theta_{nn}, t_{nn}, \Upsilon_{nn}$ ))
6:    $\Upsilon_{extend} \leftarrow$  UpdateRestrictionsAC-3( $\theta_{extend}, t_{extend}, \Upsilon_{nn}$ )
7:   if  $\Upsilon_{extend} \neq \emptyset$  then
8:     AddNode( $\theta_{extend}, t_{extend}, \Upsilon_{extend}$ )
9:     AddParent( $\theta_{extend}, t_{extend}, \Upsilon_{extend}, \theta_{nn}, t_{nn}, \Upsilon_{nn}$ )
10:    if GoalTest( $\theta_{extend}, t_{extend}, \Upsilon_{extend}$ ) then
11:      return Path from ( $\theta_{start}, t_{start}, \Upsilon_{start}$ ) to ( $\theta_{extend} t_{extend}, \Upsilon_{extend}$ )
12:    end if
13:  end if
14: end while

```

Algorithm 5 shows an unidirectional RRT-planner with constraints extended by the temporal constraints Υ . For node i , $\Upsilon(i)$ is a set of temporal constraints represented by closed intervals. In the definition of planning models, only a single temporal constraint for each node is allowed. In order to solve the TDCSP, disjunctions of temporal constraints will be used to represent the domain of each variable x_i . In contrast to domain constraints, $\Upsilon(i)$ is interpreted as the disjunction and not the intersection of its elements.

First, the RRT is initialized with the start configuration and temporal constraints of each node as the set of restrictions, i.e. $\Upsilon(i)$ contains a single closed interval for each node i . Random configurations are drawn using the function *RandomConfiguration*. The nearest RRT node is searched using *NearestNeighbor*, which can be implemented as a linear search or kd-Tree [8]. The *Constraints* of the nearest RRT node are used to extend the tree in direction of the random

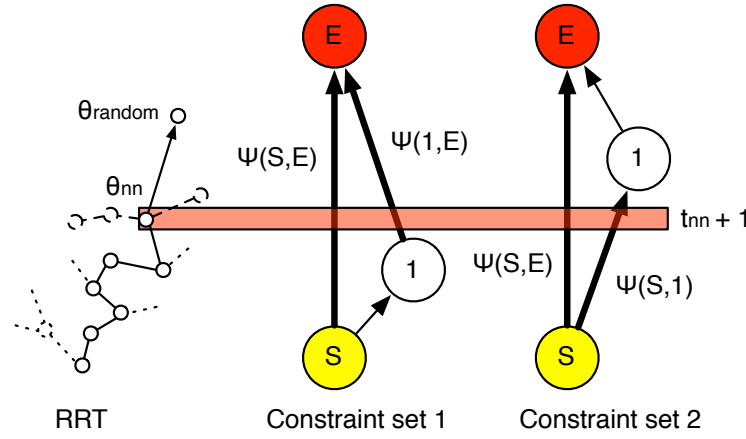


Figure 3.11.: A planning model with 3 nodes and 3 arcs is shown (middle, right). During planning a random configuration θ_{random} is chosen and the nearest neighbor θ_{nn} in the RRT calculated. Based on the restrictions of θ_{nn} and the time point $t_{nn} + 1$ different temporal orderings of x_i are possible, i.e. subgoal 1 has to be reached before or after $t_{nn} + 1$. Both possibilities lead to different sets of constraints (thick black arrows). One constraint set will be chosen randomly and a random configuration will be determined in the corresponding constraint manifold.

configuration. The set of restrictions is updated with the new time point and new configuration using AC-3. If the domains of each variable x_i are not empty, a new RRT node will be added to the tree. If the goal was reached, the path from the start RRT node to the current RRT node will be returned.

In each RRT node $(\theta_{node}, t_{node}, \Upsilon_{node})$ the set of restrictions defines the domain of each variable x_i implicitly. It is assumed that an assignment and a solution exist, which is consistent with the restrictions Υ_{node} and matches $\theta|_{0 \leq t \leq t_{node}}$.

Algorithm 6 RandomConfiguration($\theta_{target}, \epsilon$)

- 1: $u \sim \mathcal{U}(0, 1)$
 - 2: **if** $u \leq \epsilon$ **then**
 - 3: **return** θ_{target}
 - 4: **else**
 - 5: $\theta_{random} \sim \mathcal{U}(\Theta)$
 - 6: $(\theta_{nn}, t_{nn}, \Upsilon_{nn}) \leftarrow \text{NearestNeighbor}(\theta_{random})$
 - 7: **return** ConstrainedSample(Constraints($\theta_{nn}, t_{nn} + 1, \Upsilon_{nn}$))
 - 8: **end if**
-

Algorithm 6 returns a random configuration. In order to force the planner to work in direction of the goal the goal configuration will be returned with small probability. Otherwise, a random configuration in the full configuration space is drawn. The nearest neighbor in the RRT is calculated to estimate the time point of the random configuration. A random set of constraints, which covers the estimated time point, will be calculated and a random configuration will be drawn from the resulting constraint manifold, see Figure 3.11.

In contrast to standard constraint-based motion planning, the set of constraints is not fixed during planning but depends on the restrictions of the RRT node, which will be extended in the RRT. The

Algorithm 7 Constraints(θ, t, Υ)

```

1:  $E \leftarrow \emptyset$ 
2:  $(k_1, \dots, k_n) \sim \text{Permutation } \{1, \dots, |\text{Edges}|\}$ 
3: for  $i = 1$  to  $n$  do
4:    $m \leftarrow$  edge with index  $k_i$  in TCSP
5:   if  $\nexists e \in E: \text{Connected}(m.\text{to}, e.\text{from}) \vee \text{Connected}(e.\text{to}, m.\text{from})$  then
6:     for  $[a, b]$  in  $\Upsilon(m.\text{from})$  do
7:       if  $t \in [a, b + m.\text{end}]$  then
8:         add  $m$  to  $E$ 
9:       end if
10:    end for
11:  end if
12: end for
13: return  $\bigcup_{e \in E} \Psi_{\text{arc}}(e)$ 

```

goal of Algorithm 7 is to generate a random set of arcs, whichs overlap the time point of the new configuration. The algorithm processes the arcs in a random order. If the current arc does not have to be finished before or started after one of the already chosen arcs, it will be chosen. Finally, the union of all constraints $\Psi_{\text{arc}}(i, j)$ of all chosen arcs (i, j) is returned, see Figure 3.11.

Algorithm 8 Remove-Inconsistent-Values(e, Υ)

```

1:  $R \leftarrow \emptyset$ 
2: for  $[a, b]$  in  $\Upsilon(e.\text{from})$  do
3:   for  $[c, d]$  in  $\Upsilon(e.\text{to})$  do
4:      $[r, s] \leftarrow [c - e.\text{end}, d - e.\text{start}] \cap [a, b]$ 
5:     if  $q.\text{time} \notin [r, s + e.\text{end}] \vee e.\text{isValid}(q)$  then
6:       add  $[r, s]$  to  $R$ 
7:     else if  $q.\text{time} < s + e.\text{start} + 1$  then
8:       add  $[r, q.\text{time} - 1 - e.\text{start}] \cap [a, b]$  to  $R$ 
9:     if  $r < q.\text{time} + 1$  then
10:      add  $[q.\text{time} + 1, s] \cap [a, b]$  to  $R$ 
11:    end if
12:  end if
13: end for
14: end for
15:  $\Upsilon(e.\text{from}) \leftarrow \text{Simplify}(R)$ 
16: return  $\Upsilon(e.\text{from})$  has changed

```

Each RRT node contains not only the configuration and time but also the restrictions of the variables x_i . The restrictions of the root RRT node are $\Upsilon_{\text{node}}(i)$ for each x_i . If a new RRT node is added the set of restrictions has to be computed. Since a new RRT node has a unique parent, the restrictions of the parent are refined using AC-3. The restrictions define the set of consistent assignments to the variables x_i . If the assignment is known, the set of constraints, which have to be obeyed in the new configuration, can be computed. In reverse, if an arc constraint is violated in the new configuration, no consistent assignment exists in which the arc overlaps the new

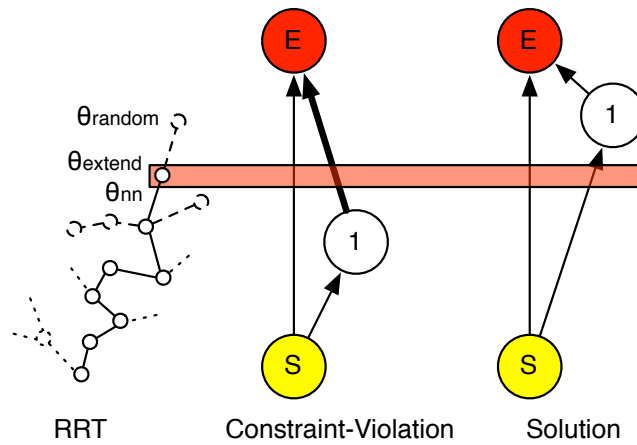


Figure 3.12.: A planning model with 3 nodes and 3 arcs is shown (middle, right). The left graph shows the RRT after adding a new node with configuration θ_{extend} . In order to update the restrictions of the new RRT node the set of arcs is generated, which potentially overlap the time point of the new RRT node. If one of the constraints of an arc is violated (see thick black arrow) the restrictions will be updated, so that the arc does not overlap anymore (right).

time point. The function *Remove-Inconsistent-Values* analyzes the restrictions of the parent and looks for arcs, in which a constraint is violated. In this case, the restrictions are refined to exclude assignments, in which the identified arc overlaps the new time point, see Figure 3.12. The implementation of *Remove-Inconsistent-Values* is shown in Algorithm 8. The algorithm cuts out parts of the restrictions resulting in a disjunction of closed intervals for each variable x_i .

The planning result is arc-consistent, which is only a weak consistency condition but can be maintained very efficiently during motion planning. The consistency check is implemented in the *GoalTest* function, which rejects paths for which no consistent assignment can be found.

3.2.3. Planning Model Linearization

In general, planning with temporal and domain constraints is time-demanding and can only be done for small numbers of subgoals¹, i.e. variables x_i . Since most manipulation tasks contain a larger number of subgoals, a simplification is necessary to plan manipulation tasks online in a short amount of time.

In Appendix B we discuss operators and metrics on strategy graphs, which allow to identify subgraphs of different strategy graphs, calculate the distance of an arbitrary trajectory to the constraint manifold defined by the strategy graph and linearize a given strategy graph.

The linearization operator transforms a planning model into a sequence of planning problems, where each problem depends on the previously solved problem, see Figure 3.13. The resulting, linear sequence of planning problems can be solved efficiently using backtracking.

¹The general planning problem is PSPACE-complete and consistency checking is NP-complete.

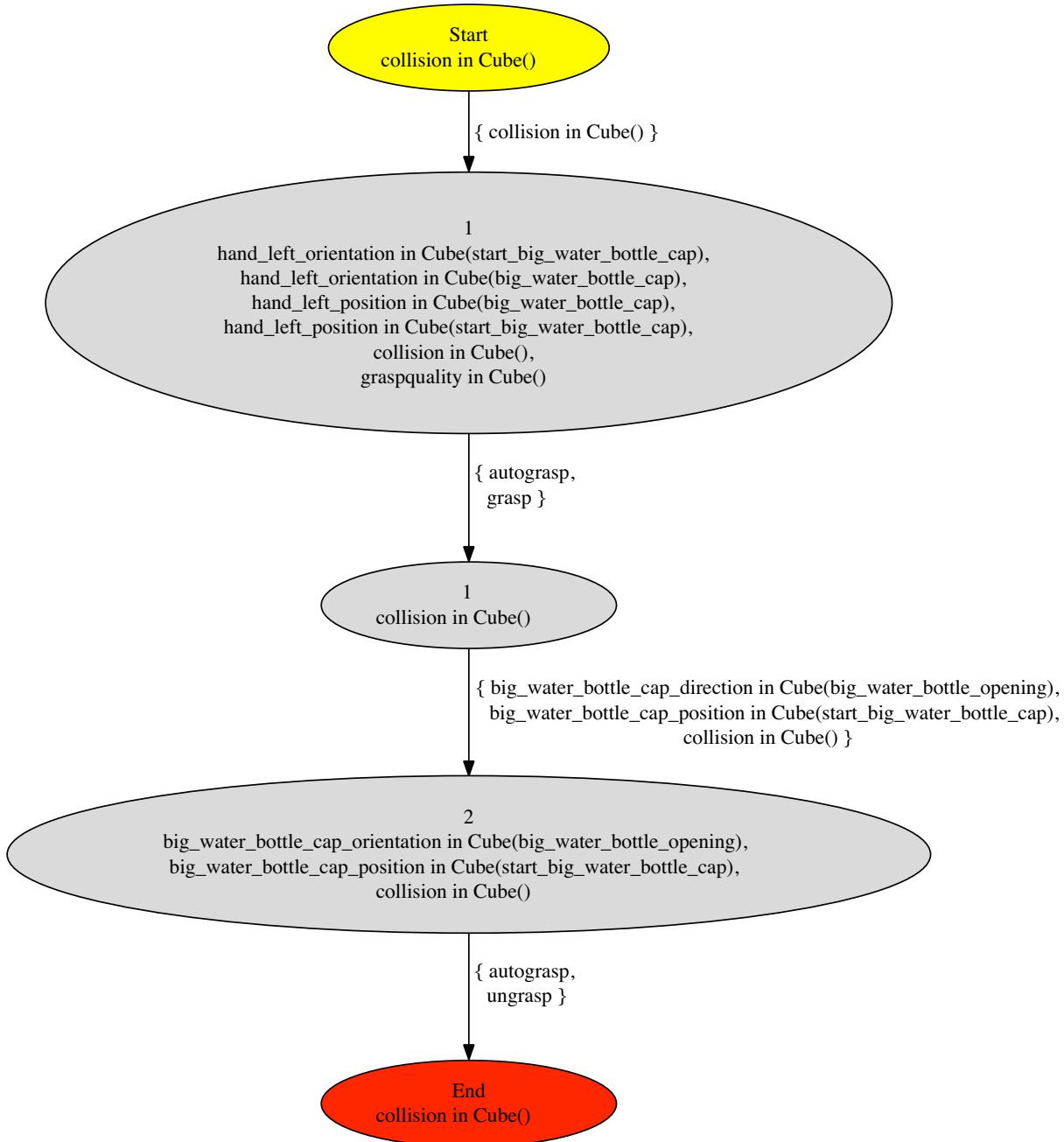


Figure 3.13.: Example of a linear planning model: opening a bottle

3.2.4. Discussion

Even simple manipulation tasks, e.g. grasping, can usually be partitioned into multiple segments, e.g. open the hand, approach the object, close the hand and apply force through the fingers. Therefore, it is natural to consider subgoals of the manipulation task, which have to be reached, and we represent them explicitly. We identify subgoals with a set of configurations, which have to be reached, leading to the use of a set of constraints to define goals. The robot motion between two subgoals might be constrained as well, e.g. holding a cup with water upright, leading to the inclusion of constraints between subgoals. In general, subgoals might be independent from each

other, e.g. opening the fridge with one hand and grasping a bottle with the other, leading to the inclusion of temporal constraints. If the planning process fails on a given segment it is possible to replan a previous segment recursively. In contrast to representations, which represent the whole manipulation task as a inseparable entity, e.g. as GMMs or DMPs, more complex tasks, i.e. with larger number of subgoals, can be planned. By using goals and constraints, the execution of the planned trajectory can be validated in the controller.

Solving the temporal constraint satisfaction problem has high complexity for complex manipulation tasks with multiple subgoals. Resulting planning times are unsuitable for online execution on the robot. We consider only linear planning models in the remaining part of the thesis. Learning, generalization and planning is therefore limited to manipulation tasks, which can be expressed by a linear sequence of subgoals.

3.3. Mapping of Planning Models

Planning models are defined based on a set of atomic constraints, which are grounded in predefined coordinate frames, e.g. the opening of a bottle, and online generated coordinate frames, for instance in a contact point. These coordinate frames are object- and task-dependent. Due to the large variety of objects and tasks in the human environment it is inefficient to learn planning models for each combination of objects and tasks. Naturally, we expect a robot system to be able to generalize learned planning models to similar objects. In other words to map constraints and thereby coordinate frames to novel objects.

Based on the assumption that predefined coordinate frames are defined for all objects in a given class, the problem reduces to the mapping of online generated coordinate frames to objects with similar geometry. Since coordinate frames are generated in contact points, we ground coordinate frames in local features of the object surface. Based on recent developments in the field of computer graphics, morphing algorithms are applied to find a global mapping of two object surfaces, which will be described in Section 3.3.1. The global mapping allows to instantiate the local features on the novel object surface and reconstruct the coordinate frame, see Section 3.3.2.

3.3.1. Morphing of Geometric Object Models

In computer graphics, a well-studied problem is the morphing of one 3D model into a second 3D model with applications in different domains, e.g. animation or model-fitting. A simple but efficient way to morph two 3D models is to define a mapping between the points on the surfaces, usually the vertices of triangular meshes, and interpolate the motion between two correspondences. The definition of a surface mapping, which maps points in a semantically comprehensible way, e.g. the opening of a small bottle to a larger opening of a different bottle or the center of the surface of a square to a round button, is demanding.

Kim et al. define blended intrinsic maps [62] based on a set of extreme points, e.g. points with high curvature, which are identified efficiently on both surfaces. The surface mapping is defined

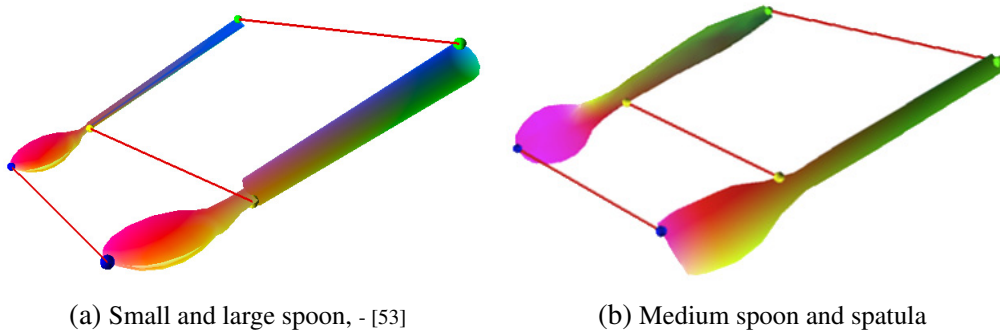


Figure 3.14.: Examples of blended intrinsic maps. Point correspondences on both surfaces are visualized by similar colors.

based on the mapping of extreme points, which produces semantically comprehensible results for articulated bodies and geometries with pointy structures. In Figure 3.14, the blended intrinsic maps of different object pairs are shown.

In order to instantiate a planning model with a novel 3D model, a blended intrinsic map is calculated for the known and the novel 3D model. First, the 3D model with fewer vertices is interpolated to generate models with the same number of vertices. In order to reduce the influence of ambiguities, the 3D model is translated and rotated to align the boundary boxes of both 3D models. Finally, the blended intrinsic map is calculated.

3.3.2. Coordinate Frame Mapping

Based on the surface mapping, we define the mapping of coordinate frames [53]. Let oH_f be the coordinate frame on the surface of object o , which will be mapped to object o' . The general approach is

1. Calculate local features of coordinate frame oH_f of object o
2. Calculate global surface mapping between object o and o'
3. Map local features using surface mapping to object o'
4. Reconstruct coordinate frame ${}^{o'}H_{f'}$ using mapped local features

We define the global surface mapping as the blended intrinsic map between o and o' . The local features consist of a fixed homogenous transformation matrix and the vertices of the closest face to the origin of coordinate frame oH_f . The local features are calculated in several steps. First, the closest face on object o to the origin of the coordinate frame oH_f is calculated. We define the distance of a point to a face as the distance to the closest point inside the face [39, p. 145f]. On the closest face we define a coordinate system. The origin is the closest point inside the face to oH_f . The x-axis points towards the first vertex and the z-axis points towards the face normal. We store

the homogenous transformation matrix from this coordinate frame to oH_f and the set of vertices as local features.

In the next step, the blended intrinsic map is used to map the feature vertices to the object o' . The properties of the blended intrinsic map ensure that the mapped vertices form a face on o' . We define a coordinate frame in the same way inside the mapped face and use the feature homogenous transformation to calculate ${}^{o'}H_{f'}$.

The defined local features are simple since, from a local point of view, a semantically-comprehensible surface mapping is hard to obtain. Therefore, we rely heavily on the global surface mapping between objects and benefit directly from new developments in computer graphics.

3.3.3. Discussion

Mapping of learned constraints to novel objects is difficult, e.g. Song et al. [98] showed that features like object convexity and size, grasp stability or the used preshape have an influence on choice of a grasp configuration. We make the assumption that the mapping of the coordinate frame, which is referenced by the constraint, is sufficient to transform a constraint to a visually similar object. The constraint region is not transformed. The assumption is usually sufficient for contact coordinate frames. The constraint region represents the region, in which a fingertip or wrist have to be placed. In combination with contact and force constraints, the robot motion can be adapted to small changes in the object surface, e.g. a contact point on a larger or smaller bottle opening. Since the goal of the task usually depends on the induced object motion, the planning process will be successful if the mapped constraint manifold contains at least one valid configuration. This might be different for non-contact coordinate frames, e.g. the constraint region, which restricts the position of a bottle opening above a cup opening, has to be transformed if the size of the cup opening changes to guarantee that no fluids are spilled. An automatic validation of the mapping process is not possible since it affects the constraints, which are the basis to validate the execution. We consider this limitation by supervising the mapping process, interactively adapting a constraint region or demonstrating robust solutions, e.g. placing the bottle opening near the center of the cup opening, which generalizes to different cup sizes.

3.4. Constraint-based Motion Planning

In real-world tasks with multiple sub-goals and a large number of constraints the temporal constraint satisfaction problem with domain constraints cannot be solved efficiently. The main problems are that temporal constraints induce a NP-complete constraint satisfaction problem and finding a robot trajectory, on which a set of domain constraints is obeyed, is in PSPACE. The first problem can be solved using our AC-3 approach in Section 3.2.2 but the second problem requires more advanced planning techniques, which are difficult to integrate into the AC-3 approach. Since most manipulation tasks are linear or can be linearized efficiently, see Section 3.2.3, we concentrate on the second problem and work only with linear planning models.

In order to generate a robot motion efficiently, which is consistent with the learned set of task constraints, the unidirectional RRT with constraints is insufficient since it lacks important enhancement from state-of-the-art constraint-based motion planners. We extend the planning algorithm to be bidirectional [66], which improves performance in manipulation tasks, where either the start or goal are heavily restricted, e.g. by a narrow passage. Rejection sampling, which is heavily used in the unidirectional RRT, is problematic for thin constraint manifolds. Thin constraint manifolds occur naturally in manipulation tasks, e.g. when pushing an object across a surface, the object motion will be in a 2D-plane. In the unidirectional RRT, configurations will be sampled randomly and rejected if a constraint is violated. Since the probability is 0 to sample a configuration on an infinitely thin constraint manifold, no solution will be found. We integrate projection techniques from [101] to project a configuration to a nearby configuration, in which position, orientation and direction constraints are obeyed, see Section 3.4.1. Additionally, we define special projection techniques for contact, see Section 3.4.1, and force constraints, see Section 3.4.1.

Projection techniques are usually local optimization algorithms to enable short planning times. In [10], projection techniques are also used to generate goal configurations, i.e. configuration, in which all constraints of the current (sub-)goal are obeyed. Using a random configuration as starting point for the projection techniques is inefficient for complex constraint manifolds. We introduce an additional step, in which constraints are sampled and inverse kinematics are applied to generate a configuration, in which a subset of constraints is already obeyed, as starting point for constraint projection. The approach will be described in Section 3.4.2.

3.4.1. Projection Techniques

Complex constraint manifolds restrict different task aspects, e.g. relative positions or contacts between different objects, at the same time. They can be defined efficiently as the intersection of a number of simpler constraints, each restricting only a single task aspect. Each constraint allows to sample configurations, in which the constraint is obeyed. Sampling a configuration, in which a number of constraints is obeyed, is more complicated. Rejection sampling is prohibitive since constraint manifolds can be arbitrarily thin. We apply different techniques to project a given configuration to the constraint manifold: constraint, contact and force projection.

Constraint Projection

Stilman et al. [101] integrate Randomized Gradient Descent (RGD) into the RRT algorithm. RGD is a stochastic gradient descent method, which samples a direction in the configuration space randomly and moves in this direction if the distance to the constraint manifold is smaller, see Figure 3.15 and Section 2.1.4. In contrast to Jacobian-based approaches, e.g. Fraction-Retraction [101] or Tangent-Space-Sampling [101], no distance vector is required since RGD approximates the distance vector online based on the distance to the constraint manifold. This allows to consider arbitrary constraints at the cost of higher execution time. An example is the grasp quality constraint,

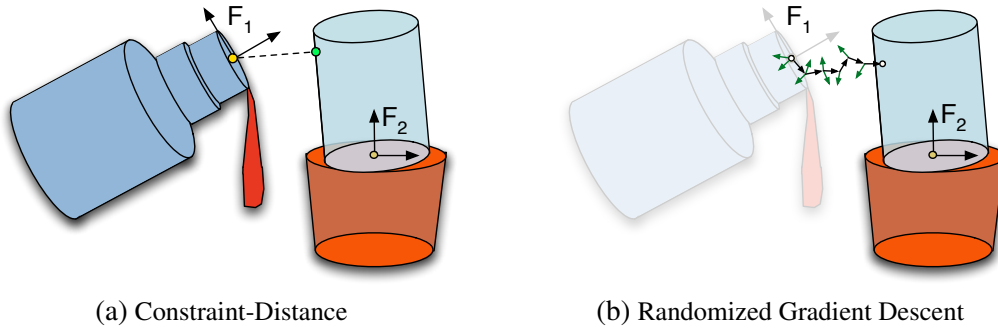


Figure 3.15.: Constraint Projection: Randomized Gradient Descent

which allows to compute a distance, e.g. the grasp quality in the current configuration is 0.1 below the minimum grasp quality, but not a distance vector, e.g. how to adapt the current configuration to improve the grasp quality. We apply Randomized Gradient Descent to project position, orientation, direction and arbitrary constraints.

Contact Projection

In general, contact constraints define a thin constraint manifold and it is impossible to find a configuration, in which the robot has a specific contact with an object, using rejection sampling. In a random configuration, two different situations occur for a contact constraint. Either the referenced 3D bodies are in collision, see Figure 3.16a, or no contact exists, see Figure 3.16b. If a collision exists, the joint values have to be adapted (in a minimal way) so that no collision exist and the distance between the two bodies is below the contact threshold. Since the bodies might belong to a kinematic chain, a complicated problem arises. We apply the Penetration Depth algorithm [56] to solve the problem iteratively using the penetration depth, i.e. a metric how much both 3D bodies intersect. In the second situation, the contact has to be established. This problem is known from grasp planning literature and is solved in this work using continuous collision checking C2A [105] based on the distance between two 3D bodies.

Force Projection

We use force constraints in combination with contact constraints to model finger tip manipulation tasks. Contact projection ensures that the two referenced 3D bodies are in contact. The force constraint models the actual amount of force, which has to be applied to the object. In physics simulation and on the real robot we will use a simple position controller to execute planned robot trajectories. Therefore, the force has to be translated into a motion of the robot resulting in another projection step. For a force constraint $(e_1, e_2, e_3, f_{for}, R)$ we draw a random sample $x \sim R$ and calculate

$$({}^0Rot_{e_1}, {}^0t_{e_1}) = f_{for}^{-1}(x, e_2, e_3, \theta) \quad (3.54)$$

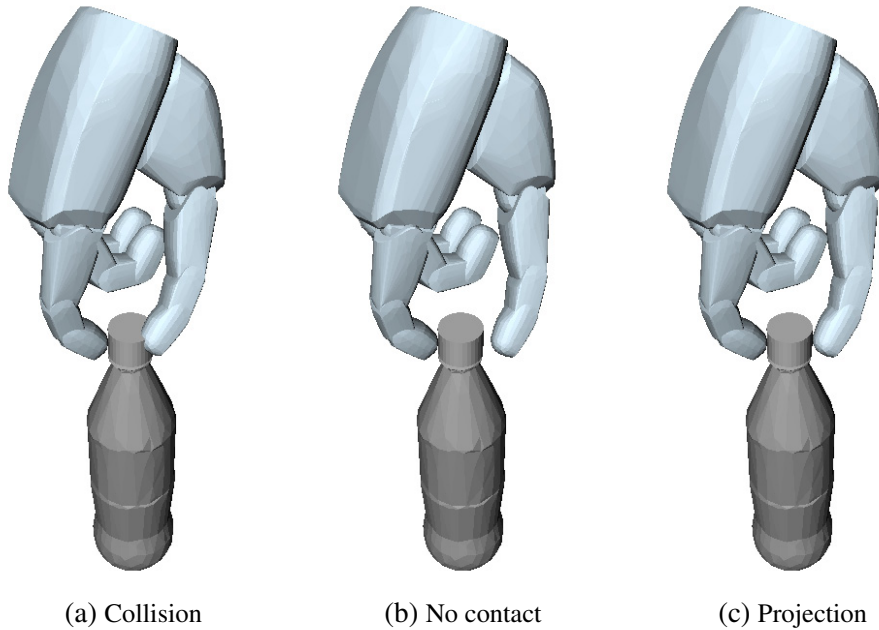


Figure 3.16.: Contact projection: different start situations and projection result.

${}^0t_{e_1}$ is the force in global coordinates. Based on the dynamic model of the robot, we calculate the corresponding joint displacement. The displacement is added to the configuration resulting in the projected configuration.

3.4.2. Sampling of Goal Configurations

In the unidirectional RRT, a path is planned beginning at the start configuration. The RRT succeeds if the last configuration lies on the goal constraint manifold. Since the probability to find a configuration, which lies on a thin goal constraint manifold is ~ 0 , an advanced RRT planner has to be applied, which allows to plan in a more goal-directed way. Before planning starts, we generate a set of configurations, which lie on the goal constraint manifold, the so-called *goal configurations*. In [10], a configuration is sampled randomly and projected to the goal constraint manifold using RGD. Since RGD is a local optimization technique, the result depends heavily on the start configuration. Additionally, only configurations on the border of the constraint manifold are generated. In contrast to this work, we do not generate a goal configuration with a uniform distribution on the configuration space but sample it from a subset of goal constraints. It is automatically guaranteed that the goal configuration obeys all constraints in the subset. Finally, we project the configuration to the constraint manifold using the above projection techniques.

Figure 3.17 shows an example. The goal is to place the grasped object *Box* on the table. Two goal position constraints and one goal orientation constraint exist. The first position constraint restricts the *Box* to be closer than 260mm to the *Plate* (green). At the same time, the second position constraint ensures that the *Box* is closer than 200mm to the *Cereals* (red). Both constraints are represented by a cylindrical constraint region. The intersection of both constraints is a small

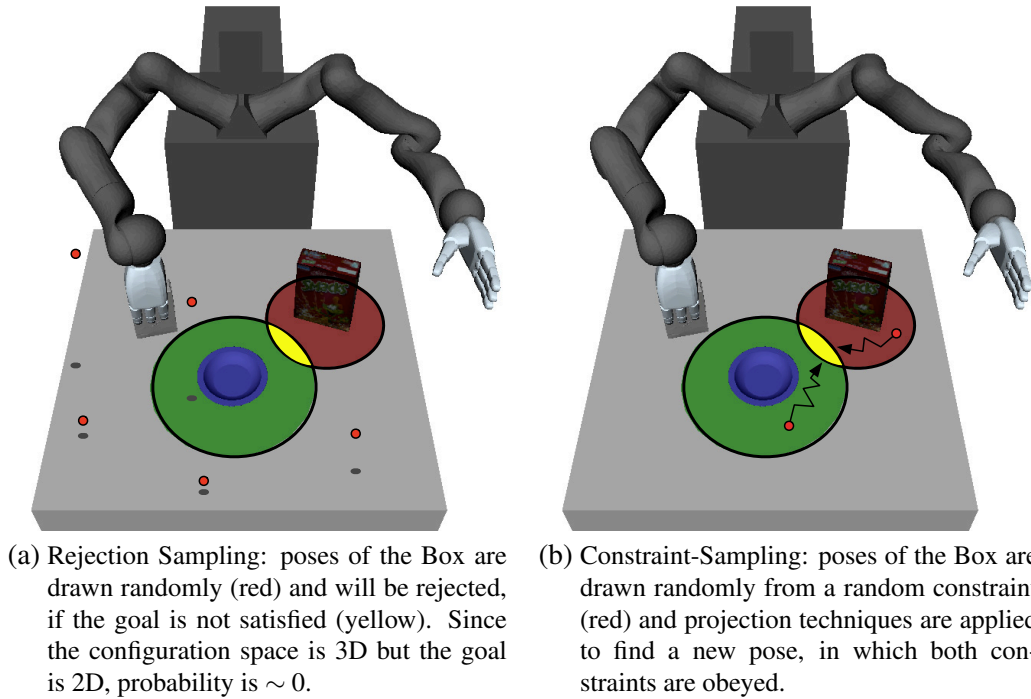


Figure 3.17.: Rejection- and Constraint-Sampling. The goal of the task is defined by two constraints: (Box, Plate, f_{pos} , Cylinder(0,360,0,260,0,0.1)) and (Box, Cereals, f_{pos} , Cylinder(0,360,0,200,0,0.1)). The grasped object Box has to be placed in two 0.1mm-thin cylinders centered on the objects Plate and Cereals.

region on the table (yellow). The goal is to find a configuration, i.e. joint values of the robot arm, so that the grasped object is placed in the intersection. Rejection sampling, see Figure 3.17a, generates a random configuration resulting in a random 6D pose of the arm. Since the table surface is 2D, probability is ~ 0 to find a a valid goal configuration. In this thesis, a different approach is followed. A random subset of constraints is chosen and inverse kinematics are applied to generate a configuration, in which the subset of constraints is obeyed. In this case, two random subsets are considered. The first set consists of the first position constraint and the orientation constraint. The second set contains the second position constraint and the orientation constraint. The generated configuration will be in the green, red or yellow area on the table. Finally, projection techniques are applied resulting in a configuration, in which all constraints are obeyed, see Figure 3.17b.

In Section 3.1.2, a constraint sampling set Ψ_S for the set of constraints Ψ was defined representing a set of pairs of constraints, where each pair restricts the position and orientation or direction of a 3D rigid body. In order to generate a goal configuration, a constraint sampling set is chosen randomly. The constraints in the constraint sampling set are ordered based on the coordinate frame dependency, i.e. constraint A will be before constraint B, if the constrained coordinate frame of A is relative to one of the reference coordinate frames of B. For example, A restricts the spoon relative to the table and B restricts the fingertip relative to a spoon. In order to generate a goal configuration, A will be sampled to generate a random spoon pose. Since the fingertip position depends on the spoon pose, B will be sampled after A to generate a valid goal configuration.

For each pair of constraints a random pose, i.e. a random position and a random orientation, is generated, in which the constraints are obeyed. Based on the random pose, we apply inverse kinematics (IK) to generate the corresponding joint angles. In order to treat robots with multiple kinematic chains and free-floating objects, e.g. manipulated objects on the table, in a consistent way, free-floating objects are considered holonom robots with six cartesian degrees-of-freedom. If a contact or force constraint restricts the same object, contact projection is applied.

We accumulate the IK results into a single configuration. The configuration is projected using constraint projection. If all constraints are obeyed, the configuration is added to the set of goal configurations.

Figure 3.18 shows an example of the generation of a random goal configuration. A random position and orientation or direction constraint is drawn randomly for each 3D rigid body: spatula, fingertip and wrist. The fingertip is relative to the wrist and the wrist relative to the spatula. Due to this dependency, the pose of the spatula is drawn randomly, first. Contact projection moves it in opposite direction to gravity until contact is made with the table. The wrist and finger pose is sampled and joint values are generated. Contact projection is applied to move the fingertip down until contact. All constraints are obeyed.

The random set of goal configurations is the basis to plan a robot motion in a goal-directed way, which will be explained in the next section.

3.4.3. Goal-directed Planning

Constraint-based motion planning is applied to generate a smooth path from the start configuration to one of the goal configurations. Manipulation tasks require the coordinated movement of robot arms and fingers. Similar to human morphology service robot arms usually consists of 6 or 7 degrees of freedom. Anthropomorphic hands require at least 3 DOF per finger resulting in roughly 20 DOF for a human-like robot arm with four articulated fingers. The resulting planning problem is high-dimensional. In this context, sampling-based motion planners offer the best performance, e.g. RRTs or PRMs, see Section 2.1. In household tasks the robot faces a large number of obstacles and objects and cannot position itself always in the same way relative to the environment. From a planning point of view, the environment changes all the time and precalculated roadmaps, e.g. PRMs, cannot be reused. Based on this observation, we employ the Constrained Bidirectional RRT (CBiRRT) by Berenson et al. [9], which grows one RRT at the start configuration and one RRT in each goal configuration and constantly tries to connect both to solve the planning problem. Algorithm 9 shows the algorithm by Berenson. We apply the original algorithm and only integrate the additional projection techniques into $Extend(\mathcal{T}, \theta_{nn}, \theta_{rnd})$, i.e. first we generate the configuration

$$\theta_{new} = \theta_{nn} + \frac{\epsilon}{|\theta_{rnd} - \theta_{nn}|}(\theta_{rnd} - \theta_{nn}), \quad (3.55)$$

then we apply constraint projection followed by contact projection to θ_{new} and return the projected configuration. The function *RandConfig* returns a uniform random configuration in

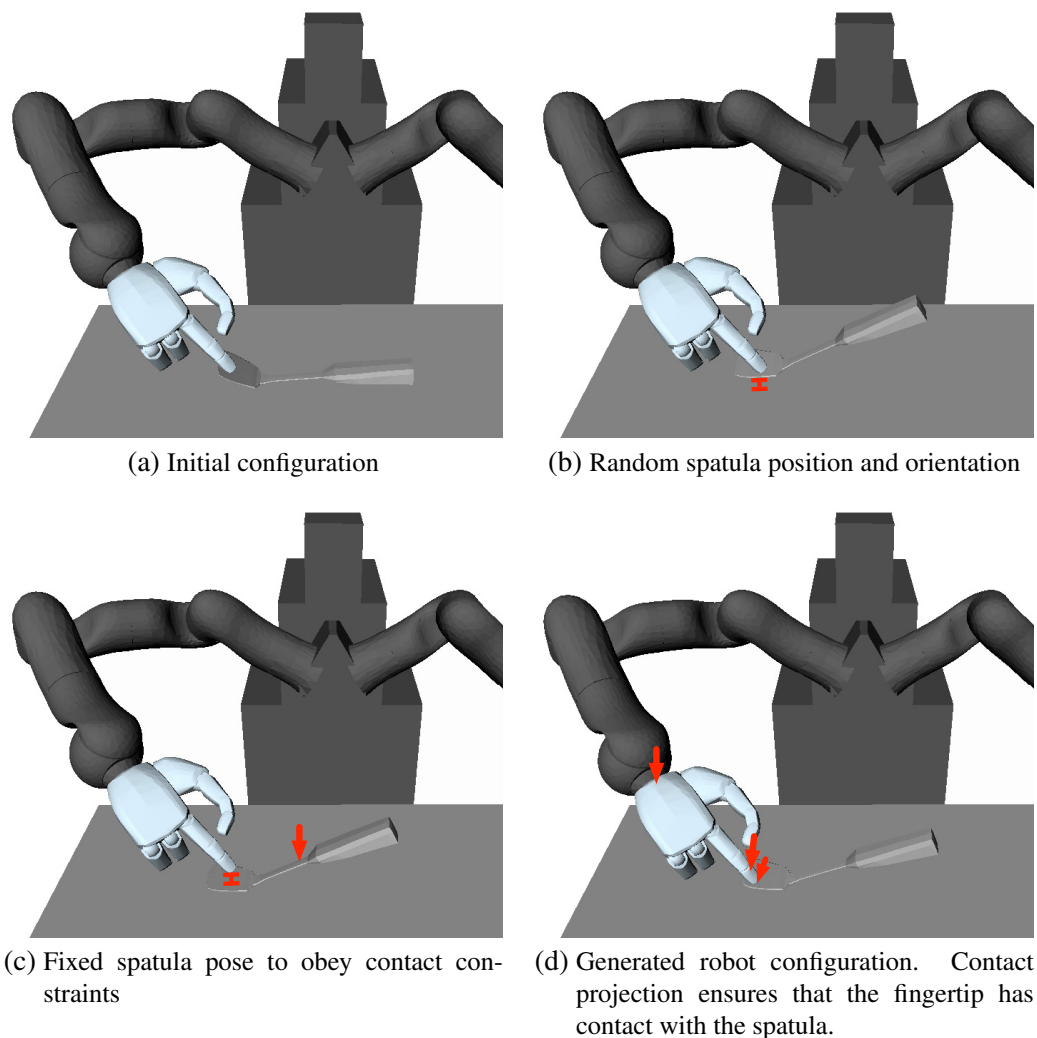


Figure 3.18.: Sampling of goal configurations: pushing with the index finger on a spatula to lift the handle and grasp it. Arrows emphasize the changes in each step. (a) initial configuration: fingertip placed on spoon. (b) a random position and orientation of the spoon is sampled. (c) contact projection is applied to establish the contact between the spoon and the table. (d) the wrist pose and finger tip pose is sampled based on the random spoon pose. Inverse kinematics are applied to generate a robot configuration. Contact projection is applied to move the fingertip down until contact.

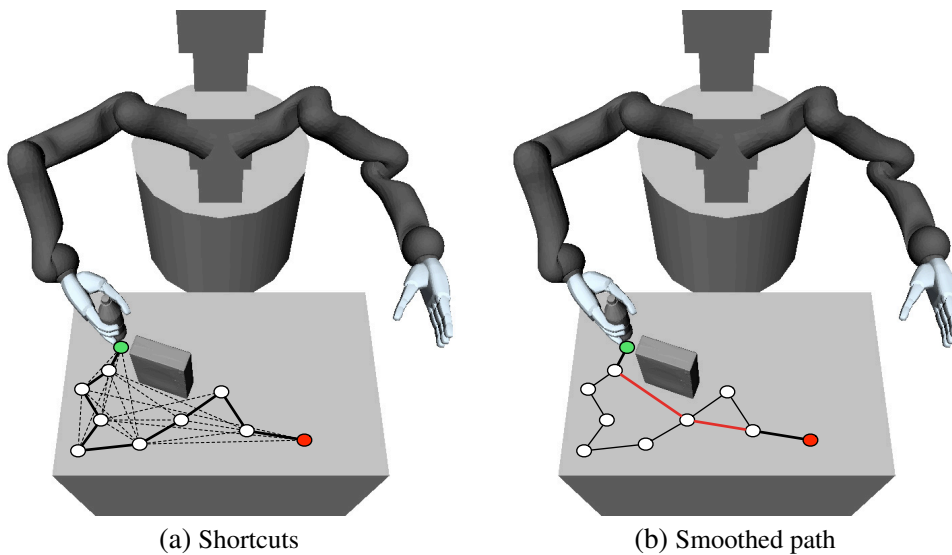


Figure 3.19.: Smoothing of a planned path (solid lines): (a) shortcuts (dotted lines) are generated between pairs of configurations, on which all constraints, e.g. no collision, are obeyed. (b) the smoothed path (red lines) is shorter than the original path.

the configuration space Θ . We calculate the nearest configuration to θ in the tree \mathcal{T} using $NearestNeighbor(\mathcal{T}, \theta)$, which is implemented as a linear search. The function $Swap$ switches both trees, i.e. in the next iteration, the other tree will be extended. The solution path is calculated by following the parent links in both trees using the last added configurations, see function $ExtractPath$. The goal of $SmoothPath$ [9] is to shorten the solution path by finding valid shortcuts between configurations on the path. A shortcut will be valid if all constraints are obeyed on the direct path, which connects both configurations, see Figure 3.19.

In combination with projection techniques, CBiRRT [9] allows to plan manipulation tasks in high-dimensional spaces. The planning time is usually high and the algorithm cannot be applied online. In the next chapter, we look at learning techniques to generate planning models from a set of explicit human demonstrations. The planning model captures not only goals and task-relevant constraints but also constraints for the robot arms and hands, which efficiently restrict the search space for planning and leads to shorter planning times.

3.4.4. Planning with Dynamics Simulation

In dexterous manipulation tasks the robot manipulates an object often with its fingertips without grasping the object firmly. In order to simulate the effect of the forces, which are applied through the fingertips, we include the dynamics simulation ODE using the robot simulation framework OpenRAVE [32]. In the motion planning process, we extend the RRT node and store the state of the dynamics simulation, i.e. poses and velocities of all objects, joint values and joint velocities. In order to calculate the forces, which are generated by the robot motion, we implement a positional controller in ODE, which mimics the controller used on the real robot system. We do not assume

Algorithm 9 CBiRRT($\theta_{start}, \{\theta_{goal}\}, \Psi$), [9]

```

 $\mathcal{T}_a$ .init( $\{\theta_{start}\}$ );  $\mathcal{T}_b$ .init( $\{\theta_{goal}\}$ );
while not TIME_LIMIT() do
   $\theta_{rnd} \leftarrow \text{RandConfig}()$ ;
   $\theta_{nn}^a \leftarrow \text{NearestNeighbor}(\mathcal{T}_a, \theta_{rnd})$ ;
   $\theta_{reached}^a \leftarrow \text{Extend}(\mathcal{T}_a, \theta_{nn}^a, \theta_{rnd})$ 
   $\theta_{nn}^b \leftarrow \text{NearestNeighbor}(\mathcal{T}_b, \theta_{reached}^a)$ ;
   $\theta_{reached}^b \leftarrow \text{Extend}(\mathcal{T}_b, \theta_{nn}^b, \theta_{reached}^a)$ 
  if  $\theta_{reached}^a = \theta_{reached}^b$  then
     $P \leftarrow \text{ExtractPath}(\mathcal{T}_a, \theta_{reached}^a, \mathcal{T}_b, \theta_{reached}^b)$ ;
    return SmoothPath( $P$ )
  else
    Swap( $\mathcal{T}_a, \mathcal{T}_b$ );
  end if
end while
return NULL;

```

that the controller can be temporally inverted. The consequence is that for dexterous manipulation tasks we cannot plan in a bidirectional way, i.e. both from the start and the goal configurations. In this case, we use the unidirectional pendant to CBiRRT [9].

The dynamics simulation is integrated into the motion planner in the function *Extend*, see Algorithm 9. Given a random configuration, the *Extend* function is used to extend the nearest configuration in the RRT in the direction of the random configuration. In this step, we load the state of the dynamics simulation of the nearest neighbor configuration θ_{nn} and use the controller to execute the projected configuration θ_{new} in the dynamics simulation. We retrieve information about the resulting object poses and robot configuration from the dynamics simulation to check, if all constraints are obeyed.

3.4.5. Discussion

State-of-the-art sampling-based motion planners allow to plan robot motions on constraint manifolds in high dimensional spaces. General purpose projection techniques like RGD rely on a representation of the constraint manifold, which allows to compute inclusion and distance of a configuration to its border efficiently. We considered these requirements in the definition of constraint regions allowing to consider arbitrary position, direction and orientation constraints during motion planning. The computational effort is low since inclusion and distance calculations depend only on coordinate transformations. Contact and collision constraints depend heavily on object geometry resulting in high computational effort to calculate inclusion and distance. For this purpose, we introduced algorithms from computer vision resulting in special projection algorithms, which were combined with the general purpose projection techniques.

A severe limitation of sampling-based motion planners is the randomness of the generated result. The consequence is that the generated paths are neither smooth nor optimal. We apply post-

processing to smooth and shorten the paths using [9]. The technique can be parallelized efficiently [70], which was not considered in this thesis. A limitation of planning with dynamics simulation is that the unsmoothed path is executed implicitly during planning. The unsmooth motion leads to unnecessary contacts with the objects resulting in auxiliary forces, which have to be compensated by the planner. Empirically, this increases the planning time and failure rate. More research in the domain of planning with dynamics simulation is necessary to overcome this limitation.

3.5. Summary and Conclusion

Constraint-based programming allows to model and execute manipulation tasks on control level. In order to generalize to different environments, not only constraints but also (future) goals have to be considered, e.g. how to place the robot hand above a bottle to be able to rotate the cap, which is not considered on control level. In constraint-based motion planning goals and constraints are defined in a unified way as constraint manifolds, i.e. subsets of the search space for motion planning, but the constraint representations are limited. Based on this observation, we defined a constraint-framework consisting of position, orientation, direction, force, moment, contact, configuration, temporal and additional constraints like grasp quality metrics. Each constraint consists of a constraint function, whose values are restricted by the constraint region.

In PbD, learning is often restricted to a certain set of constraints in a predefined feature space. Manipulation tasks usually do not contain a single set of constraints but multiple goals and constraint sets, which change over time. This observation led to the definition of strategy graphs, in which nodes define (sub-)goals of the task and arcs define the constraints between two (sub-)goals. A strategy graph represents a planning model, which will be used by a constraint-based motion planner to generate a robot motion for a given task. In general, the temporal ordering of nodes is not predefined but restricted in a flexible way by a set of temporal constraints. In the planning process, we have to find an ordering of nodes and generate a robot trajectory, on which all constraints are obeyed and all subgoals are reached. Therefore, the planning model defines a temporal constraint satisfaction problem. We integrated the AC-3 algorithm into the Rapidly Exploring Random Tree motion planner to solve the temporal constraint satisfaction problem. The complexity is usually too high to solve the planning problem online. To overcome this limitation, planning models are linearized resulting in a sequence of planning problems.

Position, orientation, direction, force, moment and contact constraints are defined based on a set of coordinate frames. In new environments, the coordinate frames and therefore the constraints adapt automatically to different start configurations, object and obstacle poses. By using constraint-based motion planning, a planning model is executed in a flexible way in these environments. In order to execute a planning model with different objects, we rely on the existing correspondence of coordinate frames between different objects, e.g. of the coordinate frames *cup bottom* and *glass bottom*. In the learning process, coordinate frames in contact points, e.g. where a finger touches an object, will be generated automatically. In order to map an automatically gen-

erated coordinate frame to a new object, we calculate local features on the object surface, map the local features to the new object using the morphing algorithm blended intrinsic maps and create the mapped coordinate frame based on the transformed, local features.

In order to plan efficiently on constraint manifolds, projection techniques are used to project an arbitrary configuration to the constraint manifold. General purpose projection techniques like Randomized Gradient Descent require only the definition of a distance function. Complexity can be very high, e.g. to find a hand configuration, in which multiple fingertips are in contact with an object. We applied the Penetration Depth algorithm and the continuous collision checker C2A to project a configuration based on contact constraints efficiently. Additionally, we defined a force projection to consider force constraints.

In the planning process, we generate goal configurations randomly. A set of constraint pairs is drawn randomly, where each pair restricts the position and orientation of an object. Based on each pair, a random object pose is drawn and inverse kinematics are applied to generate a robot configuration, in which the subset of constraints is obeyed. By using the defined projection techniques, a random goal configuration, in which all constraints are obeyed, is generated.

If a planning model contains force constraints, it is necessary to simulate the force interaction between the robot and the environment. In this case, we apply dynamics simulation in the constraint-based motion planner. The main limitations are that accurate dynamic models are required and a tradeoff between accuracy and computation time has to be found.

Conclusion Manipulation tasks are represented on the basis of constraints, which model task constraints and goals in a unified way. The set of constraints, which can be defined based on a set of coordinate frames, is polynomial. Each constraint requires multiple parameters and dependencies between constraints have to be considered, e.g. to restrict the wrist pose in a way that the fingertips reach a certain point. For complex manipulation tasks, it is difficult to define a strategy graph manually. In the next chapter, we make use of the human as an expert in manipulation to make illustrative examples of the manipulation task and learn a planning model, i.e. structure and constraints, automatically.

Based on the constraints in a planning model and result of the planning process, we can decide prior to the execution if the planning model can be executed in the given environment or not, which is an advantage compared to control-based approaches. If the execution is not possible, the set of constraints allows to localize the cause of failure, e.g. which constraints could not be obeyed at the same time. The latter will be used in Chapter 4 to identify irrelevant constraints and increase flexibility of the learned planning model.

A second advantage of the constraint-based representation, e.g. compared to probability distributions in a latent space, is that constraints can be relaxed and tightened easily by changing the parameters of the constraint regions. In the next chapter, this property will help us to solve the correspondence problem, i.e. how to automatically consider the differences in morphology between the human and the robot.

4. Programming by Demonstration of Planning Models for Manipulation Tasks

The automation of everyday manipulation tasks in human-centered environments with different tasks, objects and obstacles demands high flexibility from a service robot. A prerequisite is a sophisticated model of the task in which all relevant constraints and goals are encoded. Based on this model, the robot can automatically generate a robot motion in different environments and monitor the execution, e.g. to ensure that the goal of the task was reached. In the previous chapter, we defined planning models, which represent a sophisticated task model based on position, orientation, direction, force, moment, contact and temporal constraints suitable for constraint-based motion planning. The manual definition of such a planning model is time-demanding, error-prone and requires in-depth knowledge about the planning algorithm. The main reason is that a large number of constraints, usually more than 30, have to be defined to model even simple manipulation tasks. Additionally, constraints describe not only the motion of objects but also of the robot. The programmer has to define the constraints as flexible as possible while ensuring that a certain task goal is reached. For example, it is difficult to define constraints, which result in the coordinated motion of the robot hand and fingers necessary to rotate a bottle cap.

In a typical household scenario, the programmer is a non-expert and the question is how the task model, i.e. in our case the planning model, can be defined in a more flexible and natural way. Although the human is not an expert in programming, he is definitely an expert in manipulation. From early childhood on we constantly manipulate our environment. First starting with simple tasks like punching or squeezing, advancing to everyday tasks like pouring in and, in some cases, reaching the ability to execute complex manipulation tasks like repairing a leaking oil pump or assembling a car engine. Additionally, we are trained to guide other people with complex tasks and show them necessary manipulation tasks, e.g. how to use a corkscrew to open a wine bottle, in an efficient way. In this process, the human abstracts automatically from the large set of different instances of the manipulation task and concentrates his explanations only on the important aspects of the manipulation task.

A more flexible and natural way to program a robot can thus be achieved by letting the human demonstrate a manipulation task with his own hands in the real environment. The robot actively observes these demonstrations and tries to deduce a planning model, which captures all relevant constraints and goals of the task. By choosing illustrative examples and making pauses, which implicitly provide a segmentation of the sensor data stream, the human can effectively ease the learning problem and use his intellect to steer the learning process. Flexible means to distinguish between important and irrelevant aspects of a task are necessary. Due to the number of constraints,

the lacking symbolic interpretation and lacking knowledge of the human, it is not possible to explicitly decide for each constraint, if it is important or not. Instead, we investigate two different approaches to identify irrelevant constraints.

In the first approach, so-called *demonstration-based generalization*, the human sorts the demonstrations into two sets of different difficulty. The initial planning model is learned based on the less difficult one and generalized by removing constraints, which are inconsistent with the more difficult examples. Based on this relatively simple concept, arbitrary sets of irrelevant constraints can be removed resulting in a high flexibility. The main disadvantages are that a large number of demonstrations is required and that the teacher requires a lot of experience.

In the second approach, so-called *robot-test-based generalization*, we investigate a different, more automatic approach. Based on a set of test problems, e.g. demonstrated by the teacher by placing a set of objects in the sensory environment or real scenes encountered during online operation of the robot, a parallelized optimization process is started with the goal to find a maximum subset of learned constraints which admits a successful solution to all test problems. Since the generalization process relies heavily on planning and execution of hypothetical planning models in (dynamics) simulation, interaction with the teacher is reduced to a minimum. The definition of test problems reflects directly the normal use-cases of the robot (and can be done online) and thus requires less effort and less experience compared with the demonstration of a large number of demonstrations with distinctive difficulty.

By learning from explicit human demonstrations as well as test problems, the relevant properties of complex tasks with hundreds of constraints can be efficiently deduced, e.g. compared to trial-and-error or exploration-based learning. The main disadvantage is that the differences in morphology between the human and the robot may be severe and thus cannot be neglected in the learning process, which is called the *correspondence problem*. We consider the correspondence problem by relaxing learned constraints restricting the motion of human body parts, e.g. the fingers and wrists. Since the effects of the manipulation, e.g. an induced object motion, will not be altered, the goal of the task can still be achieved using constraint-based motion planning but in a larger search space. We cope with this problem by learning search heuristics online, which depend on the actual robot morphology and geometry of manipulated objects. A fast control algorithm is incorporated into CBiRRT to speed up the planning problem with the learned search heuristics. We project the set of search heuristics to each learned constraint, resulting in a refinement of the constraint region R . The resulting tightening of the constraint is the counterpart of the previous relaxation. Since the tightening depends on the robot morphology, the correspondence problem is considered explicitly in the mapping process.

In Figure 4.1, an overview of the developed PbD system with reference to the introduced major components is shown. In Section 4.1 we describe the sensor systems, sensor environment and sensor data, which represents the basis for the developed learning algorithms. Learning of the initial planning model, which is usually overspecialized due to a large number of automatically generated constraints, is described in Section 4.2. The differences in morphology are considered explicitly

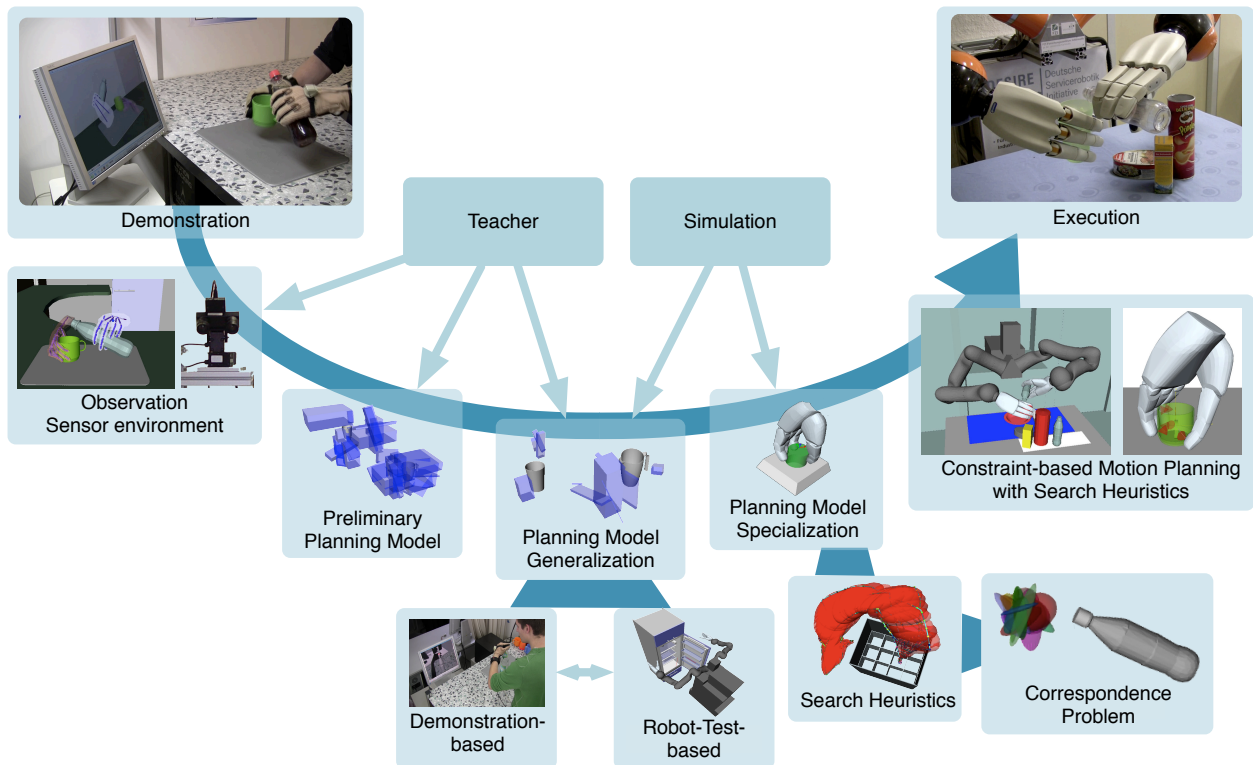


Figure 4.1.: Overview of developed PbD process.

during the learning of constraints and is defined in Section 4.3. In Section 4.4, we describe a generalization algorithm based on the idea of Curriculum Learning, i.e. learning with sets of examples of different complexity, to remove irrelevant constraints from the initial planning model resulting in a higher flexibility. Since training examples are usually limited and a lot of expertise is required, a different, complementary algorithm was developed. In Section 4.5, a parallelized evolutionary algorithm is defined, which iteratively adapts the overspecialized planning model. The goal is to find a maximum number of constraints which allows to solve a number of test problems. Finally, we refine or specialize the generalized planning model, see Section 4.6, to reduce planning time by learning from multiple planning results and by considering geometric properties of the manipulated objects and the robots.

4.1. Observation of Explicit Human Demonstrations of Manipulation Tasks

In the human environment, service robots have to be able to solve manipulation tasks of different complexity. A bimanual task requires the coordinated motion of two robot arms, e.g. to open a rough-running bottle cap. In contrast to this, a dexterous manipulation task might require the coordinated motion of the robot arm and one or more fingers. The set of constraints, which we consider in the planning process, see Chapter 3, contains position, orientation, direction, force, moment, contact and temporal constraints. The set of constraints captures a large variety of different manipulation effects, e.g. induced object motion due to fingertip force interaction. In order to learn

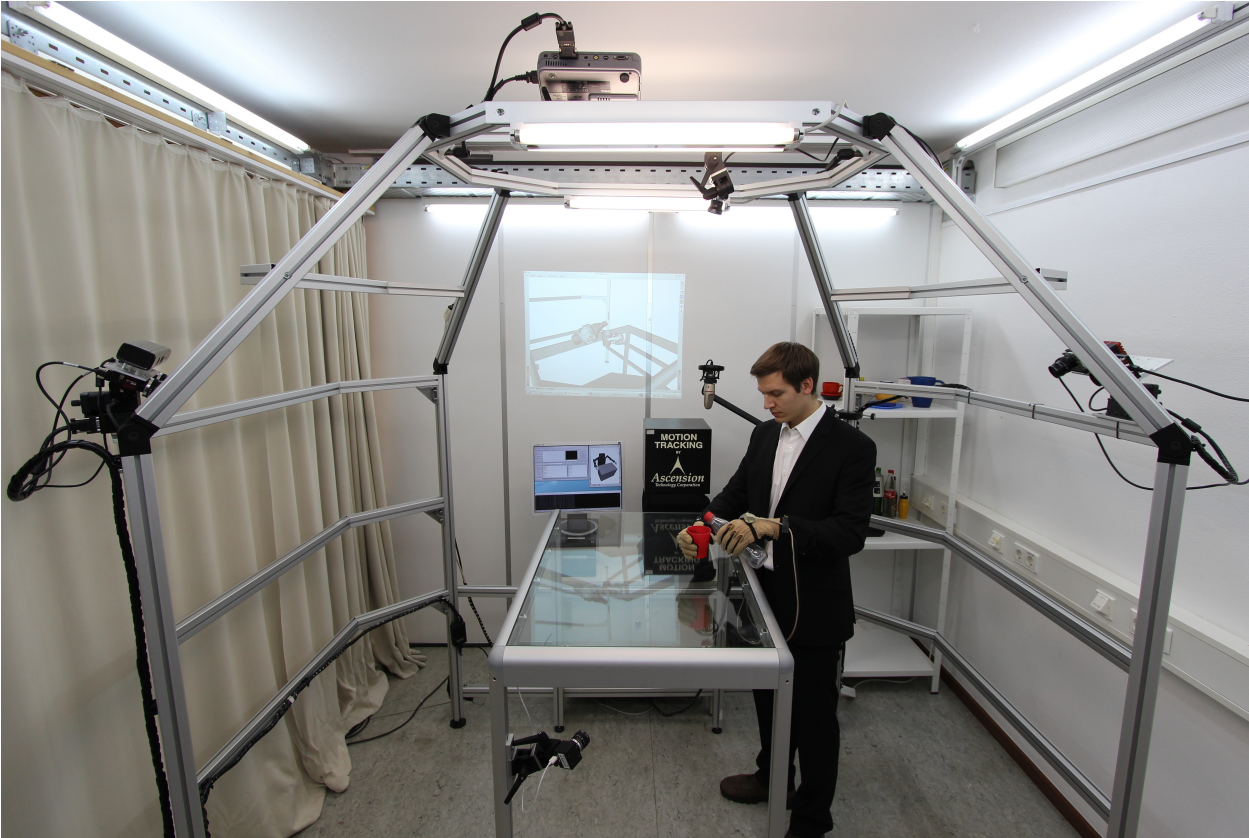


Figure 4.2.: Sensor environment with two robot heads, magnetic field trackers, datagloves and tactile sensors to observe human demonstrations of a manipulation task.

planning models based on a number of human demonstrations, these effects have to be observed consistently, i.e. without interruptions and limited noise. The accuracy has to be high since the effects are encapsulated into constraints and used directly in the motion planning process. If accuracy is low, learned constraints might include situations, in which the robot or an object collides with the environment, leading to problems in physics simulation and collision checking. Currently, observation of fingertip motion is insufficient with online robot perception not only because of occlusions but also because of the limited resolution in RGB-D data and the problem to find correspondences in RGB data.

A dedicated sensor environment to measure human fingertip forces, finger joint angles, wrist pose and object poses is used to overcome the limitations of online robot perception.

4.1.1. Sensor environment

The used sensor environment is shown in Figure 4.2. A stereo-based camera system is used to detect and localize known objects in the environment. The position and orientation of both human hands is determined using a magnetic-field-based motion tracker. The joint angles of the human hands are captured with two datagloves. We measure the forces in the fingertips with two gloves with tactile sensors, which can be combined with the datagloves.

The joint angles of the human fingers are measured using two Cyberglove II datagloves. Each dataglove measures 4 DOF for each finger and 2 DOF for the lower arm orientation relative to the palm. In the learning process, we do not consider the human arm configuration and the latter 2 DOF are ignored. Since the position of the sensor strips on the finger joints changes each time, the datagloves have to be calibrated. We use an existing software tool based on hand gestures to calibrate the datagloves. The advantage of the usage of datagloves compared to a camera-based approach is that the sensors will not be occluded by the manipulated objects, the environment or the human arms.

In general, no direct measurement of the fingertip position is available and it has to be calculated using a kinematic model of the human hand. We distinguish between user-dependent, see Figure 4.3c, and user-independent, see Figure 4.3d, hand models. The user-independent model was created in prior work and can be used in manipulation tasks, in which the fingertip position does not have to be accurately known, e.g. when objects are only grasped and not manipulated with the fingertips. The user-dependent hand model was generated using a laser scanner [53]. First, we made a plaster cast of the human hand. The plaster cast was mounted on a rotation-plate and scanned using a Minolta VI-900 laser scanner. Different 3D scans from multiple angles were registered automatically using the Rapidform-library to generate the initial hand model. After filling gaps and removing outliers with the Rapidform-library, the model in Figure 4.3b was obtained. We added the bone structure manually resulting in a fully articulated hand model with similar kinematics and highly-accurate geometry of the human hand.

Similar to the finger joint measurements, we aim at a consistent measurement of the human wrist pose without occlusions. In the context of manipulation, the human hand is often occluded by objects or the human himself. We employ a Flock of Birds magnetic-field-based tracking device to measure the 6D pose of each human hand. The sensor has a static accuracy of 0.18mm RMS¹ for the position and 0.5° RMS for the orientation [3]. Although the results are distorted by metal objects in the environment, the influence is only local and can be efficiently avoided. If objects, which are usually made from metal, e.g. spoons, have to be manipulated, they are replaced by a plastic or wood imitate.

The gloves with tactile sensors [53] are based on an elastomer, which changes resistance when pressure is applied, see Figure 4.4. In each fingertip, a tactile sensor pad is used to measure the force intensity applied to an object. Due to the size and location of the pads, the human operator has to adapt the manipulation motion in order to make consistent force measurements. The system is calibrated using a dynamometer.

The stereo vision system with two DragonFly II cameras is used to localize known objects in the environment. The IVT library [4] is trained with a 3D model to generate different views of the object, which are then used for the localization algorithm. The 3D models are available in the KIT ObjectModels Web Database [59].

¹Root mean square (RMS) is the square root of the mean square error.

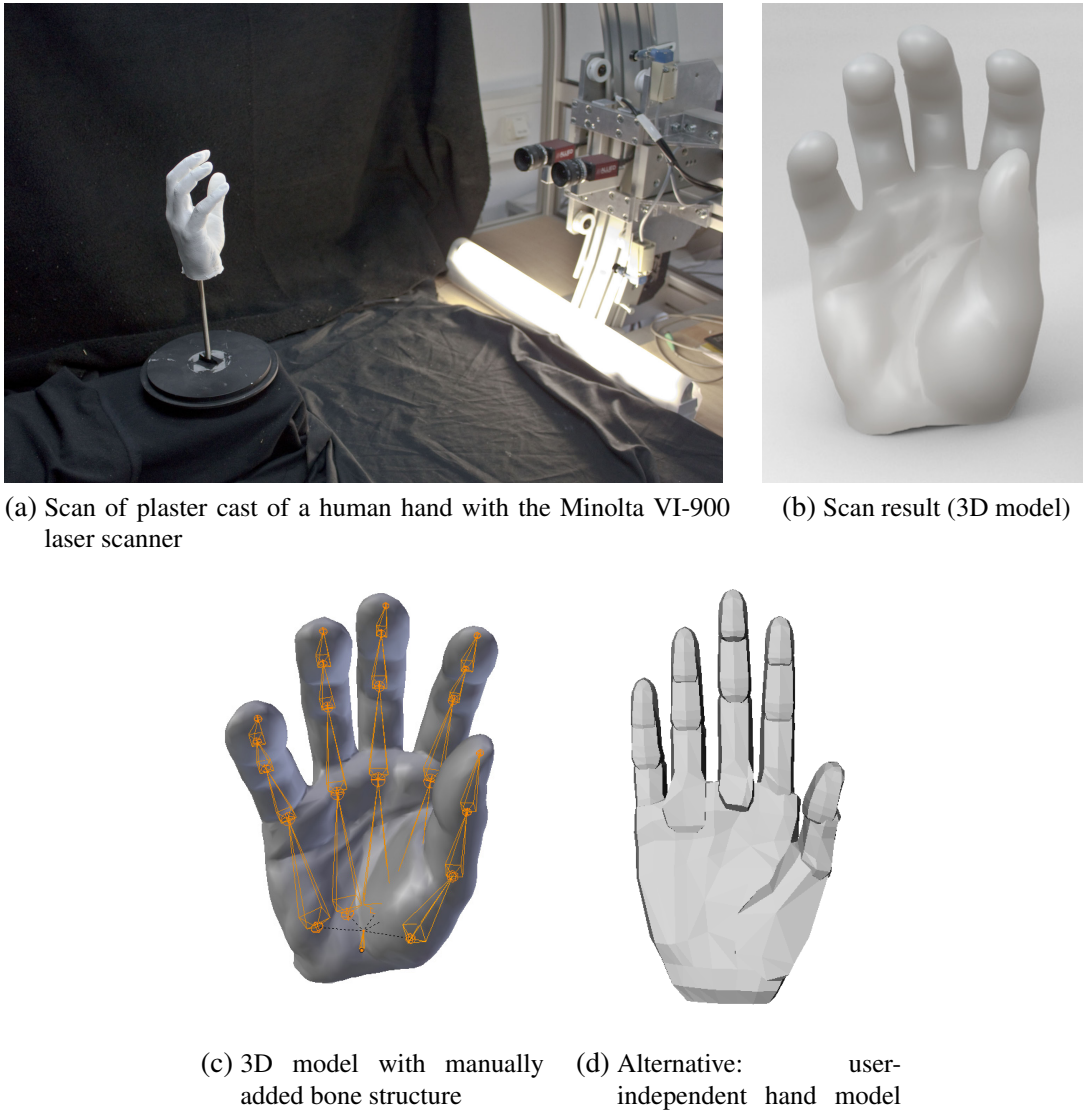


Figure 4.3.: Generation of a user-dependent, accurate model of the human hand with manually added bone-structure (a,b,c). (d) shows an alternative, manually defined user-independent hand model. - [53]

4.1.2. Sensor data

The sensor data consists of two 6D poses for the human wrists, two 20D joint angle vectors for the finger joints and a 1D force value for each fingertip of both human hands. Additionally, for each object the vision system produces a 6D pose. The measured data is used to visualize the observation result in a 3D simulation environment, see Figure 4.5. We calculate and add the pose of each fingertip using forward kinematics based on the bone-structure of the hand, see Figure 4.3c. In the 3D simulation, the proximity query library PQP [106] is used to determine contact pairs $(o_1, p_1, n_1, o_2, p_2, n_2, \Delta x)$ with objects o_i , contact points p_i , contact normals n_i and distance Δx . The contact pair will be added to the sensor data if $\Delta x < 5\text{mm}$. The threshold is a worst-case estimation of the localization error. We apply a grasp classifier to the joint angles of hand i

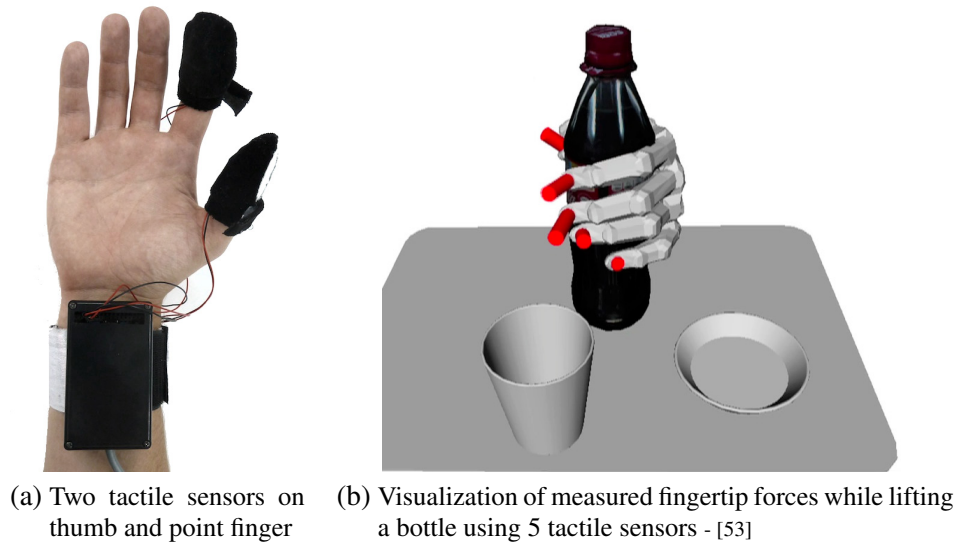


Figure 4.4.: Tactile sensors to measure fingertip contact forces.

resulting in the classification p_i . The grasp classifier was learned using a Support Vector Machine [108] on a large number of demonstrated grasps of the Cutkosky grasp hierarchy [27], which were generated in prior work.

In the example in Figure 4.5, the goal is to grasp a bottle with the right hand. When the fingers apply force on the object and the arm is ready to lift it, we measure one force value in each fingertip, The 6D pose of the wrists and the bottle are measured. Based on the measured finger joint angles we generate a 6D pose for each fingertip. Contacts between bottle and table and between the bottle and each fingertip are calculated. The resulting sensor data stream in this example is 109D. The high dimensionality of the sensor data stream, even in this simple manipulation task, shows the need to extract relevant constraints and generalize the learning result.

4.1.3. Explicit Demonstrations

The problem to learn manipulation knowledge based on human demonstrations can be defined on various levels of difficulty. The most difficult problem arises when the human is observed passively, i.e. without knowledge of being observed. In this case, the segmentation problem, i.e. how to segment the sensor data stream and match segments from different demonstrations to each other, is demanding. In the least difficult, the human possesses expert knowledge about the learning algorithm and segments the sensor data stream manually and assigns segments to each other manually. In our work, the human segments the manipulation task in a natural way by making pauses in the demonstrations in the order of $100ms$. He is instructed to make the same pauses during several demonstrations. Based on this assumption, the sensor data stream is segmented automatically by looking for pauses. Segments from different demonstrations are mapped to each other automatically based on a simple quality criteria, e.g. using Dynamic Time Warping [90].

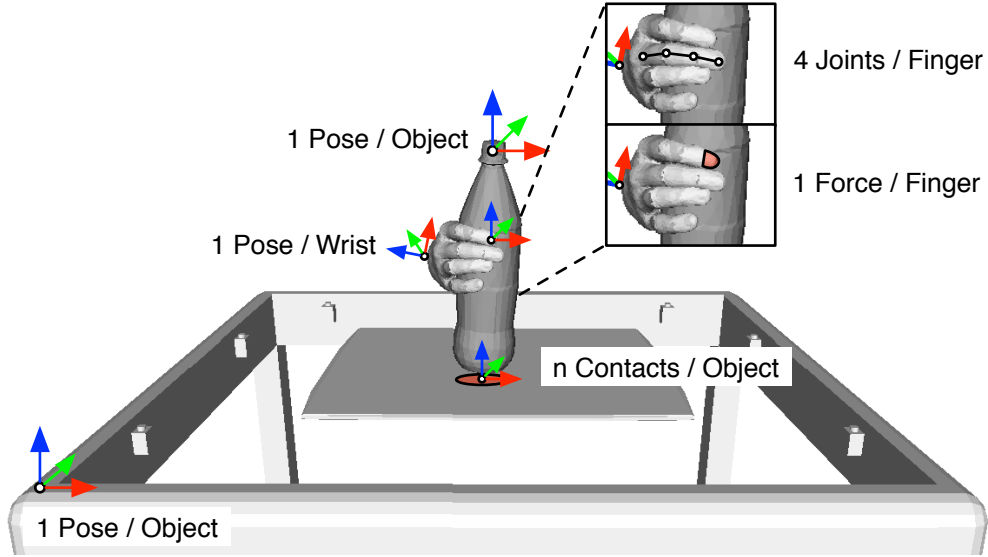


Figure 4.5.: Sensor data visualization. The goal is to grasp a bottle with the right arm. The measurements are the pose of the bottle (6D), the pose of the human wrist (6D), the finger joint angles (20D) and the force values in the fingertips (5D). In the simulation, contacts between the table and the bottle and each finger with the bottle are calculated and added as artificial measurements.

Definition 3 (Segmentation) A segmentation of a path π is a tuple (t_0, \dots, t_n) with $t_i \in [0, 1]$, $t_0 = 0$, $t_n = 1$, $t_i \leq t_{i+1}$.

In Definition 3, a segmentation (t_0, \dots, t_n) of a path π is defined. We generate a segmentation automatically by analyzing the velocities of fingers $v_{Finger_{ij}}$, hands v_{Hand_i} and forces $v_{Force_{ij}}$ as well as the minimum distance of all fingers to an object d_{object_k} . The approach is similar to the definition of the Grasp predicate in [80, p. 86]. A segmentation point t_i will be generated if the hand pose, finger joints and measured forces do not change, i.e.

$$\begin{aligned}
 & \|v_{Hand_i}\| < \delta_{Hand} \\
 \wedge & \|v_{Finger_{ij}}\| < \delta_{Finger} \\
 \wedge & \|v_{Force_{ij}}\| < \delta_{Force}
 \end{aligned} \tag{4.1}$$

with predefined thresholds δ_{Hand} in $\frac{m}{s}$, δ_{Finger} in $\frac{RAD}{s}$ and δ_{Force} in $\frac{N}{s}$. The result is a large set of segmentation points since the criteria is usually valid for multiple successive time points. We cluster the set of segmentation points using Agglomerative Hierarchical Clustering with single linkage and threshold δ_{Time} in s . Segmentation points belong to the same cluster if they occur within δ_{Time} of each other. In the experiments, $\delta_{Time} = 0.5s$ was used. Let n be the number of clusters. For each cluster i , we calculate the minimum t_i^{min} , maximum t_i^{max} and average value t_i^{avg} , which will be used in the next section to learn the preliminary planning model. Each demonstration is represented as a path π with n segmentation points (t_0, \dots, t_n) . The result of the observation



Figure 4.6.: Segmentation of a single human demonstration of the pour-in task.

process is the measured sensor data stream with a set of segmentation points, in which the human teacher made explicit pauses during the demonstration.

In Figure 4.6, the segmentation of a single human demonstration in the pour-in task is shown.

4.1.4. Discussion

During object manipulation fingertip or hand motion is often occluded to an external viewer. State-of-the-art vision systems are currently not capable of providing a continuous precise observation of all relevant effects of a manipulation tasks. We overcome this limitation by using a dedicated sensor environment with occlusion free sensors and a model-based approach, e.g. supply 3D models of all

manipulated objects as well as a geometric and kinematic model of the human hand. This allows us to calculate contacts using state-of-the-art proximity query algorithms based on the 3D models and add artificial contact measurements. Important information, which cannot be measured directly, e.g. fingertip position due to occlusions, is deduced based on a user-dependent 3D hand model.

4.2. Generation of Preliminary Planning Model

In the previous section, the foundation of the learning process was laid. The human demonstrations in the real world were transformed into a purely virtual representation by applying different sensor systems to measure the effects in the real-world and by applying simulation techniques to generate artificial measurements, which cannot be observed consistently in the real-world. The domain knowledge of the human teacher is exploited to choose a set of illustrative examples, from which the manipulation task can be learned efficiently, and make explicit pauses to highlight important aspects of the task.

In this section, the goal is to generate an initial planning model. The planning model will be generated in two distinctive steps. First, the segmentation of the sensor data will be used to generate the structure of the planning model, i.e. number and ordering of (sub-)goals. Since the segmentation points are sorted by time, the result will be a linear planning model.

In the second step, the set of constraints to describe nodes and arcs in the planning model has to be chosen and the parameters have to be learned based on the set of human demonstrations. Different constraints represent different correlations between coordinate frames. Based on the assumption that we can observe all relevant information in the sensor environment, we can only deduce that coordinate frames relative to detected objects as well as measured forces are relevant to the task. The dependencies between these coordinate frames are unknown, e.g. when placing a bottle on the table the position of the bottle bottom relative to the table is important but while pouring liquid into a cup it might be irrelevant. Based on this observation we generate a large number of constraints automatically, usually more than 20 for each node and arc. The set of constraints can be interpreted as a Task Space Pool [76]. We learn the parameters of each constraint and add it to the planning model. The planning model captures a lot of irrelevant information, is therefore overspecialized but can already be executed in the demonstrated scenes.

4.2.1. Structure of the Preliminary Planning Model

Our goal is to generate a planning model with a linear sequence of (sub-)goals, which can be planned efficiently using the developed constraint-based motion planning algorithm. Since the human teacher is responsible for making explicit pauses to segment the sensor data, this is a trivial step [52]. The first demonstration will be the reference. All other demonstrations will be rejected if they differ in the number of generated segmentation points. If a demonstration is rejected the human teacher will be notified.

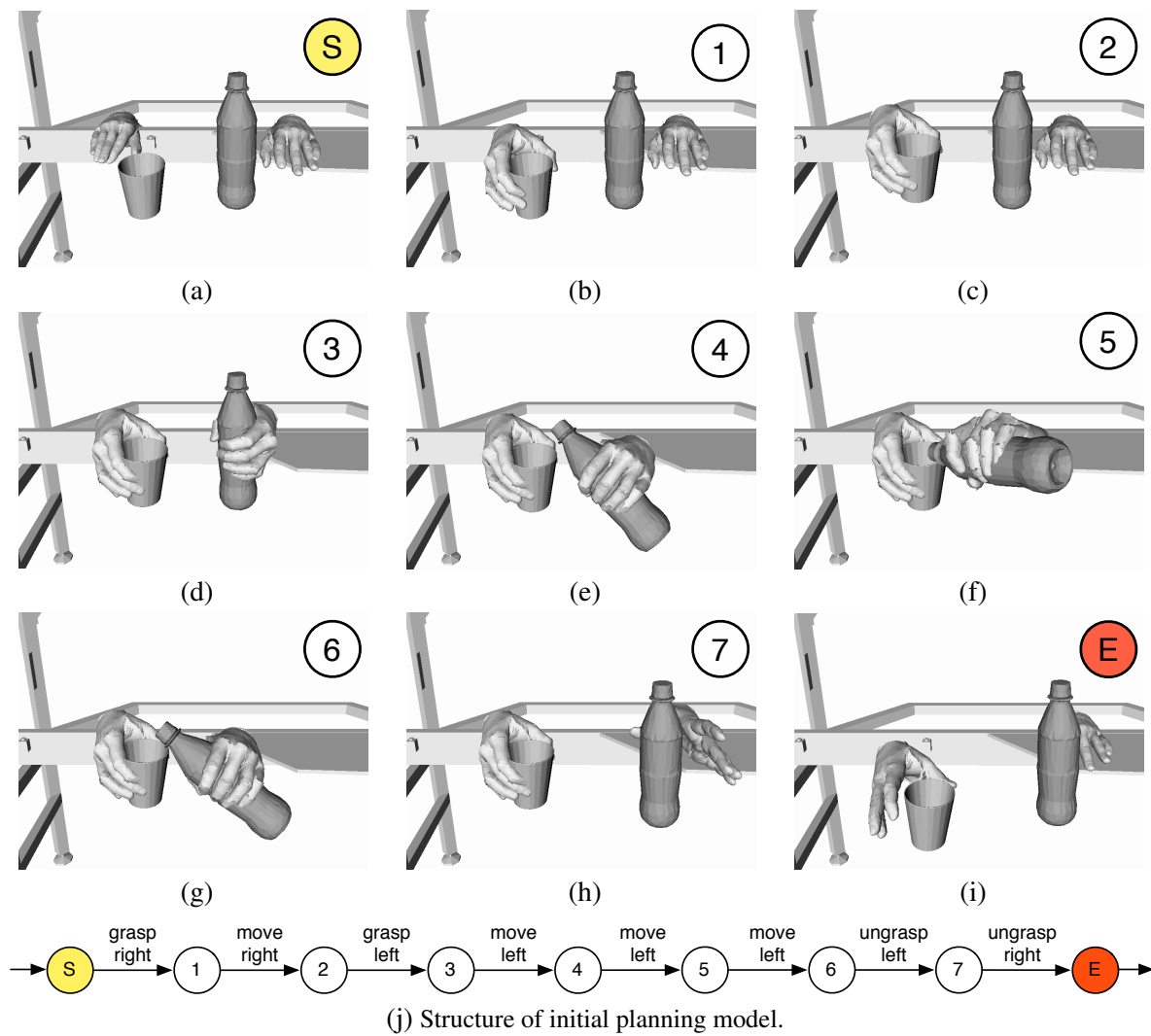


Figure 4.7.: Segmentation points (a) - (i) extracted from the human demonstrations of the pour-in task and generated structure (j) of initial planning model.

The generation of the structure in the bimanual pour-in task is shown in Figure 4.7. The human teacher makes explicit pauses, which are reflected in the generated structure of the planning model. Demonstrations will be rejected if the number of pauses differs. In the experiment, a small number was rejected since the cup is usually not held still during transport of the bottle. In the worst case, this produces an extra (sub-)goal for the right arm but, since the teacher is notified, he usually reacts to this error source and provides different demonstrations.

4.2.2. Automatic Generation of Contact Coordinate Systems

Coordinate frames form the basis for the definition of the constraint set. In the object database [59], predefined coordinate frames are stored for each object, e.g. for a bottle the coordinate frame in the center of mass, in the center of the opening and the center of the bottom surface exist. During manipulation, the robot makes contact between its fingertips and the object surface. In Figure 4.8, a human demonstration to open a bottle with two fingers is shown. The human makes contact with

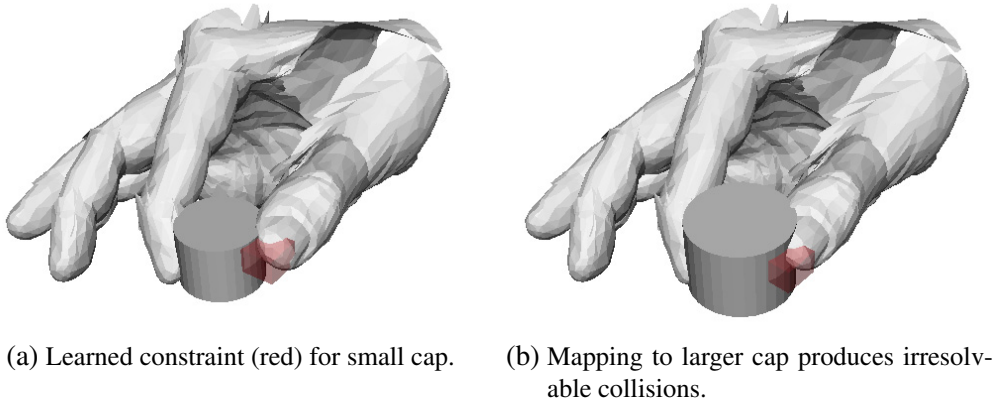


Figure 4.8.: Rotating a bottle cap without contact coordinate frames.

the bottle, applies force through the fingertips and the cap moves. If the motion is encapsulated in constraints, which refer to the predefined coordinate frame in the center of the cap, the position of the fingers will be restricted to be in a certain fixed distance to the center. If we map the constraint to a bottle with bigger cap, the finger will collide with the bigger cap since the constraint does not adapt to the different geometry.

The object geometry has to be taken into account to overcome this limitation. We generate coordinate frames in contact points, which were measured consistently on at least 95% of a segment [53]. For each segment (t_i^{min}, t_j^{max}) we analyze the measured contact pairs $(o_1, p_1, n_1, o_2, p_2, n_2, \Delta x)$ and count the number of contacts for each (o_1, o_2) pair. If the number is greater $0.95(t_j^{max} - t_i^{min})$ a contact frame will be generated, i.e. we allow 5% false positives to cope with noise, which was determined experimentally. The contact frame is relative to o_2 . The position is the (statistical) mode of all contact points p_2 . The orientation is calculated in two steps. The z-axis is set to n_2 of the mode. The x-axis points along the first principal component and the y-axis along the second principal component of the contact points. An example is shown in Figure 4.9. Based on multiple contact pairs, a contact frame for a big cap is generated, see Figure 4.9a. The contact frame is mapped to a scanned bottle cap, see Figure 4.9b using a blended intrinsic map, see Figure 4.9c.

4.2.3. Automatic Deduction of Constraint Sets

Constraints represent correlations between different coordinate frames, e.g. a force constraint restricts the forces measured in the coordinate frame of the fingertip relative to the coordinate frame of the touched object. By intersecting multiple constraints, a complex constraint manifold is generated, which can be used to represent a given manipulation task adequately, see Section 3.2.

For the planning process, one important consequence is that all coordinate frames, which are referenced by the constraints, have to exist in the execution environment. Otherwise it is difficult to determine if the planning model can still be applied or not. If only relevant constraints are stored, the answer is clear: the planning model cannot be applied and the robot cannot execute

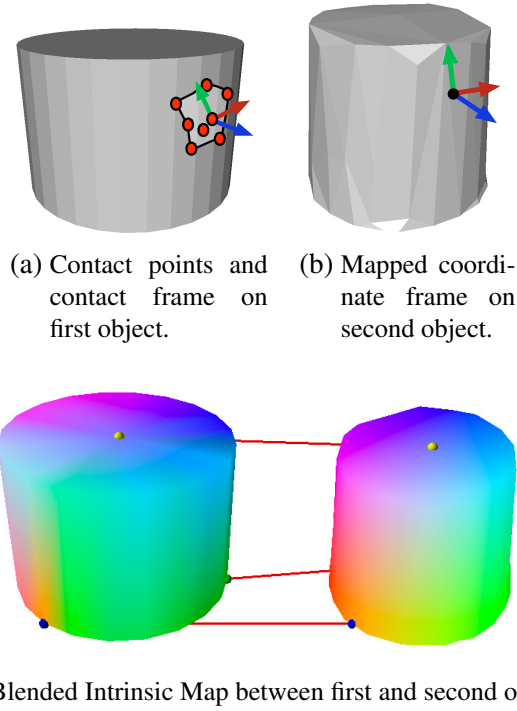


Figure 4.9.: Example of a generated contact frame. Circles represent calculated contact points on the object. The coordinate frame is generated in the mode of the contact points with the z-axis pointing along the object normal and x- and y-axis along the principal components (a). It is mapped to a different object (b) using the blended intrinsic map (c).

the manipulation task in the current environment. A second consequence is that each constraint restricts the search space during planning, which leads to potentially shorter planning times but leads to less flexibility. We refer to a planning model with a larger number of irrelevant constraints as overspecialized. In this case, the robot will be able to execute the manipulation task successfully in the environments, which were used in the demonstrations, but it will generalize poorly to different environments. Since flexibility is one of the key abilities to master real-world applications, we will address this problem in the next sections and apply different techniques to identify irrelevant constraints and generalize the planning models.

Based on the types of constraints and set of coordinate frames we can only identify a small number of irrelevant constraints, e.g. if we cannot measure a force in the tip of the middle finger on the segment (t_i^{min}, t_j^{max}) , the finger will not participate in the manipulation and all constraints, which restrict the motion of the coordinate frame in the fingertip will be removed.

In this section, we introduce a rule-based system to build the set of potentially relevant constraints [53] based on measured sensor data, which is similar to the concept of the Task Space Pool by Mühlig [76]. Two constraint sets are defined. The first set is applied to learn constraints for Pick-Transport-Place tasks, in which an object will be picked up, moved on a complex path, e.g. to align it with a different object, and placed relative to a second object. In this set, a grasp quality constraint will be automatically added for grasps and the fingertip coordinate frames as well as

4. Programming by Demonstration of Planning Models for Manipulation Tasks

Label	$g_j(t_i^{min})$	$g_j(t_{i+1}^{max})$	Label	$g'_j(t_i^{min})$	$g'_j(t_{i+1}^{max})$
Pick	-	+	Constrained	-	+
Place	+	-	Constrained	+	-
Transport	+	+	Constrained	+	+
Free	-	-	Free	-	-

(a) Constraint set 1
(b) Constraint set 2

Figure 4.10.: Different labels for constraint set 1 and 2, which are calculated based on the value of the predicates g_j and g'_j in the first t_i^{min} and last time point t_{i+1}^{max} of a segment. The predicates can either be true (+) or false (-).

forces in the fingertips will not be considered, which reduces the number of generated constraints. Since forces are only measured between a fingertip and an object, we assume that it is not required to simulate force interaction. In the second set, we model dexterous manipulation tasks, which require to capture force interaction with the fingertips, and consider all fingertip constraints.

We generate two initial sets of constraints for each segment $(t_i^{min}, t_{i+1}^{max})$: one set for the transition from $(t_i^{min}$ to $t_{i+1}^{min})$ and one for the goal $(t_{i+1}^{min}, t_{i+1}^{max})$. Constraints will be generated based on a set of coordinate frames \mathcal{F} . \mathcal{F} contains one coordinate frame representing the measured force in each fingertip, one coordinate in each fingertip, one coordinate frame in each wrist, all coordinate frames of all recognized objects in the scene and all generated contact frames.

The first step is to generate additional coordinate frames, which represent the pose of a coordinate frame relative to the base coordinate frame at the beginning of the segment. The reason is that a lot of motions during manipulation tasks are relative to the configuration at the beginning of the task, e.g. if we rotate a bottle cap, the motion depends on the orientation of the cap prior to the manipulation. For each coordinate frame $f \in \mathcal{F}$ we add a *start* coordinate frame with a fixed homogenous transformation ${}^0H_f(t_i)$.

Constraint set 1: Pick-Transport-Place tasks

We assign a label $l_j = \text{Pick}, \text{Place}, \text{Transport}$ or *Free* for each manipulator j based on the result of the grasp classifier p_j and the minimum distance of a finger to an object in the environment $\min_k d_{object_k}$ at the beginning and end of the segment. The latter is represented by a predicate

$$g_j(t) = (p_j(t) \geq 0) \wedge \left(\min_k d_{object_k} \leq \delta_{Contact} \right), \quad (4.2)$$

which is true if a grasp was classified and at least one finger is closer than $\delta_{Contact}$ (in mm) to an object. The label is assigned based on the value of the predicate in the first and last time point of the segment. The assignment is shown in Figure 4.10a. For each label, different constraints are generated based on the assumption that the robot interacts with the world only by grasping objects and transporting objects relative to each other.

Free-segment In this case, the human arm moved and no object was grasped at the beginning or end of the segment. We generate all possible position, orientation and direction constraints (e_1, e_2, e_3, f, R) , where e_1 is the human wrist and e_2 and e_3 are relative to an object. For position constraints, e_3 can also be relative to e_1 to consider the correlation of the position of e_1 and its orientation. Since the objects could not have moved, *start* frames are omitted. For goals, we add a predefined configuration constraint for the fingers, which corresponds to the detected grasp class $g_j(t_{i+1}^{max})$, e.g. a circular power grasp constraint. In the mapping process, this predefined constraint will be mapped to a preshape of the robot hand to consider the correspondence problem.

Transport-segment We observed that an object was grasped at the beginning and end of the segment and the fingers remained motionless (otherwise the motion would have been segmented differently). The motion of the grasped object relative to other objects in the scene is analyzed. We generate all possible position, orientation and direction constraints, where the first coordinate frame e_1 is relative to the grasped object, the second e_2 and third coordinate frame e_3 are relative to an object in the scene or a *start* coordinate frame of an object in the scene. For position constraints, e_3 can also be relative to the grasped object. Based on the assumption that all objects are rigid², constraints, where e_1 , e_2 and e_3 are relative to the same object, will always be obeyed and are therefore removed. Due to the large number of possible coordinate frame combinations, a large number of constraints is usually generated.

Pick-segment In this segment, the human hand has no object grasped in the beginning, approaches the object, closes the hand and applies force to the object. The motion is always object-dependent and therefore we generate all possible position, orientation and direction constraints, where e_1 is the human wrist and e_2 and e_3 are relative to the grasped object. In order to consider the correlation between the position of the human wrist and its orientation, e_3 can also be relative to e_1 for position constraints. For goals, we add a grasp quality constraint for the grasped object.

Place-segment Due to the duality of approach and retreat during grasping, we generate the same constraints as in the Place-segment. For goals, no grasp quality constraint is added but a predefined configuration constraints, which corresponds to the detected grasp class $g_j(t_{i+1}^{max})$.

In the Transport-, Pick- and Place-segments, contact pairs between two objects, which were measured on 95% of the segment, will be mapped directly to contact constraints, see Section 4.2.2. Since we cannot measure forces between objects but only between the human fingers and objects, no force constraints will be learned.

In the Transport-segment, constraints are generated based on the combination of coordinate frames relative to the grasped object and the environment. Due to the large number of combinations, a high complexity arises. Even in simple examples, a large number of ambiguities exist, which cannot be resolved based on the analysis of the motion data, e.g. when lifting a chair, the

²Objects with inner degrees of freedom, e.g. a fridge with door, are modeled as a kinematic chain linking two objects.

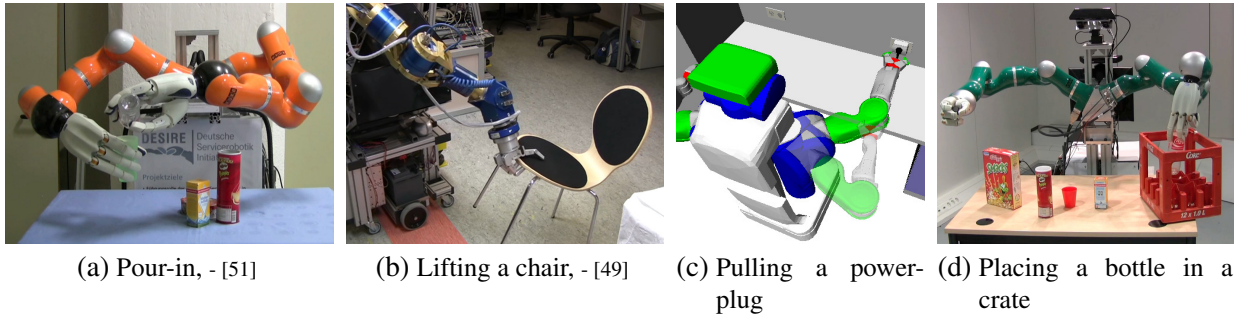


Figure 4.11.: Constraint set 1: examples of manipulation tasks.

chair motion could be described relative to the start pose of the coordinate frame in the contact point or relative to predefined coordinate frames in the seat, arm rests or backrest. Since irrelevant constraints limit flexibility of the learned planning model, we discuss different techniques to identify irrelevant constraints in Section 4.4 and 4.5.

Figure 4.11 shows examples of manipulation tasks, which were learned using constraint set 1.

Constraint set 2: Fingertip-Manipulation-tasks

The previously defined constraint set does not cover fingertip motion, forces applied through the fingertips and object motion induced by forces without a firm grasp. Similar to the previous constraint set we assign two labels, *Free* or *Constrained*, to each segment. *Free* is assigned to a segment, if no finger is close to an object, otherwise the segment is labelled *Constrained*. Figure 4.10b shows the assignment of labels based on the value of the predicate

$$g'_j(t) = \min_k d_{object_k} \leq \delta_{Contact} \quad (4.3)$$

in the first and last time point of the segment.

Free-segment The same set of constraints is generated as in the Free-segment of constraint set 1. The motivation is to allow the human teacher to demonstrate explicitly preshapes, e.g. hand configurations, from which a fingertip manipulation can be started.

Constrained-segment We generate all possible position, orientation and direction constraints, where e_1 is the human wrist and e_2 and e_3 are relative to the coordinate frame or *start* coordinate frame of an object. For position constraints, e_3 can also be relative to e_1 to consider correlations between the position and orientation of e_1 . If a force was measured in fingertip i on the segment, force and position constraints will be added, where e_1 is the fingertip i and e_2 and e_3 are relative to the touched object. If a force between one of the fingertips and object i was measured, we add all possible position, orientation and direction constraints, where e_1 is relative to object i and e_2 and e_3 are relative to coordinate frames or *start* coordinate frames of objects.

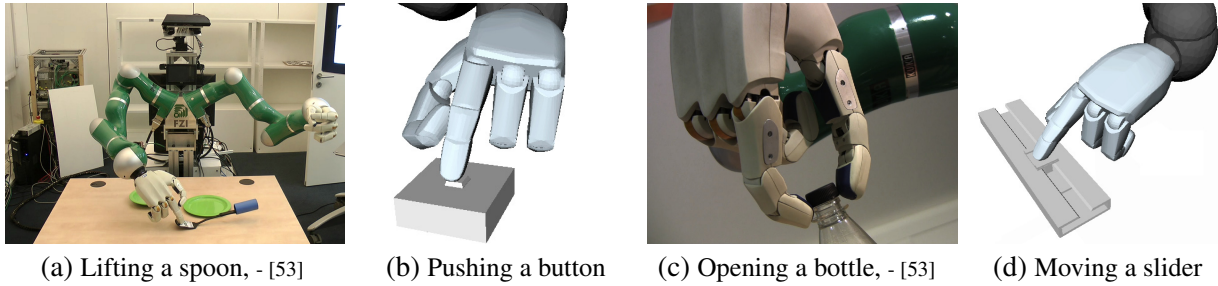


Figure 4.12.: Constraint set 2: examples of manipulation tasks.

Less restrictions on the set of constraints lead to a higher number of constraints compared to constraint set 1. In Figure 4.12, examples of manipulation tasks using constraint set 2 are shown.

4.2.4. Learning of Constraint Parameters

In the previous subsection, we introduced techniques to define a general set of constraints, which can be used for a large number of different manipulation tasks. At this point, the parameters of each constraint $\tau = (e_1, \dots, e_m, f, R)$ are only partially defined, i.e. the constraint region R is still missing. We already specified the type of constraint f and which coordinate frame e_1 changes its value relatively to the other coordinate frames $e_i, i > 1$.

In this subsection, the goal is to determine a minimal region R , which spans all values of f on the current segment $(t_i^{min}, t_{i+1}^{max})$. In Section 3.1.3, we defined different types of parameterized constraint regions, e.g. hypercubes and convex hulls, along with algorithms to determine the parameters for a given set of points. The resulting parameters lead to a constraint region R with minimal surface area $\mu(R)$ [51]. We apply the surface area rather than the volume, e.g. Lebesgue measure, since the learned regions are bounded but may be infinitely thin, e.g. when learning a goal region to place a bottle on a table. In general, different μ are possible. In [18] and [47], surveys about 3D similarity search and different numeric descriptors, e.g. surface curvature or compactness, can be found.

First, we accumulate the values of f on the segment $(t_i^{min}, t_{i+1}^{max})$ into the set $X = \{(f(t), t) \mid t_i^{min} \leq t \leq t_{i+1}^{max}\}$. Second, we calculate the parameters of each region type using the algorithms in Section 3.1.3 based on the data X and choose the region type with minimal surface area $\mu(R)$. The main advantage of this *bag of algorithms*-approach is flexibility. It scales to all region types, which can be calculated efficiently and allow to calculate, at least approximately, the surface area. The defined constraint region types, except sequences of ellipsoids, ignore the temporal component t of X . The motivation is that we learn an approximation of the constraint manifolds defining the manipulation task, i.e. the search space to generate goal configurations and plan a robot motion in a goal-directed way. In the planning process, the geometric, kinematic and dynamic properties of the robot are considered, which are not available at the time of learning.

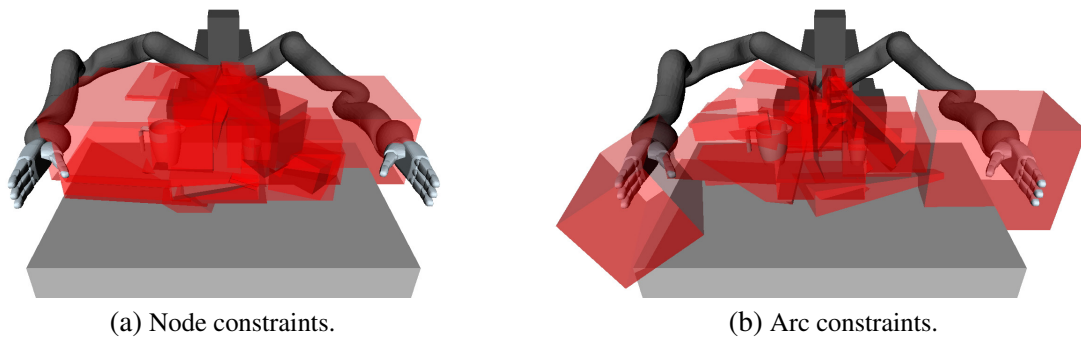


Figure 4.13.: Learned cubical constraints in the pour-in task. - [50]

In Figure 4.13, the automatically generated constraints in the pour-in task are visualized. For simplicity and clarity, only the region type *cube* was used. 64 position constraints are shown. The set of constraints includes highly relevant constraints, which restrict the bottle opening to stay above the cup opening during pour-in, but also irrelevant constraints, which restrict the bottle bottom relatively to the world coordinate frame. In Section 4.4 and 4.5, we discuss complementary approaches to identify relevant constraints and increase flexibility of the learned planning model.

4.2.5. Discussion

The first constraint set is suited to tasks, in which objects are picked, placed, transported and used to manipulate other objects, see Figure 4.11 on page 100. Examples are bimanual pour-in, lifting a chair, pulling a power-plug out of an outlet or placing a bottle in a crate. By using grasp quality constraints, complex grasp quality measures are incorporated and the resulting planning process implicitly contains a grasp planning process, in which the robot hand is placed based on demonstrated wrist poses relative to the object and the fingers form a preshape based on predefined configuration constraints. The results are potentially high-quality, task-dependent grasps and complex, collision-free transport motions.

In the second constraint set, the finger motion as well as finger contacts and forces on the object are not generated indirectly based on the grasp quality constraints but learned directly. The predefined grasp quality constraint is replaced implicitly by constraints for the position of the fingertips, the pose of the wrist, constraints for the forces in the fingertips as well as constraints for the contact region of the fingertips. The second constraint set is suited to tasks, in which objects are manipulated with the fingertips only, e.g. opening a bottle with the fingertips, pushing a button, moving a slider or lifting a spoon, see Figure 4.12 on page 101. In principle, the constraint set is more general and can be applied to arbitrary dexterous manipulation tasks. Current limitations are the large number of constraints and the increased complexity of the planning process.

The two constraint sets serve as templates to generate large numbers of constraints independent of the task but depending on the objects in the scene and measured forces in the fingertips. Due

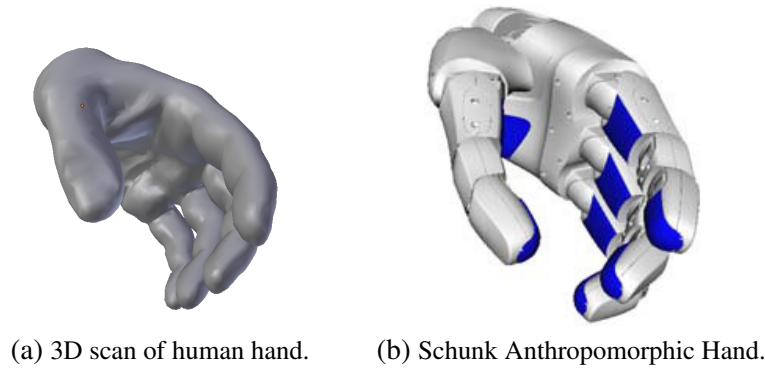


Figure 4.14.: Example of a human and a robot hand.

to the limitations of the available sensor systems, only forces between the fingertips and objects are considered. If the force interaction between different objects can be measured, additional force constraints have to be considered in the constraint sets. In general, the learning algorithm, the generalization algorithms in Section 4.4 and 4.5, and the specialization algorithm in Section 4.6 scale to arbitrary constraint sets.

4.3. Consideration of the Correspondence Problem

In order to execute the planning model on the robot system, we have to consider the differences in the morphology of the human teacher and the robot system. State-of-the-art robot hands are usually larger than the human hand, have a different geometry, e.g. thicker fingers, and less degrees-of-freedom, see Figure 4.14. The result is that fingers cannot be placed at the exact same locations as the human fingers, e.g. touching each other, or the wrist has to be placed at a different pose to place the fingers at the same location.

We define the correspondence problem similar to Skoglund et al. [97] but emphasize the importance of producing robot motions, which are semantically equivalent to the demonstrated human motions, see Definition 4.

Definition 4 (Correspondence Problem) *The correspondence problem describes the problem to transfer motions from one manipulator to semantically equivalent motions of a second manipulator with differences in geometry, kinematics and/or dynamics.*

In related work, manually engineered mappings from human fingertip and wrist motion to the robot finger and wrist motion are defined [100]. The disadvantage is that a fixed, manually engineered mapping is task-dependent since the robot fingers and wrist might have to move in a slightly different way to achieve different goals, e.g. rotate a bottle cap with flexed fingers or rotate a screw with outstretched fingers.

In our case, the semantics are encoded into constraints restricting the motion of the manipulated object, e.g. when opening a bottle with the fingertips, the goal of the task is that the cap rotates

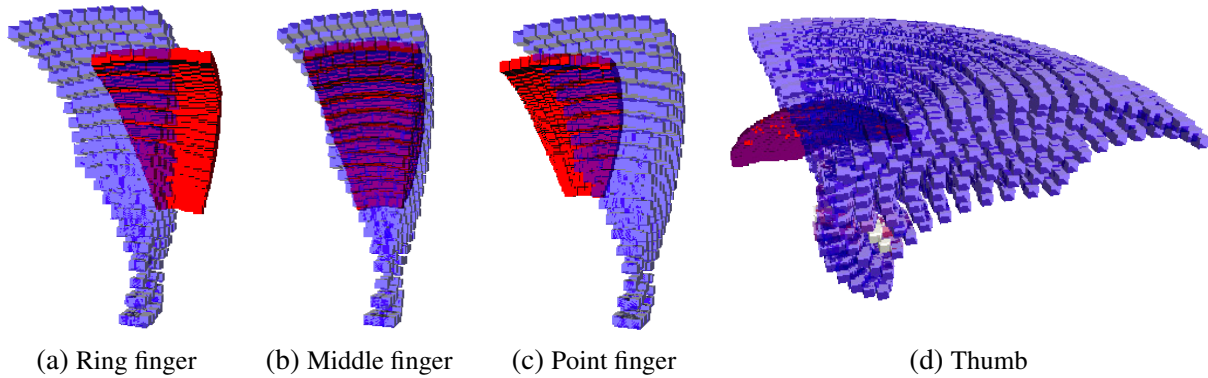


Figure 4.15.: Discretized subset of the workspaces of the human (red) and robot (blue) fingers.

and moves upwards. Based on this observation, we relax learned constraints, which restrict the fingertip and hand motion, but do not alter constraints restricting object motion. In principle, we could relax the learned *human* constraints arbitrarily since the planner will search the whole configuration space until a motion of the fingers is found, which produces the same object motion. Due to the high-dimensionality of the planning problem, this will not work for complex fingertip manipulation tasks. Therefore, the goal is to find a minimal relaxation, which allows us to plan a robot motion to produce the same object motion while considering the differences in morphology. We start with an analysis of the differences in the workspace of the robot hand and the workspace of the human hand for a subset of manipulation motions.

4.3.1. Workspace Analysis

One of the main assumptions to learn fingertip manipulation tasks is the availability of an anthropomorphic robot hand, i.e. with independently controllable fingers with at least 3 DOF and at least four fingers including one opposing finger in a human-like arrangement. In this work, we use the Schunk Anthropomorphic hand (SAH) in our experiments. Our goal is to calculate a rough estimate, how much the workspaces differ for a subset of manipulation tasks. We use this estimation to increase the constraint regions of each human constraint and therefore relax the constraints [53].

The basis is a discretization of a subset of the workspaces of the human and robot fingers, see Figure 4.15a to 4.15d. First, the human teacher demonstrates a set of relevant fingertip motions, e.g. pushing, screwing, in the sensory environment and we store the finger joint values. We use a predefined mapping to choose a number of human fingers and assign them to the robot fingers, e.g. for the SAH, we map all fingers except the human pinky to the corresponding robot finger. Figure 4.14 shows the 3D models of the human and robot hand. We determine the minimum l_i and maximum u_i value of each joint i in the set to extract the subset of joint values $[l_i, u_i]^3$ for each finger, which corresponds to the demonstrated fingertip motions. The subset represents a hypercube in joint space. Based on the forward kinematics, i.e. the bone structure in the hand model, see Figure 4.3c, we transform the hypercube into the Cartesian space and discretize it

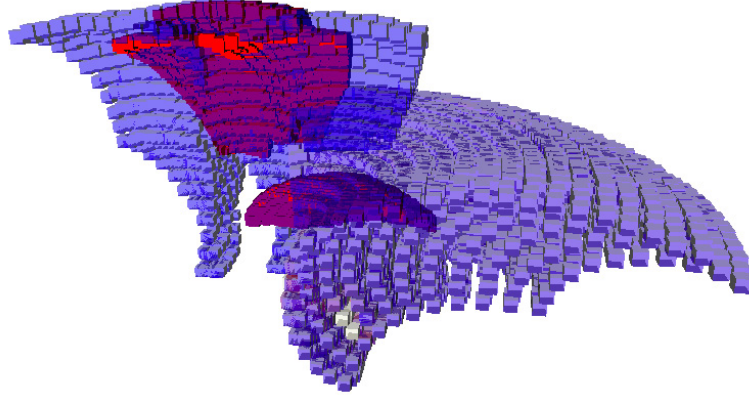


Figure 4.16.: Comparison between discretized subset of the workspaces of the human (red) and robot (blue) fingers. - [53]

resulting in the set \mathcal{H}_j for each finger j . We apply the same technique to get a discretization \mathcal{R}_j of the workspace of the robot fingers but use $l_i = 0$ and $u_i = 1$, i.e. the full configuration space.

Our goal is to find a homogenous transformation of the human workspaces, so that the maximum distance of an arbitrary point in the human workspace to the robot workspace is minimized, i.e. we maximize the overlap. The motivation to use a homogenous transformation is that the result can be mapped to a coordinate frame, which is compatible to the constraint definition. If the differences in kinematics between the human and robot hand are negligible the homogenous transformation will be zero. Let $H(x, y, z, rx, ry, rz)$ be the homogenous transformation matrix with translation vector $(x, y, z)^T$ and rotation matrix corresponding to the scaled-axis representation $(rx, ry, rz)^T$. We solve the following 5D-optimization problem with parameters x, y, z, rx, ry using Rosenbrock [86]:

$$\max_j \left(\max_{x \in F_j \setminus \mathcal{R}_j} \left(\min_{y \in \mathcal{R}_j} \|x - y\| \right) \right) \xrightarrow{x, y, z, rx, ry} 0 \quad (4.4)$$

$$\text{with } F_j = H(x, y, z, rx, ry, 0) \cdot \mathcal{H}_j \quad (4.5)$$

The result is a single homogenous transformation, which minimizes the maximum error of all finger workspaces. We store the homogenous transformation along with the errors ε_j of each finger workspace. The homogenous transformation transforms the human finger workspaces into the robot finger workspaces, see Figure 4.16.

4.3.2. Constraint Relaxation

The error ε_j of a finger workspace represents the maximum distance of a point in the workspace of the human finger, which is not in the robot workspace, to the latter. In the worst case, the human teacher demonstrates only points with maximum distance as the basis to learn constraints for the fingertip positions. In this case, the robot will not be able to place its fingers at the demonstrated positions due to its kinematic restrictions. Additionally, the human teacher can place two fingers in

contact to each other on an object. Since the robot fingers are thicker, the robot will not be able to reach the fingertip positions. Assuming that the fingertips can be approximated by a sphere around the fingertip coordinate frame, we calculate the difference in thickness δ as the difference of both radii. We increase the constraint regions of all learned fingertip constraints by $\varepsilon_j + \delta$ to overcome both described problems.

In principle, we can tessellate the constraint region R into a 3D polytope and enlarge the region using standard algorithms from computer vision. The main disadvantage is that random samples cannot be drawn efficiently from 3D polytopes, which is one of the four characteristics defined in Section 3.1.3. Another approach, e.g. described by Himmelstein [42, p. 33ff], is to draw a sample r from the border ∂R of the region, i.e. $r \sim \mathcal{U}(\partial R)$, create a sphere $\mathcal{B}(r, \varepsilon_j + \delta)$ around r with radius $\varepsilon_j + \delta$ and sample from it. Finally, calculate the parameters of the region R' using the (extensive) set of samples. The advantage is flexibility, i.e. the approach can be applied to all regions, where ∂R can be calculated efficiently. Since this is true for all defined region types, we follow this approach to relax constraints. The disadvantage is that a large number of samples is required, leading to a high constant in the optimization process.

In general, constraint relaxation introduces a fifth characteristic for constraint regions:

5. Enlargement by $\varepsilon > 0$: calculate $\arg \min_{R'} \mu(R')$ with $\forall r \in R : \mathcal{B}(r, \varepsilon) \subseteq R'$

The characteristic ensures that the constraint region can be enlarged by ε .

4.3.3. Discussion

In this work, a mapping from human to robot finger motions was defined based on the Cartesian finger tip positions. The advantage is that similar contact points can be generated with the robot hand, which we assume more relevant in order to reproduce the observed object motion than reproducing the hand configuration. The disadvantage is that only hand gestures can be learned which are trained in the grasp classifier.

The mapping is done implicitly by relaxing finger and hand constraints, which increases the search space for a robot motion in the constraint-based motion planner. The advantage is that the geometry, kinematics and dynamics of the robot hand can be considered in the constraint-based motion planner to generate finger and hand motions, which are similar to the human finger and hand motions and reproduce the observed object motions exactly. The disadvantage is an increase in planning time, which will be reduced by specializing the finger and hand constraints to the robot hand, see Section 4.7.

4.4. Demonstration-based Generalization of Planning Models

The pour-in example in Figure 4.13 on page 102 shows that a large number of constraints is usually generated in the initial planning model, even for simple tasks. The main reason is that the defined constraint sets make only assumptions about which objects and manipulators play an important

role in the manipulation task based on the measured sensor data. The constraint sets do not specify which coordinate frames or constraints are relevant to a given task. In manipulation tasks, usually only a small number of reference frames has to be considered, e.g. Pastor et al. [83] use a single, manually defined reference coordinate frame to describe a pool stroke or how to use chopsticks and Prats et al. [84] show that a single, manually defined Task Frame is sufficient to describe a large number of manipulation tasks. Based on this observation, we assume that only a small number of constraints are relevant to the task and the robot has to be able to resolve this kind of ambiguity.

In machine learning, Bengio et al. introduced the paradigm of *Curriculum learning* [7]. The idea is to sort training data according to complexity, train a classifier with the least complex training data and refine the classifier using training examples with higher complexity.

We adapt this idea and let the human teacher demonstrate two sets of training examples. In the first set, the human teacher demonstrates the manipulation task with a small set of objects and tries to cover all important aspects of the task. In the second set, demonstrations with different objects and more complex motions, e.g. to avoid local obstacles, are encouraged. We generate the initial planning model based on the first set of demonstrations. The result is an overspecialized planning model for the manipulation task, which can be executed with the same objects in the same object arrangements. Generalization is limited due to the small number of training examples and large number of irrelevant constraints. The initial planning model $(\mathcal{X}, \Upsilon_{node}, \Psi_{node}, \Upsilon_{arc}, \Psi_{arc}, \theta)$ is linear. We segment the second set of demonstrations and reject demonstrations, which are incompatible with the first set of demonstrations.

4.4.1. Removing Inconsistent Constraints

The second set of demonstrations is used to identify and remove inconsistent constraints, i.e. constraints, which were learned on the first set, but are violated in the second set [52]. Since noise in the training data has to be considered explicitly, a constraint $\tau = (e_1, \dots, e_m, f, R)$ is only removed, if the distance $d_{\mathcal{M}}(\tau, \theta) > \delta_f$ on more than 50% of the segment. The distance thresholds δ_f for position, orientation, direction and force constraints have to be determined empirically. We measured the standard deviation σ and set the thresholds to 3σ guaranteeing that 99.7% of all data points are captured on average. The constraints $\Psi_{node}(i)$ of node i will be reduced using the segment (t_i^{min}, t_i^{max}) and arc constraints $\Psi_{arc}(i-1, i)$ using the segment $(t_{i-1}^{min}, t_i^{max})$.

In order to increase usability of the PbD system, the teacher gets visual feedback about which constraints are ought to be removed by showing the constraint regions and is encouraged to deselect constraints, which shouldn't be removed. If the human teacher deselects a constraint, it should have been obeyed on the segment and we (re-)parameterize the constraint region using the current demonstration and all demonstrations of the first set. Additionally, usability is increased by rejecting demonstrations, which lead to an empty node constraint set $\Psi_{node}(i)$. Finally the human teacher can run the 3D simulation on the current object setup to test the execution of the robot.

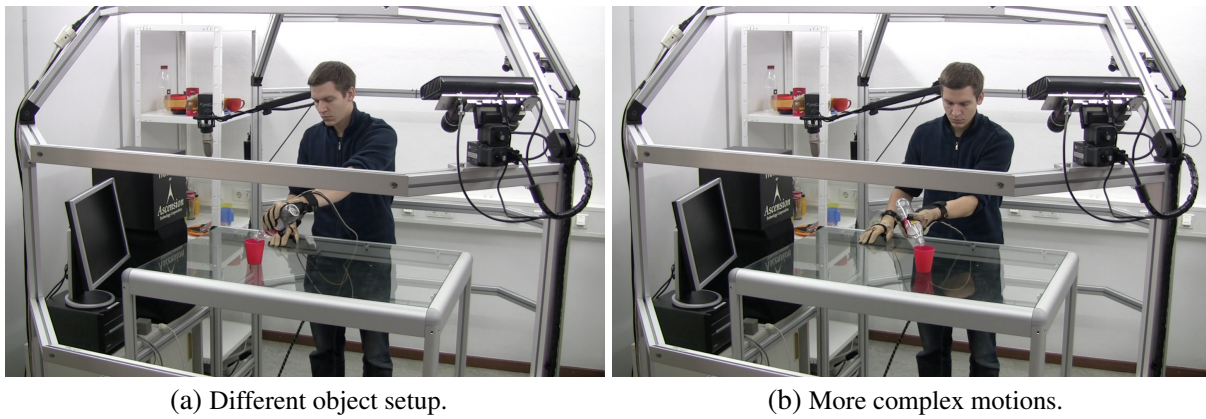


Figure 4.17.: Two types of human demonstrations are shown. (a) shows a different object setup compared to Figure 4.6 on page 94. The cup is placed farther away from the bottle. (b) shows a more complex motion, in which the wrist is rotated into the opposite direction while pouring in.

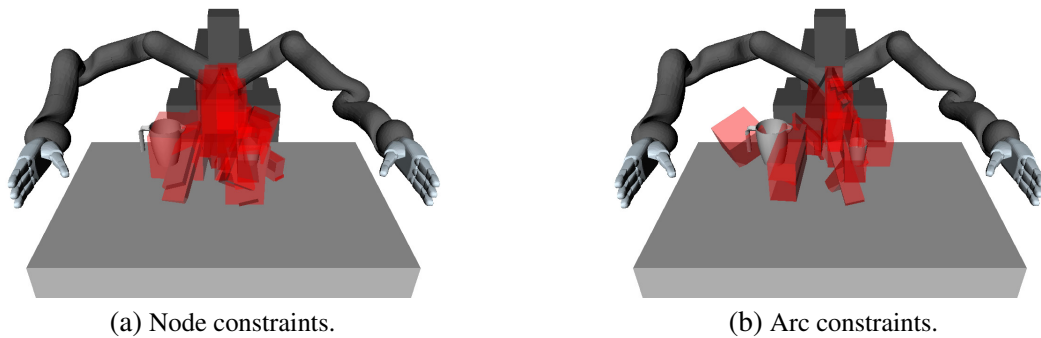


Figure 4.18.: Visualization of reduced number of constraints in the pour-in task after demonstration-based generalization compared to Figure 4.13 on page 102. - [50]

Figure 4.17 shows a second set of demonstrations for the pour-in task. In Figure 4.17a, the cup is placed farther away from the bottle on the table. Constraints, which restrict the pose of the bottle relative to the cup and vice versa at the beginning of the task, are inconsistent and will be removed. The resulting planning model can be executed in a more flexible way with different cup and bottle poses on the table. Figure 4.17b shows a more complex motion, in which the bottle is rotated in a different way and the bottle is placed on a different side of the cup. Constraints, which restrict the bottle bottom relative to the cup are inconsistent and will be removed. Relevant constraints, e.g. that the bottle opening has to stay above the cup opening, remain leading to a smaller number of constraints, see Figure 4.18, and improving flexibility.

4.4.2. Discussion

In the learning process, we consider a large number of automatically generated constraints to learn the preliminary planning model. In general, a subset of constraints will be irrelevant to the task,

which limits flexibility of the learned planning model. In this section, we remove constraints, which are inconsistent with a second set of demonstrations. The main assumptions are that in the second set of demonstrations, all relevant constraints are obeyed but a number of irrelevant constraints is not.

If the first assumption is violated a relevant constraint might be removed and the robot is not able to reproduce the task. In order to ensure that the first assumption is valid the human teacher requires experience in the learning system, e.g. based on a training course. Knowledge of the implementation of the machine learning algorithms is not required. The human teacher is supported in the learning process by the feedback in 3D visualization and by executing the planning model in simulation on the robot.

The second assumption reflects the goal to identify and remove as many as possible irrelevant constraints with a limited number of demonstrations. If this assumption is violated no irrelevant constraints will be removed but the learned planning model allows the robot to reproduce the task in the environments, which were used during demonstration.

4.5. Robot-Test-based Generalization of Planning Models

Demonstration-based generalization, see Section 4.4, relies on an experienced teacher using different objects and multiple demonstrations to show the important characteristics of a manipulation task. The required experience as well as time to generate a large number of demonstrations is often not available in the execution environment, e.g. an industrial or household environment. Additionally, the human teacher has to anticipate the future field of application of the robot, e.g. if the planning model has to be applied to a single set of objects less training data and less flexibility is required. In industrial applications, the field of applications is usually well-defined. In a household environment, applications arise naturally, for instance when the robot was taught how to place a bottle in the fridge door, some of the learned constraints encode task-irrelevant geometric information about the environment. When the geometry changes, e.g. we rearrange the fridge shelves, these constraints will not be obeyed and the planning model cannot be applied. In order to cope with this new application, automatic means to identify irrelevant constraints without the need for much user interaction, i.e. a large number of demonstrations, is required.

If a planning model cannot be executed in a new application, a number of constraints will be inconsistent, i.e. they cannot be obeyed at the same time, or superfluous, i.e. they reduce the search space artificially and no solution can be found. In the motion planner, these inconsistencies can be identified since the projection of a configuration to the constraint manifold will fail. We gather statistics about which constraints are invalid after the projection in the motion planner. These statistics form the basis to identify irrelevant constraints and adapt the planning model. Since we have no prior knowledge about which of the inconsistent constraints are irrelevant, different hypotheses will be evaluated. We define an objective function and run a combinatorial optimization process to identify a maximum subset of constraints, which admits a successful solution in the

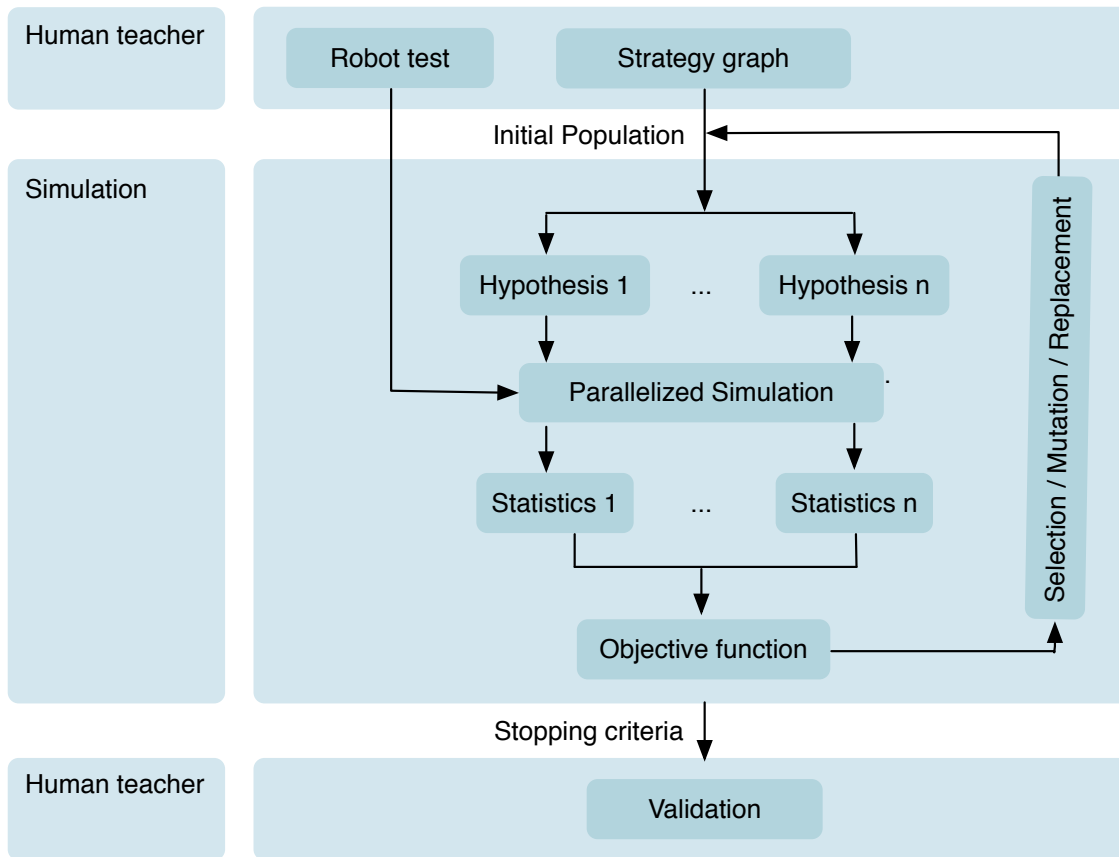


Figure 4.19.: Overview of the robot-test-based generalization process. The human teacher defines a test the robot has to solve using the given strategy graph. Multiple hypotheses about irrelevant constraints are generated. Each hypothesis is planned and run in simulation to generate statistics about constraint inconsistencies. The statistics are used to rate the hypotheses and select, mutate and recombine the set of hypotheses. If the optimization process converges the human teacher validates the result.

novel application case [49]. The human teacher is not involved in the time-demanding optimization process allowing for parallelization and massive use of simulation and planning to adapt multiple hypotheses at the same time. In the end, multiple solutions will be available and we use the objective function to sort the results. Since it is not guaranteed that the assumptions in the objective function match to the semantics of the task, the human teacher validates the solutions by looking at the execution in the simulation environment and chooses the first semantically valid solution. In the experiments, the human teacher picked the 1.1th solution on average showing that the assumptions, on which the objective function is based, are valid for a large number of manipulation tasks and user interaction is reduced significantly. Figure 4.19 gives a short overview of the process.

We represent new applications as robot tests, e.g. an environment with a specific object arrangement, see Section 4.5.1. The combinatorial optimization problem is solved using an evolutionary algorithm, which is described in Section 4.5.2.

In Figure 4.20, a strategy graph to grasp a bottle and place it in a fridge door is shown. The constraints in the node *End* restrict the placement of the bottle in the fridge door. Three constraints

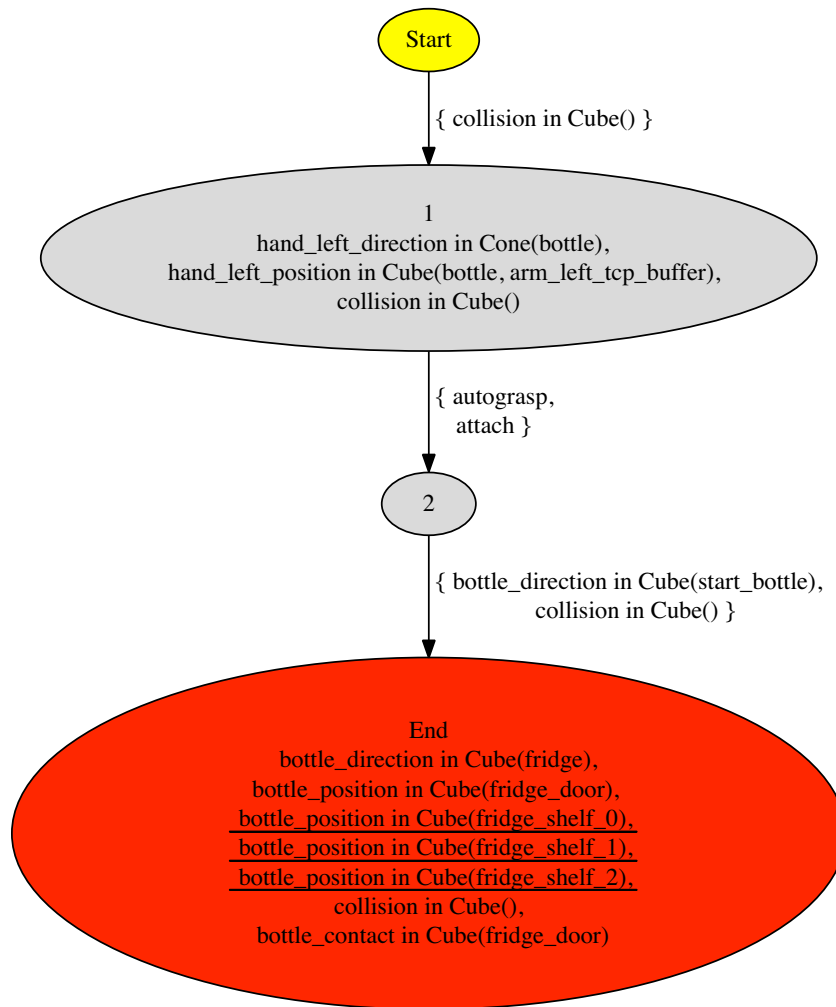


Figure 4.20.: Planning model to grasp a bottle and place it in a fridge door containing irrelevant task constraints (underlined).

restrict the position of the bottle relative to the three shelves in the fridge and one constraint restricts it relative to the fridge door. Since the angle of the fridge door was fixed in all demonstrations, this ambiguity cannot be resolved in the *demonstration-based generalization*. Figure 4.21 visualizes the constraints in an environment with a different door angle. Since the constraint relative to the fridge door adapts automatically to the door angle but the constraints relative to the shelves are fixed, the constraints do not overlap and are inconsistent. The constraint-based motion planner fails to find configurations, in which all constraints are obeyed.

4.5.1. Robot-Test Specification

A robot-test consists of a set of objects, the pose of each object in the environment and a mapping of object names to the parameters in the planning model, i.e. which object in the test corresponds to which object in the learning process. Optionally, a position and orientation constraint for each object is included, which has to be obeyed at the end of the robot test. In the fridge example, the

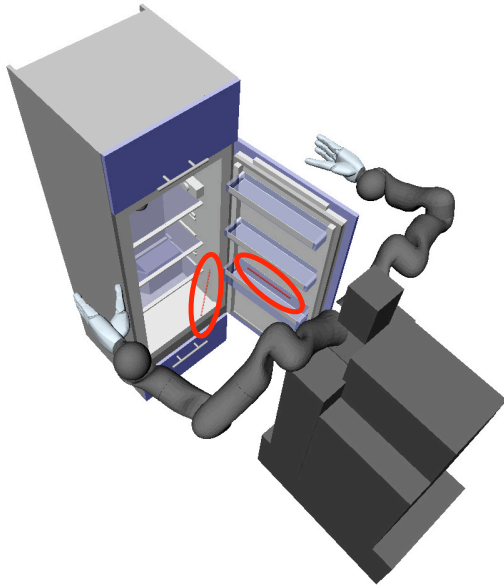


Figure 4.21.: Visualization of the four position constraints of node *End* in the planning model in Figure 4.20 using a different door angle. The constraints relative to the fridge shelves (left circle) and the constraints relative to the fridge door (right circle) are inconsistent. - [49]



Figure 4.22.: Four different robot tests, each with a different door angle. - [49]

test consists of four different door poses, see Figure 4.22, which the robot is not able to solve with the planning model in Figure 4.20 due to inconsistent constraints, see Figure 4.21.

In the PbD system, the human teacher generates robot tests by placing a set of objects in the sensory environment and defining the object mapping [49]. The object poses and constraints will be generated automatically using stereo vision-based object localization and the algorithm to learn constraint parameters in Section 4.2.4. Additionally, robot tests can be defined by modifying objects in the 3D visualization to overcome the limitations of the sensory environment and state-of-the-art perception algorithms.

4.5.2. Evolutionary Algorithm

In order to generalize the planning model, an evolutionary algorithm will be used to determine a maximum subset of constraints [49], which can be included in the planning model to be able to solve a number of robot tests.

State and Population

The evolutionary algorithm sustains a number of hypotheses about irrelevant constraints, called population, which are constantly adapted in the optimization process. Each hypothesis is represented as a state s , which is defined as a binary vector $s \in \{0, 1\}^n$, where n is the total number of constraints. The i th bit will be 1 if and only if the corresponding constraint τ should be included in

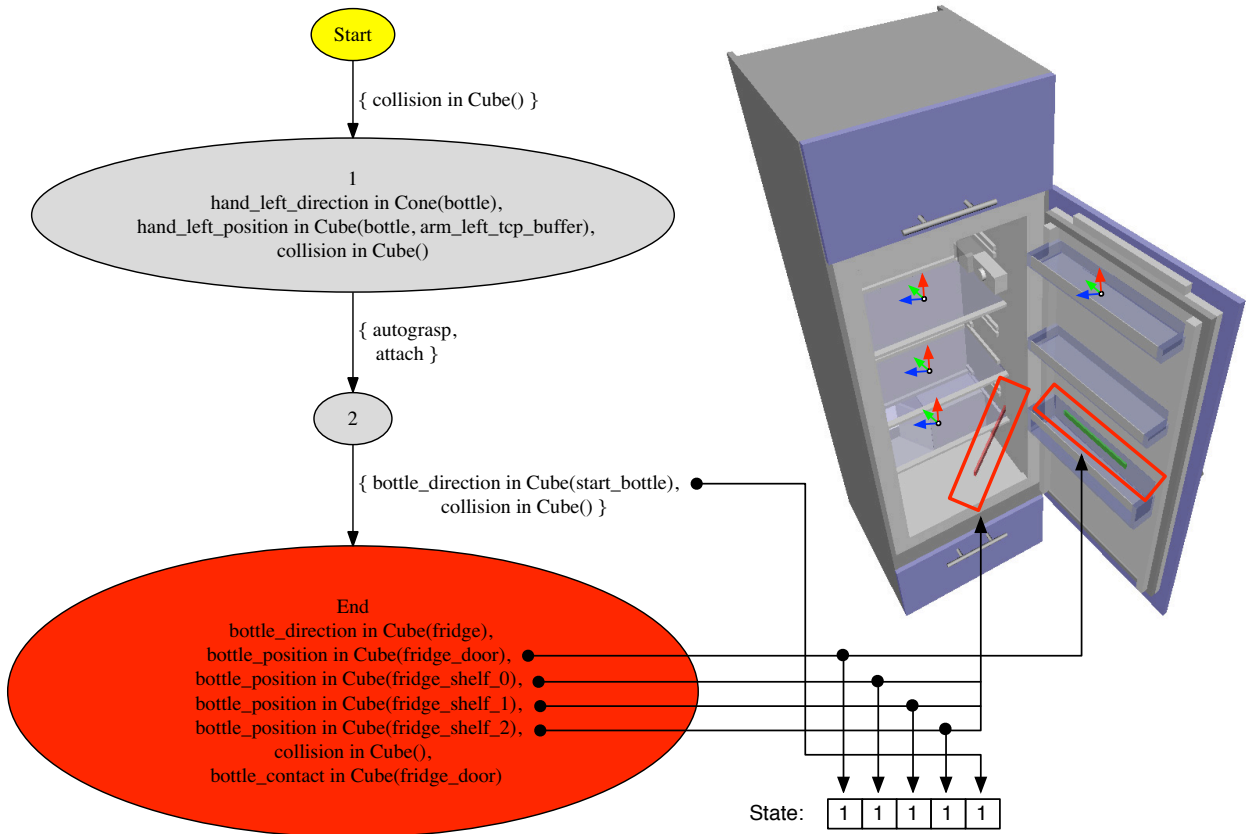


Figure 4.23.: State consisting of 5 bits, one for each constraint that can be deactivated in the planning model in Figure 4.20.

the planning model. In general, not all constraints can be deactivated due to restrictions imposed by the constraint-based motion planner. In order to generate random goal configurations, in which the constraints of a given node are obeyed, we use inverse kinematics and projection techniques, see Section 3.4.2. Inverse kinematics are applied to a random pose for each object, which is referenced by a coordinate frame in the constraints of the given node. The random pose is generated randomly based on a pair of position and orientation or position and direction constraints. Let a planning model be *valid*, if at least one position and one orientation or direction constraint for each object is present in all nodes. A position, orientation or direction constraint will be considered in the state, if the planning model, which is obtained after deactivation of the constraint, is valid.

In Figure 4.23, the state representing the planning model in Figure 4.20 is visualized. The first bit represents the position constraint *bottle_position* in the node *End*. The last bit corresponds to the direction constraint on arc $(2, End)$. The remaining bits correspond to the constraints relative to *fridge_shelf_i*, $i = 0, 1, 2$, in the node *End*, which restrict the bottle relatively to the shelves.

Inconsistency Statistics

For a given state s , statistics about the inconsistency of constraints are gathered, which prevent the constraint-based motion planner to find a solution for the test. The strategy graph is planned sequentially, from one goal to the next, e.g. $(S, 1)$, $(1, 2)$ and $(2, End)$ in Figure 4.20. When

planning $(i - 1, i)$, the node constraints of i , the constraints of the ingoing arc $(i - 1, i)$ and the constraints of the outgoing arc $(i, i + 1)$ are used to generate goal configurations for the robot system, in which all constraints are obeyed. A motion from the start configuration to one of the goal configurations is planned, in which all constraints of the arc $(i - 1, i)$ are obeyed. If two constraints are inconsistent, the intersection will be empty and the planner will not be able to find a configuration, in which both constraints are obeyed. For each constraint τ , $success(\tau)$ counts the number of times, in which τ is obeyed, and $failure(\tau)$ the number of times, in which τ is violated. The failure frequency

$$freq(\tau) = \frac{failure(\tau)}{failure(\tau) + success(\tau)} \quad (4.6)$$

is used as an indicator for the inconsistency of τ with a second constraint. Figure 4.24 shows the constraint statistics for the state in Figure 4.23. The position constraints relative to the fridge shelves and relative to the fridge door cannot be obeyed at the same time. The generation of goal configuration fails and the projection result will be a configuration, in which both constraints are violated. Therefore, the failure frequencies are 100%. In contrast to the position constraints, the direction constraint is obeyed in the resulting configuration in 99%. The failure frequency of 1% has to be accounted to the random start configuration of the projection techniques.

Fitness

The definition of the objective function has a significant impact on convergence of the evolutionary algorithm. Since the goal is to find a maximum subset of constraints, which admits a successful solution to the robot-test, we have to ensure that progress in the planning process is made. We define $progress(s)$ as the index ≥ 0 of the last subgoal in the (linear) strategy graph, which was successfully reached in the planning process. By maximizing $progress(s)$ we ensure that a set of constraints is found, which can be used to find a solution to the planning problem. Since the constraints describing the (sub-)goals are also part of the optimization problem, finding a solution to the planning problem is not a sufficient but a necessary condition.

A high value of $progress(s)$ indicates that the planning process was successful or aborted near the end. By maximizing $progress(s)$, an increase of the rate of convergence could be observed experimentally, since the optimization process will focus on states s close to satisfying the necessary condition.

In order to find a maximum subset, we have to include the fraction of active constraints into the objective function. A simple choice would be the total number of constraints in the planning model, i.e. the number of 1s in the state s . This choice has proven inefficient, since the number of constraints restricting a given frame is not equally distributed and a lot of constraints exist, which represent similar dependencies. For instance, orientation constraints are often equivalent when they restrict the orientation of a coordinate frame relative to different coordinate frames of the same rigid body, since they often have the same orientation.

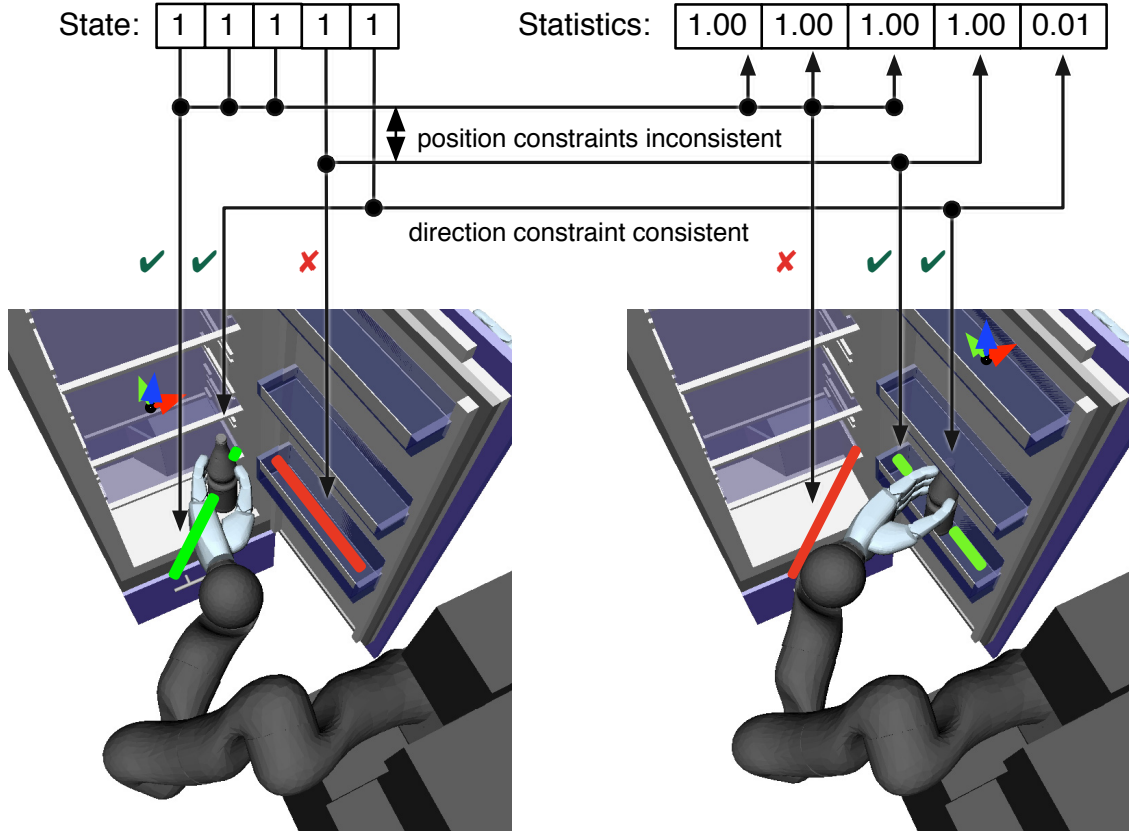


Figure 4.24.: Visualization of statistics about inconsistency of the constraints. The position constraints relative to the fridge shelves (state bit 1, 2, 3) and relative to the fridge door (state bit 4) cannot be obeyed at the same time (pictures). The result is a failure probability of 1.0 in the statistics. The direction constraint (state bit 5), which restricts the bottle to stay upright, can be obeyed with a failure probability of 0.01.

We normalize the number of constraints by calculating the fraction of active constraints with the same type f and restricted coordinate frame e_1 . Let $Partition(\Psi)$ be a partition of the constraint set Ψ into sets with equal constraint type f and equal constrained coordinate frame e_1 . We set ${}^s|X|_1 = |\{x \in X \mid s(x) = 1\}|$ and define the fraction of active constraints as

$$active(s, X) = \frac{{}^s|X|_1 - 1}{|X| - 1} \quad (4.7)$$

setting $\frac{0}{0} = 0$. We weight node and arc constraints differently to reflect the different importance during motion planning. In the previous subsection, we motivated that arc constraints are used three times as often as node constraints. Based on this observation we define

$$total(s) = \sum_{\substack{X \in Partition(\Psi_{arc}(i,j)) \\ {}^s|X|_1 > 0}} (3 + active(s, X)) + \sum_{\substack{Partition \in P(\Psi_{node}(i)) \\ {}^s|Y|_1 > 0}} (1 + active(s, Y)) \quad (4.8)$$

in order to include the normalized fraction of active constraints in the objective function. If X is non-empty, $active(s, X)$ is in $[0, 1)$. Since the base fitness of each set X with $|X|_1^s > 0$ is either 1 or 3, see Equation 4.8, states s with active constraints of different constraint type or coordinate frame e_1 are assigned a higher fitness.

Finally, we define fitness $fitness(s)$ as the sum of the planning progress and the normalized fraction of active constraints:

$$fitness(s) = progress(s) + \frac{total(s)}{total\left(\left(1, \dots, 1\right)^T\right)} \quad (4.9)$$

Mutation

The mutation operator mutates a given state s based on $progress(s)$ and the planning statistics. We define two cases. In the first case, the motion planner found a solution, which is determined by checking if $progress(s)$ is equal to $|\mathcal{X}| - 1$, i.e. all nodes \mathcal{X} were considered in the motion planning process. Since a solution was found, the inconsistency statistics are irrelevant. In order to find a maximum of constraints, which admits a successful solution to the robot test, we add a random constraint to the state. First, a random constraint τ is chosen. If τ is included in s and belongs to a node, it will be removed and an alternative constraint of the same node and same type is added to s , which forces the optimization algorithm to explore sets of constraints with alternative goal descriptions. Otherwise, τ is not included in s and will be added in order to maximize the number of constraints.

In the second case, the motion planner did not find a solution, i.e. $progress(s)$ was smaller $|\mathcal{X}| - 1$. We remove inconsistent constraints randomly based on the inconsistency statistics to quickly find states satisfying the necessary condition. Each constraint τ with $s(\tau) = 1$ will be set to 0 with probability equal to their failure frequency $freq(\tau)$. We draw a sample u from the uniform distribution $\mathcal{U}(0, 1)$

$$u \sim \mathcal{U}(0, 1) \quad (4.10)$$

and set the constraint to 0 if u is smaller the failure frequency:

$$u \leq freq(\tau) \Rightarrow s(\tau) = 0 \quad (4.11)$$

If the resulting planning model is not *valid*, τ will be a node constraint and none of the node constraints will be included in s . A random constraint of the same node and the same type will be picked randomly and added to s .

If no constraint is removed from s , failure frequency of all constraints is probably low but the planning process failed nevertheless, which indicates that the inconsistency of some constraints

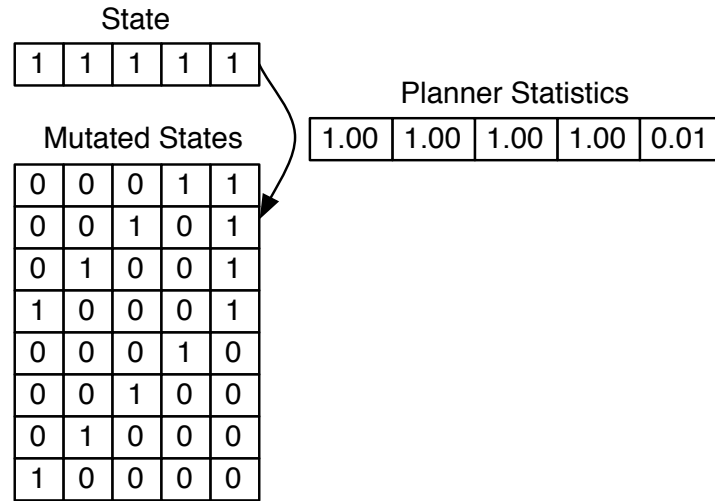


Figure 4.25.: Result of the mutation operator applied to the state in Figure 4.23 using the planning statistics in Figure 4.24. Eight different states can be generated with different probability.

could not be recognized. In this case, we deactivate a single, random constraint τ with $s(\tau) = 1$, if the resulting planning model is *valid*.

In the example in Figure 4.21 on page 113 a bottle has to be placed in a fridge door. A contact constraint ensures that the bottle has contact with the fridge door. Three position constraints restrict the placement of the bottle relative to the fridge shelves and one position constraint restricts the bottle pose relative to the fridge door. Let the state s contain the first three position constraints but not the fourth. Based on the constraints, the motion planner generates a goal configurations where the bottle is placed in the free space in front of the fridge door. The three position constraints are obeyed, i.e. failure frequencies are 0%, but the contact constraint is violated. Therefore, the inconsistency statistics do not provide sufficient information and we deactivate a single, random constraint. After a few iterations, the three position constraints are deactivated and the fourth is activated randomly, leading to the correct result.

Figure 4.25 shows an example using the state in Figure 4.23 and statistics in Figure 4.24. Based on the failure frequency of 100% all position constraints are deactivated. Since the resulting planning model is invalid, a random position constraint is activated. The direction constraint is deactivated with a probability of 1%. In total, eight mutated states are generated.

Initial population

In the evolutionary algorithm, the population, i.e. a set of states, will be mutated iteratively based on the planning statistics to find a state with (locally) maximum fitness. In the initial population, all states are set to 1, i.e. all constraints will be included in the planning model at the beginning.

Starting with the most restricting state has several advantages. The most restricting state is unique in contrast to the least restricting, since it is possible that 0 is not a *valid* planning model. Probability is high that the planning process fails in the beginning. In this case, inconsistency

statistics are generated and provide valuable information to deactivate inconsistent constraints efficiently. In a least restricting state, probability is high that the planning process succeeds but constraints can only be added randomly to find a (local) maximum since the inconsistency constraints provide no relevant information. Therefore, we rely on the most restricting state to achieve a higher convergence rate.

Optimization Algorithm

The goal of the generalization algorithm is to find a state s , which maximizes the fitness function $fitness(s)$. Since the state space is discrete, the optimization problem is combinatorial. The objective function, $fitness(s)$, is computationally expensive to evaluate, since a constraint-based motion planning process has to be started to generate the inconsistency statistics, but can be parallelized easily. We applied evolutionary computation [36] using the *evolving objects* [61] library to solve the combinatorial optimization problem. The algorithm starts with calculating the fitness of the initial population. The algorithm now iterates the following steps: evaluate the stopping criteria, select a set of genitors, apply mutation and crossover operators to the genitors to produce the offspring, calculate the fitness of the offspring and replace some parents by some offspring to obtain the new parents. A detailed description of all steps can be found in [36].

For each state a parallel process is started, in which the constraint-based motion planner is executed to calculate the inconsistency constraints. In the experiments, the size of the population and the offspring was fixed at 24, which corresponds to the number of parallel processes. The probability of mutation was set to 0.5, no crossover was applied. The genitors were selected one after another in a random order, which is called the *Unordered Sequential* scheme [36]. Following the *Plus* replacement method, the new population consists of the best individuals in the set of offspring and parents. The algorithm is stopped, if the best state does not change in 20 iterations or a time threshold is reached.

The result of the optimization algorithm is a set of states s . We remove states, in which the motion planning process failed, i.e. $progress(s) < |\mathcal{X}| - 1$ and sort the remaining states by $fitness(s)$. In general, we can not guarantee that the state with highest fitness is the solution a human teacher would select due to several reasons, e.g.:

1. The computation time is limited and the optimization algorithm might not have converged.
2. The assumptions of the objective function might be violated.
3. The objective function might not cover the semantics, e.g. implicit constraints, of the task.

In contrast to the time-demanding optimization algorithm, which is fully automatic, user-interaction by the human teacher is therefore required to verify the optimization result. Beginning with the state with highest fitness, the robot solves the robot test using the corresponding planning model and shows the execution in simulation to the human teacher, who assigns *true* or *false* to

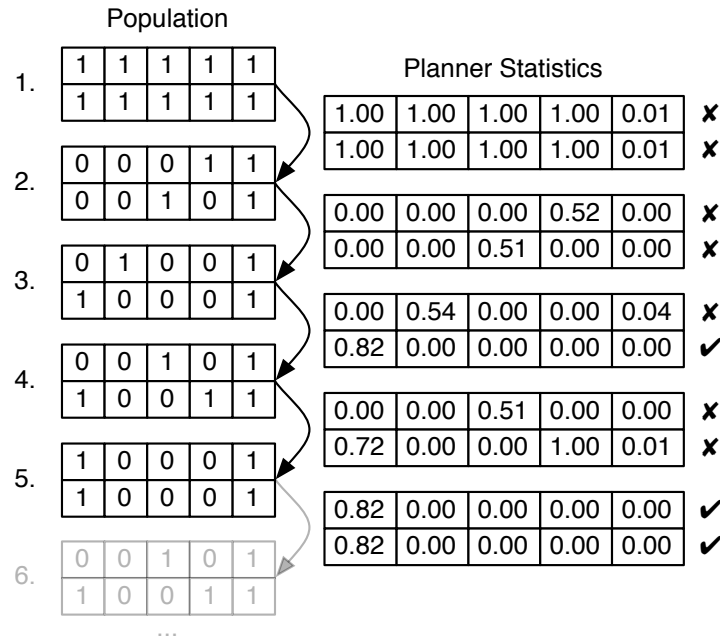


Figure 4.26.: Example of evolutionary algorithm applied to the example in Figure 4.20 using 2 states. The annotations of the statistics show if the planning process was successful.- [49]

the state. If a state is *true*, the process will be stopped. In the experiments, the human teacher stopped the process after looking at only 1.1 states on average. Therefore, the assumption to find a maximum subset of the learned constraints, which admits a successful solution to the robot test, has proven reasonable in the experiments.

In Figure 4.26, an example of the evolutionary algorithm using the planning model in Figure 4.20 and two individuals is shown. Based on the planner statistics, the individuals will be mutated until in the third step, a valid solution is found. In the fourth step, a constraint is added to the best state in order to maximize the number of constraints. Planning fails and the constraint is removed in the fifth step. Since no better state will be found after the next 20 iterations, the algorithm stops after the 23rd iteration.

Figure 4.27 shows the set of node and arc constraints after applying the robot-test-based generalization to the pour-in task.

4.5.3. Discussion

In general, a large number of constraints is present in learned planning models. The resulting combinatorial optimization problem is high-dimensional and, in the worst-case, cannot be solved efficiently. Nevertheless, problems with more than 100 constraints can be solved in the matter of hours using massive parallelization. The reason is that the number of constraints restricting a given coordinate frame is not equally distributed and clusters of constraints exist, which represent similar dependencies between coordinate frames, see Section 4.5.2. The consequence is that the

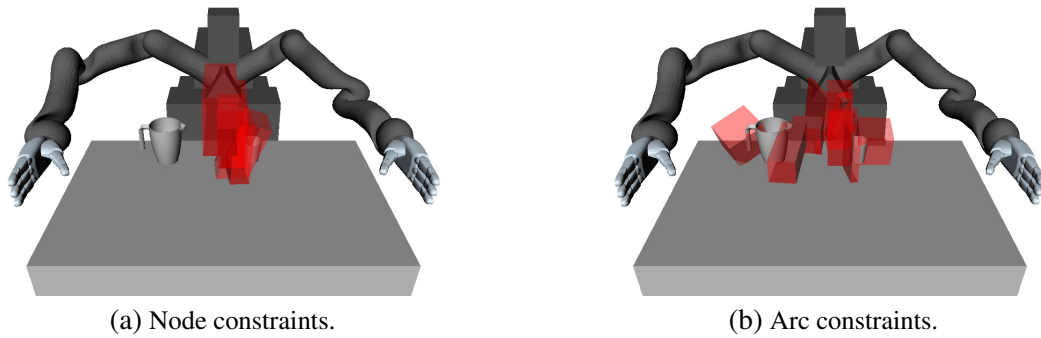


Figure 4.27.: Visualization of reduced number of constraints in the pour-in task after robot-test-based generalization compared to Figure 4.13 on page 102. - [50]

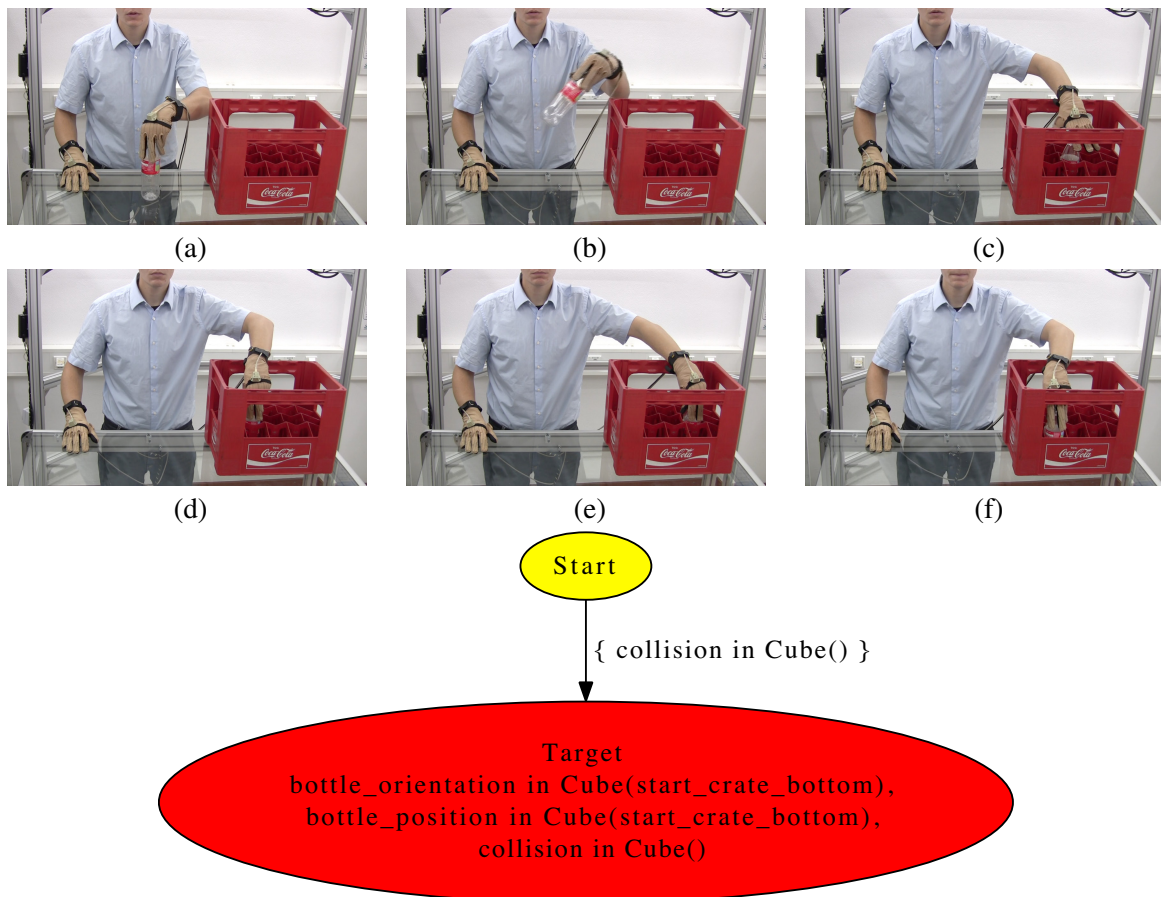
inconsistency statistics are high for all constraints in a cluster leading to the deactivation of a large number of constraints in a single iteration.

Since at most one constraint is activated in each iteration using the mutation operator, computation time depends on the number of constraints in the optimization result. We observed earlier, see Section 4.4, that usually only a small number of constraints is required to describe the motion of a single coordinate frame in typical household manipulation tasks. Based on this assumption, only a small number of constraints has to be activated in the optimization process in the worst case. The computation time depends less on the total number of constraints but on the number of constraints, which represent different dependencies between coordinate frames.

The main advantage of the robot-test-based generalization is that the human teacher is only required to demonstrate a robot test and to verify the optimization result, which requires less experience and less time compared to the demonstration-based generalization. The time-demanding optimization process is fully automatic.

4.6. Efficient Execution of Planning Models

The main motivation to generate robot motions in a goal-directed way using constraint-based motion planning is flexibility. Flexibility is required, when the execution has to be adapted to different objects or the workspace of the robot is restricted, e.g. due to obstacles or self-collisions. The main disadvantage is high planning time due to narrow passages in the configuration space, its high dimensionality and broad constraint manifolds. In manipulation tasks, narrow passages exist naturally in the vicinity of objects. Figure 4.28 show a simple manipulation task: placing a cylinder in a crate. The cylinder has to be placed in one of the pockets of the crate resulting in a narrow passage. The goal constraints, see Figure 4.29, restrict that the cylinder has to be placed in a cube covering the bottom of the crate and the cylinder does not collide with the crate. In the constraint-based motion planner, a collision-free pose to place the cylinder in one of the pockets is found automatically. The constraint generates a flexible constraint manifold, which can be used



(g) Planning model to place a cylinder in a crate. - [55]

Figure 4.28.: Human demonstrations to place a cylinder in a crate (a - c) with different goal poses (c - f), planning model (g).

for crates with identical size but different pocket sizes and positions. For this object, the constraint manifold is unnecessarily large: a high percentage of sampled bottle poses will be in collision with the crate. The same observation applies to finger and hand constraints, which were relaxed to consider the correspondence problem.

We achieve a significant reduction in planning time if we specialize the (general) constraint manifold to the objects in the environment. Figure 4.29 shows an example of the original and specialized goal constraint of the manipulation task in Figure 4.28. The specialized goal constraint reflects the individual pockets of the crate. In the constraint-based motion planner, less bottle poses are generated, which are in collision with the crate, resulting in a significant reduction of planning time.

Computing the constraint specialization based on the object geometry is difficult since different types of constraints are used, e.g. orientation and force constraints, and the kinematics, geometry and dynamics of the robot have to be considered. Instead, we use past experience in solving the planning problem with the given object to incrementally learn refinements of the constraint manifold [55]. These refinements are represented as sequences of ellipsoids and will be called

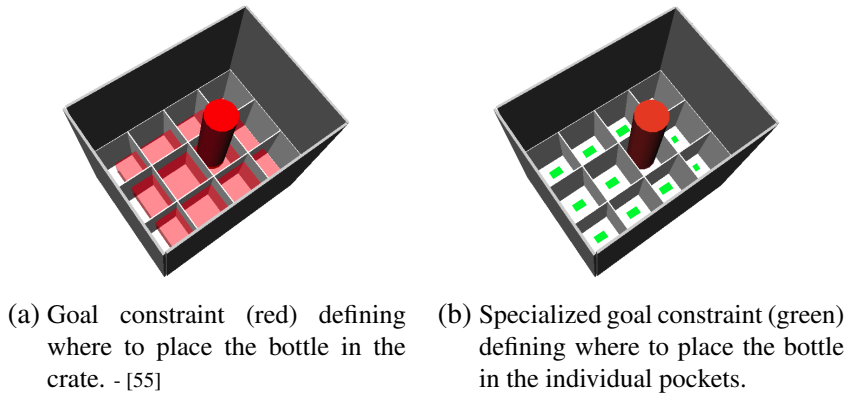


Figure 4.29.: Original and specialized goal constraints of manipulation task in Figure 4.28.

search heuristics. Search heuristics represent bundles of trajectories, which can be executed to achieve the goal of the task. Due to workspace limitations and collisions, it is not possible to execute these trajectories from beginning to end. We use the last part of the trajectory bundle to generate a robot motion in the vicinity of the object using a fast, local control algorithm. In order to reach a configuration, from which this robot motion can be executed, we start a motion planning process and therefore are flexible to obstacles and workspace restrictions. The resulting algorithm is an extension of the standard motion planner, in which search heuristics are used to generate goal configurations in a highly efficient way. In general, a single search heuristic is not flexible enough and multiple search heuristics have to be learned. We investigate an incremental approach, where new search heuristics will be learned, when planning with previously learned search heuristics fails. The section starts with a detailed overview of the approach.

4.6.1. Overview

Search heuristics are learned incrementally when planning with previously learned search heuristics fails and sufficient examples to learn a new search heuristic are available. The following steps are executed each time a planning model has to be executed by the robot [55]:

1. Plan with each search heuristic in a random order until a valid solution is found or a time threshold is met.
2. If no solution was found, plan with the original planning model without search heuristics.
3. Store the planning result in a database.
4. Temporally align results in database and generate clusters with clustering threshold P_d .
5. If a cluster with at least P_c results was found, generate a search heuristic and parameterize each sequence of ellipsoids using P_g Gaussians.
6. If a search heuristic was learned, remove all items from database.

In Section 4.6.2, we define search heuristics. In the beginning, no search heuristics are available and the first step fails automatically. If multiple search heuristics are available, the order in which search heuristics are planned, has to be determined in step one, see Section 4.6.3. Search heuristics are learned based on the planning results in the database. In order to encode the planning results into constraint regions in step five, we cluster the planning results and align the planning results in each cluster temporally, see Section 4.6.4. In order to decrease planning time using the search heuristics, we integrate a fast control algorithm, see Section 4.6.5, and extend the constraint-based motion planner, see Section 4.6.6. The last step ensures that a minimal number of distinctive search heuristics is learned but time increases to generate new search heuristics.

4.6.2. Definition of Search Heuristics

In Section 3.1.3, the constraint region type *sequence of ellipsoids* was defined, which is used as the basis to define search heuristics. For each arc (i, j) in a planning model, a set of search heuristics is learned. Each search heuristic is defined as a set of additional constraints, one for each constraint in the set of node constraints $\Psi_{node}(i)$. Let

$$\tau = (e_1, \dots, e_m, f, R) \in \Psi_{node}(i) \quad (4.12)$$

be the set of constraints of node i . We define a search heuristic of node i as the set of constraints $\Psi_{SH}(i)$, in which the constraint region R is replaced by a constraint region of type *sequence of ellipsoids*:

$$\Psi_{SH}(i) = \{ (e_1, \dots, e_m, f, R_{sequence\ of\ ellipsoids}) \mid (e_1, \dots, e_m, f, R) \in \Psi_{node}(i) \} \quad (4.13)$$

The constraint region will be parameterized based on prior planning results, which is explained in Section 4.6.4.

4.6.3. Selection of Search Heuristics

A random permutation of search heuristics is generated using a discrete probability density function $0 \leq p_i \leq 1, \sum p_i = 1$. We define the following random experiment: draw a search heuristic from the probability distribution, set its probability to zero, normalize the distribution and repeat the process. We define two different types of distributions: a uniform and non-uniform distribution.

The uniform distribution reflects that no a-priori knowledge about the correct choice of search heuristic is available.

In the non-uniform distribution, we draw a search heuristic based on the poses of the objects, which are referenced in the set of node constraints. We generate this kind of knowledge during the execution by storing, which search heuristic could be applied successfully for a specific sets

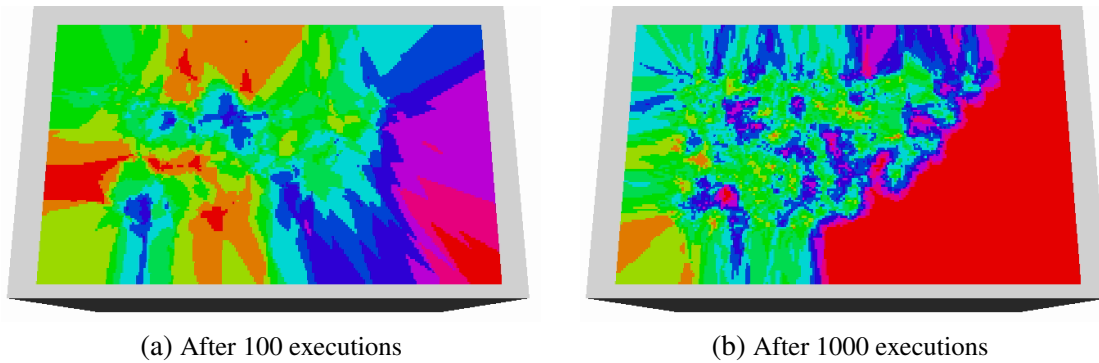


Figure 4.30.: The task is to grasp a bottle on the table with the right arm. The bottle is placed at a random position on the table. Three search heuristics are learned. The color visualizes the discrete probability density function p_1, p_2, p_3 after 100 and 1000 executions. The color is (p_1, p_2, p_3) in RGB. The right arm is not able to grasp the bottle on the far left side of the table (b, red).

of poses of the referenced objects in the planning model. We calculate the k-nearest-neighbors and calculate the histogram, how often a specific search heuristic was chosen. The result is a non-uniform discrete probability distribution. Since it depends on the poses of the referenced objects, it can be interpreted as a map of object arrangements to which a search heuristic can be applied. A similar approach was followed in [111] with the difference that a map of robot poses was generated, where a specific task can be executed. In Figure 4.30, a map for the simple task to grasp a bottle on the table is shown. The number of learned search heuristics as well as the encountered object poses increases over time. Nevertheless, the number of encountered object poses is usually small and the uniform distribution is sufficient.

4.6.4. Temporal Alignment and Clustering

Search heuristics are an abstraction of a set of planning results, i.e. trajectories. The trajectories represent valid solutions to the planning problem for different robot configurations and a different set of object poses, i.e. they avoid local obstacles and all constraints in the planning model are obeyed. In order to use the search heuristic as prior knowledge to speed-up the planning process, these properties have to be maintained during abstraction. Encoding all trajectories into a single search heuristic is insufficient. Figure 4.31 shows a set of planning results in the crate example. The planning results end in different pockets of the crate. If a single search heuristic would be learned, the ellipsoids would span multiple pockets. The constraint-based motion planner would generate configurations, which are in collision with the crate with high probability, and no significant speed-up could be achieved. Therefore, we cluster similar trajectories and generate search heuristics on each cluster.

The main problem is that trajectories have different (temporal) length. If we scale all trajectories to have the same length and ignore the spatial similarity between trajectory points, the resulting search heuristic will be suboptimal and potentially include local obstacles. Instead, we apply Dynamic Time Warping (DTW) [90] to calculate a non-linear temporal mapping between two

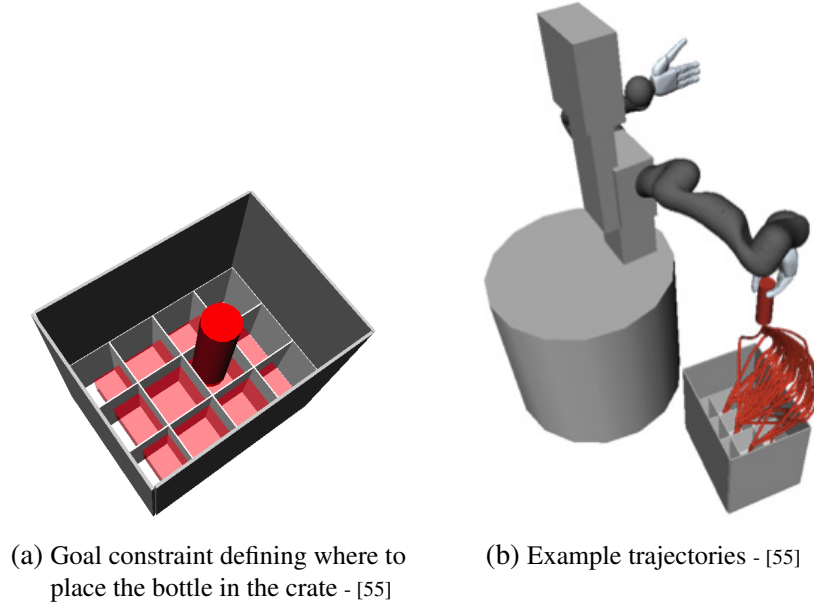


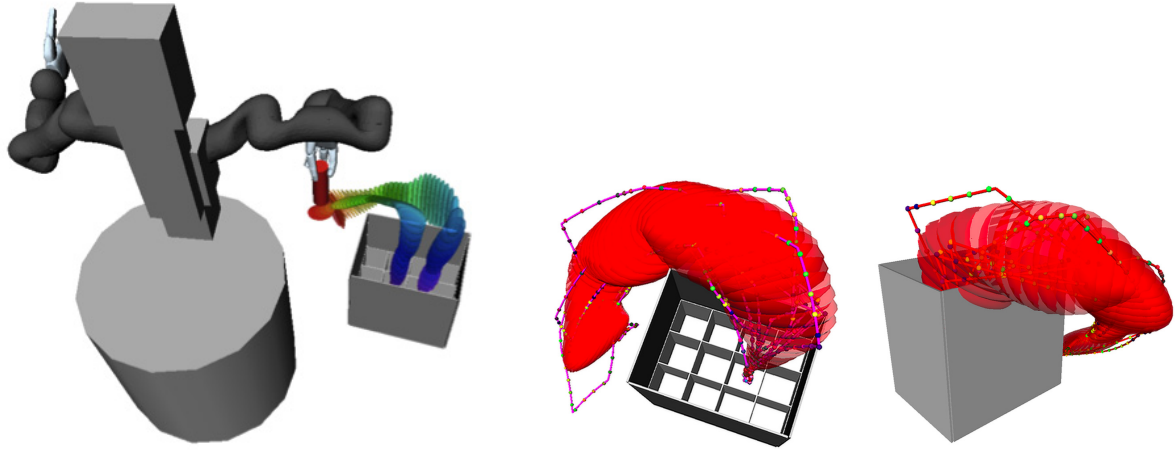
Figure 4.31.: Automatic generation of training examples during online motion planning. Collision-free goal configuration inside crate pockets are found automatically.

trajectories, which minimizes spatial distance. Based on DTW we define a distance measure, which weights the distances of configurations close to the goal higher than distances of configurations near the start.

Let $\pi = (\pi_i) \in \mathbb{R}^n$ and $\varphi = (\varphi_j) \in \mathbb{R}^n$ be paths. The result of DTW is a warped path $\Pi = ((\pi_i^k, \varphi_j^k))$, which holds the correspondences of time points of π and φ . The distance function d_{DTW} is defined by weighting the Euclidean distance with a sigmoidal function. We assign a minimum weight of α to each time point. β defines the increase in weight towards the goal:

$$d_{DTW}(\pi, \varphi) = \max_k \left(\alpha + (1 - \alpha) \frac{1}{1 + e^{\beta(1 - \frac{2k}{|\Pi| - 1})}} \right) \|\pi_i^k - \varphi_j^k\| \quad (4.14)$$

In the experiments, we applied $\alpha = 0.1$ and $\beta = 6$. The distance measure is used in Agglomerative Hierarchical Clustering with full-linkage to cluster the trajectories in the database with the distance threshold P_d . The result is a set of clusters, where each cluster contains only trajectories with DTW distance $< P_d$. In each cluster with more than P_c trajectories, we learn the parameters of the sequences of ellipsoid constraints with the algorithm in Section 3.1.3, i.e. we run the EM-algorithm to calculate a GMM, apply GMR and use the covariance ellipsoids with three standard deviations. Figure 4.32 shows learned search heuristics for different clusters as well as different distance thresholds P_d in the crate example.



(a) Learned search heuristics for two clusters - [55] (b) Search heuristic (front, back) learned with a higher distance threshold P_d . The sequence of ellipsoids intersects the crate, resulting potentially in a higher execution time.

Figure 4.32.: Learning of search heuristics.

4.6.5. Control Algorithm for Reverse Execution of Search Heuristics

We use a search heuristic in the planning process to generate the object-dependent part of the solution trajectory efficiently, which usually takes a large amount of time to plan. The object pose relative to the robot varies in real-world environments, e.g. the object is placed at a different position on the table or the robot pose is different due to obstacles in the environment. The consequence is that a planned joint-space trajectory cannot be used a second time. By learning sequence-of-ellipsoids-constraints, we represent a bundle of joint-space trajectories in Cartesian space based on coordinate systems. Since the coordinate systems vary with the object, the search heuristics adapt automatically.

In order to generate a joint-space trajectory based on a search heuristic Ψ_{SH} and a given start configuration, a controller is used. In general, we consider the controller a black box. Its inputs are the start configuration θ_0 , a constraint set Ψ and multiple temporal sequences of Gaussians $\mathcal{N}(\hat{\mu}_t^{(i)}, \hat{\Sigma}_t^{(i)})$, $0 \leq t \leq T_{max}$. The output is a trajectory $\pi : [0, 1] \rightarrow \Theta$ with $\pi(0) = \theta_0$ on which all constraints are obeyed: $\forall t \forall \tau \in \Psi : \tau(\theta(t))$. In order to use the search heuristics in the controller, the sequence-of-ellipsoids-region of each constraint j in Ψ_{SH} is transformed into a temporal sequence of Gaussians $\mathcal{N}(\hat{\mu}_t^{(j)}, \hat{\Sigma}_t^{(j)})$ using the algorithm in Section 3.1.3.

We apply the constraint-based controller in [21], which will be summarized here. The controller calculates for each constraint j the distance vector from the current value at time t_i to the mean: $\Delta x_{j+1}^{(i)}$. The distance vector is projected into joint space using the Jacobian $J(\theta_j)$ with the pseudo-inverse $J^\dagger(\theta_j)$:

$$\Delta \theta_{j+1}^{(i)} = J^\dagger(\theta_j) \Delta x_{j+1}^{(i)} \quad (4.15)$$

The covariance is projected in the same way

$$\Sigma_{j+1}^{(i)} = J^\dagger(\theta_j) \hat{\Sigma}_{j+1}^{(i)} J^\dagger(\theta_j)^T \quad (4.16)$$

The projected distance vectors are weighted, summed up and added to the current configuration to yield the goal configuration for the next time point [21]:

$$\theta_{j+1} = \theta_j + \left(\sum_{i=1}^n (\Sigma_{j+1}^{(i)})^{-1} \right)^{-1} \left(\sum_{i=1}^n (\Sigma_{j+1}^{(i)})^{-1} \Delta \theta_{j+1}^{(i)} \right) \quad (4.17)$$

The covariance in joint-space is calculated as

$$\Sigma_{j+1} = \left(\sum_{i=1}^n (\Sigma_{j+1}^{(i)})^{-1} \right)^{-1} \quad (4.18)$$

We extended the controller in the last step. If a constraint is violated on the linear interpolation from the old θ_j to the new configuration θ_{j+1} , we apply random search to find a new configuration, where all constraints are obeyed. The random distribution is defined by the projected covariance matrix in joint-space $\mathcal{N}(\theta_{j+1}, \Sigma_{j+1})$. A global timeout is respected in all steps to limit the maximum planning time.

The result of the controller is a path θ_j . The path represents the object-dependent part of the solution trajectory. In order to complete the solution trajectory, we apply constraint-based motion planning, which is explained in the next section.

4.6.6. Extension of Constraint-based Motion Planner

In Section 3.4, we defined a modified CBiRRT constraint-based motion planner to plan robot motions based on a linear planning model. The strength of motion planning is flexibility to changes in the environment, e.g. different object poses, at the cost of high planning times. In order to reduce planning time, we incorporate experience about previous planning attempts in the form of search heuristics. The downside of short execution times is inflexibility to large changes in the environment and joint-space limitations. We extend the modified CBiRRT in Section 3.4.3 to consider search heuristics during the generation of goal configurations. In the constraint-based motion planner, a set of goal configurations is generated prior to the planning process representing the goal of the manipulation task. In the algorithm in Section 3.4.2, a configuration, which obeys a subset of constraints in the planning model, is generated using inverse kinematics. The configuration is projected to the goal constraint manifold to generate a configuration, which obeys all goal constraints. We extend the algorithm in this step.

First, the search heuristic is added to the set of goal constraints. In the constraint-based motion planner, goal configurations are generated, which obey all goal constraints and lie in the last ellip-

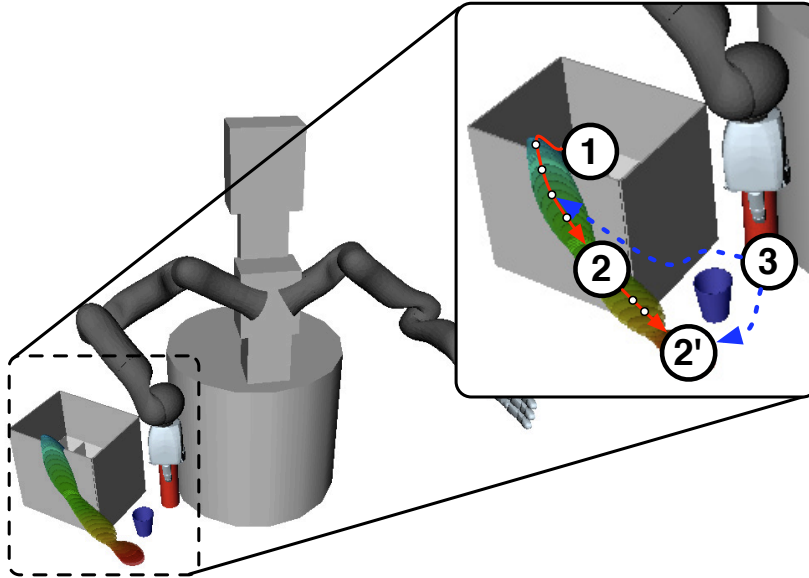


Figure 4.33.: Visualization of planning algorithm. In step (1), goal configurations are generated in the intersection of the goal constraint and final ellipsoid. The control algorithm is used to generate a path from (1) to the beginning of the GMM. Due to local obstacles and workspace limitations, different configurations (2) or (2') can be reached. The motion from (1) to (2) or (2') is discretized and the resulting points (white) serve as the set of goal configurations for a second planning process starting from (3), which allows to avoid obstacles globally, e.g. the cup. - [55]

soid of each search heuristic constraint. Due to the smaller search space, the time to generate goal configurations is reduced.

Second, we apply the controller to the generated configuration using the selected search heuristic $\mathcal{N}(\hat{\mu}_{T_{max}-t}^{(j)}, \hat{\Sigma}_{T_{max}-t}^{(j)})$, i.e. in reverse temporal order. The result is a discretized, partial path $(\theta_0, \dots, \theta_T)$ from a configuration, in which the goal constraints are obeyed, to the first time point of the search heuristic. We add each θ_i to the set of goal configurations and store a reference to the partial path $(\theta_0, \dots, \theta_i)$.

In the modified CBiRRT, a path from the start configuration to one of the goal configurations is planned obeying all constraints in the planning model. If a solution is found, the partial path, which is referenced by the goal configuration, is added to the solution in reverse temporal order. The result is a valid solution to the planning model, i.e. all constraints are obeyed and a configuration on the goal constraint manifold is reached. The approach is visualized in Figure 4.33.

4.6.7. Discussion

The planning algorithm combines two major approaches in Programming by Demonstration in a flexible way: efficient encoding and reproduction of robot trajectories and flexible, goal-directed planning based on a sophisticated task description.

In the worst-case, a configuration on the goal constraint manifold is generated but the controller is unable to generate a path based on the selected search heuristic, e.g. due to local obstacles or vicinity of the workspace boundary. The path contains only the generated configuration and the approach reduces to the modified CBiRRT in Section 3.4. In this case, the flexibility of the modified CBiRRT is maintained.

In the best case, a path to the beginning of the search heuristic, i.e. with length T_{max} , is generated in a short amount of time. A larger number of goal configurations is generated and at least one goal configuration lies in a large portion of the free space. The resulting planning problem is solved trivially using the modified CBiRRT leading to a short overall planning time.

4.7. Constraint Tightening to Solve the Correspondence Problem

Learning manipulation tasks based on explicit demonstrations by a human teacher is a natural choice. The human teacher is an expert in manipulation and teaching. He learned how to abstract from irrelevant information and how to choose illustrative examples, which contain enough information for other humans to learn the manipulation task. Based on these demonstrations, the robot is able to deduce the goals and constraints of the task, e.g. how the manipulated object should move. In state-of-the-art constraint-based motion planners, this information is insufficient to generate robot motions due to the high-dimensionality and size of the search space. Additional information about how the robot has to move its hands and fingers is required to remove irrelevant parts of the search space.

In order to learn this kind of information, the differences in morphology between human and robot have to be considered, which we called the *correspondence problem*, see Section 4.3. In our approach, constraints for the human hands and fingers were relaxed based on an analysis of the differences between the workspaces of the robot and human hand, see Section 4.3.2. The idea is to enlarge the search space, so that a robot motion with the same effect on the environment, e.g. producing the same object motion, similar to the human one is found considering the robot geometry, kinematics and dynamics. The enlarged search space leads to a higher planning time, which we reduce in a subsequent step by learning search heuristics in Section 4.6. A search heuristic i consists of a number of constraints $\tau_i^j = (e_1^j, \dots, e_m^j, f^j, R_i^j)$, one for each node constraint τ^j in the planning model. Each constraint τ_i^j refines the constraint τ^j . Arc constraints, which restrict the object referenced by e_1 , can be refined in the same way. In the constraint-based motion planner, see Section 4.6.6, we choose a search heuristic, i.e. a value for i , and generate a motion, which obeys all constraints in the planning model. Search is restricted to the constraint manifold defined by $(e_1^j, \dots, e_m^j, f^j, R^j \cap R_i^j)$, i.e. the intersection of the search heuristic constraint and corresponding node constraint. If planning fails, a different search heuristic will be chosen and the process

repeats. The search result lies on the constraint manifold defined by the disjunction of search heuristic constraints:

$$(e_1^j, \dots, e_m^j, f^j, R^j \cap \left(\bigcup_i R_i^j \right)). \quad (4.19)$$

We conclude that the set of search heuristic constraints can be interpreted as a specialization or tightening of the learned constraint τ^j , since $R^j \cap \left(\bigcup_i R_i^j \right) \subseteq R^j$. The specialization is object- and robot-dependent. It counteracts the negative effects of constraint relaxation to consider the correspondence problem.

The overall approach in this thesis to consider the correspondence problem can thus be summarized in the following way:

1. Learn constraints representing effects of the manipulation task on the environment as well as motions of the human hands and fingers based on the human demonstrations (Section 4.2).
2. Relax constraints, which restrict hands and fingers, based on the differences in morphology (Section 4.3).
3. Plan a robot motion, which reproduces the effects on the environment. The search space is restricted by learned hand and finger constraints, which focus search on motions similar to the human (Section 3.4).
4. Accumulate planning results to learn search heuristics constraints to speed up the planning process (Section 4.6).
5. Specialize the relaxed hand and finger constraints to reduce the search space and obtain a robot-specific representation of the planning model.

4.8. Summary and Conclusion

In the state of the art, we discussed subsymbolic representations of manipulation tasks originating from different research areas, e.g. constraint-based programming, motion planning, probabilistic and dynamic representations. Manual definition of even a simple manipulation task, e.g. like opening a bottle with the fingertips, is time-consuming, error-prone and requires expert knowledge about the representation. In real-world scenarios, the human is often not an expert in computer science but has, on the other side, tremendous experience in manipulation. In the PbD paradigm, the human experience in teaching and manipulation is exploited to learn manipulation tasks in an efficient and natural way based on a set of human demonstrations.

The basis of the PbD process is the observation of the human using different sensor systems. In general, the robot itself can be used but demonstration of fingertip motions is difficult. It requires either teleoperation of the robot or direct observation of the human hand. We have decided for the latter and use different sensor systems, e.g. data gloves and magnetic field trackers, to capture

the human motions (without occlusions) and forces applied through the fingertips. The effects in the environment, e.g. object motions, are either captured with a vision system or magnetic field trackers. A precise 3D model of the teacher’s hand was generated using a laser scanner. Based on the kinematics of the hand, information, e.g. the fingertip position, is calculated, which cannot be measured directly. Additionally, coordinate frames in contact points are generated dynamically using a simulation environment.

In order to learn a planning model based on a set of demonstrations three problems have been solved: generating the structure, i.e. nodes and arcs, defining a set of constraints for each node and arc and learning the constraint parameters. We defined two constraint sets, which are suitable to Pick-Transport-Place tasks, which don’t require to simulate force interaction, and dexterous manipulation tasks, which require to capture fingertip motion and simulate force interaction. In general, the approach scales to arbitrary sets of constraints.

The learned set of constraints is specialized to human morphology. Ignoring this fact is problematic. Anthropomorphic robot hands always differ in the size, workspace and placement of fingers compared to the hand of the human teacher. Even with high similarity between robot and human hands, the consequence is less flexibility since some variance in the human demonstrations will be lost, e.g. due to workspace limitations or self collisions of the robot hand. We considered this fact, which is called *correspondence problem*, by analyzing the workspace differences and relaxing constraints, which restrict finger and wrist motions.

The learned planning model is labelled *preliminary* since it contains a large number of constraints and, in general, does not generalize well to different environments. We introduced two complementary approaches to identify irrelevant constraints and increase flexibility of the planning model. In the demonstration-based generalization, the idea of Curriculum Learning is used and training data is sorted according to complexity, e.g. common and unusual solutions to a task, by the human teacher. Irrelevant constraints are removed efficiently but the teacher requires experience with the learning system to maintain relevant constraints. In the robot-test-based generalization, the human teacher defines a set of robot tests, e.g. object arrangements, to which the planning model has to be applied. We solve an optimization problem to identify a maximum subset of constraints, which admits a successful solution to all robot-tests. The basis are statistics about inconsistencies of constraints, which are generated in parallel using the constraint-based motion planner and computer-aided simulation.

The resulting planning model generalizes well to different environments but with potentially high planning time. In order to reduce planning time, we encode previous planning solutions in an efficient way as Gaussian Mixture Models to speed up later planning requests. These search heuristics are learned incrementally, i.e. new planning solutions are automatically generated if previously learned search heuristics fail. In a new environment, we use a control algorithm to reproduce the encoded planning solutions. In general, the reproduction is only partial due to collisions, constraint violations and workspace limitations. We complete these partial solutions using constraint-based motion planning, which combines two major approaches in PbD in a flexible

way: efficient encoding and reproduction of robot trajectories and flexible, goal-directed motion planning.

Finally, we use learned search heuristics to tighten constraints, which were relaxed to consider the correspondence problem. The result is a generalized planning model with constraints specialized to robot morphology.

Conclusion

Learning of manipulation tasks takes place in a high dimensional feature space, e.g. we consider forces (5D), joints (20D) and pose (6D) of each human hand, the pose of each object (6D) as well as a dynamic set of contacts. In the state-of-the-art in Chapter 2 we discussed different techniques to reduce the number of features with differing disadvantages. Projection to a latent space, e.g. PCA, mixes (potentially) all features, which makes it difficult to identify irrelevant ones. In Task Space Pools, a fixed set of criteria is introduced to rate different features, but the set of irrelevant constraints, which can be identified this way, is limited.

In this thesis, we regard the generalization process as an optimization problem. In principle, the objective function should express that the task is reproduced successfully. Since the planning model is the only representation of the task and is altered in the generalization process, this objective function can not be defined explicitly. A simple solution would be to use supervision, i.e. the teacher rates each solution, but due to the complexity of the optimization problem, this would be tiresome. We defined a different objective, i.e. maximize the number and diversity of constraints in the planning model and solve a robot-test. This objective function can be evaluated automatically, therefore supervision is delayed to rate the optimization result. The advantage is that planning and simulation can be used extensively to identify relevant constraints, which allows to consider complex interactions between constraints and reduce user-interaction.

The time to generalize a planning model scales directly with the planning time for each robot-test. High planning time is one of the main limitations of constraint-based motion planning. In the state-of-the-art, see Chapter 2, we discussed different search heuristics, e.g. searching in approximations of the free space, to speed up the planning process. In real-world scenarios, a single search heuristic might fail and will not lead to a significant speed-up. Learning is mandatory. In contrast to the referenced work, new search heuristics are learned automatically whenever previously learned search heuristics fail. Nevertheless, the number of learned search heuristics is small due to the tight integration with the planning algorithm. The approach scales to complex tasks, in which only a small number of different solutions can be found.

Search heuristics are abstract representations of planning solutions, which incorporate the robot's morphology. We exploit this fact to specialize learned constraints for fingertips and wrists, which were initially relaxed to consider the differences between the human and the robot. In contrast to work in the state-of-the-art, e.g. using a correspondence matrix to map joint values be-

tween human and robot [1], the mapping is task-dependent and considers implicitly the kinematic and geometric differences.

The result of the developed PbD process is a constraint-based representation of the manipulation task, which was generalized (semi-)automatically to increase flexibility and specialized automatically to decrease planning time and solve the correspondence problem.

5. Evaluation

In the previous two chapters, we presented concepts and algorithms to generate models to plan the execution of manipulation tasks in a flexible way and to acquire these models using Programming by Demonstration. In this chapter, the feasibility of the developed concepts, individual components as well as the fully integrated PbD-system are evaluated experimentally.

The chapter starts with an overview of the experimental setup including a short description of the real robot systems and the object database, see Section 5.1.

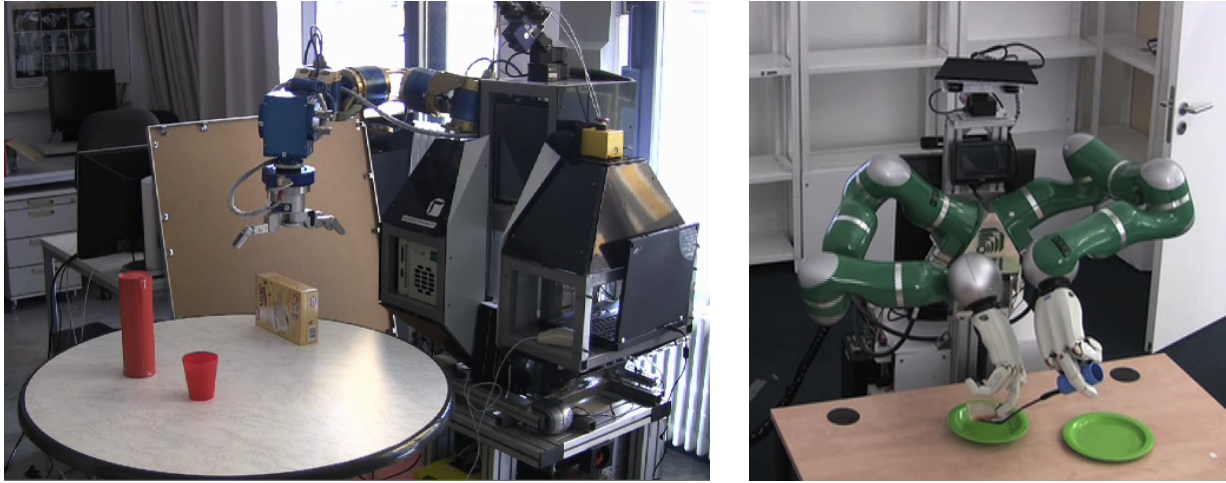
In general, we combine evaluation on the real robot platforms with extensive evaluation in computer-aided simulation. The latter allows to analyze the developed algorithms using statistical methods based on a large number of simulated execution trials. An example is the analysis of the flexibility of planning models, which requires the execution in different environments with varying object poses and different obstacles. In order to generate a large number of differing environments, we rely on statistical methods and define probability distributions to sample object and obstacle poses.

In Section 5.2, we use this approach extensively to compare flexibility of different execution mechanisms with the planning-based approach in this thesis. Additionally, the flexibility and achievable grasp quality of learned planning models for grasping strategies is compared with a public grasp database.

Section 5.3 focuses on the evaluation of the demonstration-based and robot-test-based generalization algorithm. We analyze the demonstration-based generalization on multiple manipulation tasks focusing on qualitative aspects and the evaluation of planning models in different stages of the generalization process. The robot-test-based generalization is analyzed in a quantitative way by measuring the number of identified, irrelevant constraints as well as the resulting generalization error using multiple manipulation tasks on different robot systems.

Incremental learning of search heuristics and efficient execution of planning models is evaluated in Section 5.4 using multiple manipulation tasks. We analyze flexibility in different experiments using fixed and random environments. Planning time as well as the dependency on relevant parameters is analyzed using statistical methods. Incremental learning is evaluated explicitly by running the system on sequences of environments with different object and obstacle poses. Additionally, we provide a comparison with a control-based approach without motion planning.

Section 5.5 focuses on the evaluation of the fully integrated PbD-system on dexterous manipulation tasks, which require force interaction between the robot fingertips and the manipulated object. The results of the individual components are analyzed and the resulting planning models are exe-



(a) Albert II ready to grasp a box, - [51]

(b) Adero using a spatula

Figure 5.1.: Robots used in key experiments.

cuted on the bimanual, anthropomorphic robot Adero. Additional simulation trials are generated to evaluate planning with dynamics simulation using the original and specialized planning model.

In Section 5.6, we evaluate the approach to consider the correspondence problem in learning of manipulation tasks by constraint relaxation and tightening. Multiple manipulation tasks and random environments are used to measure the reduction in volume of the learned constraint regions, which is obtained by projecting learned search heuristics on the constraint regions.

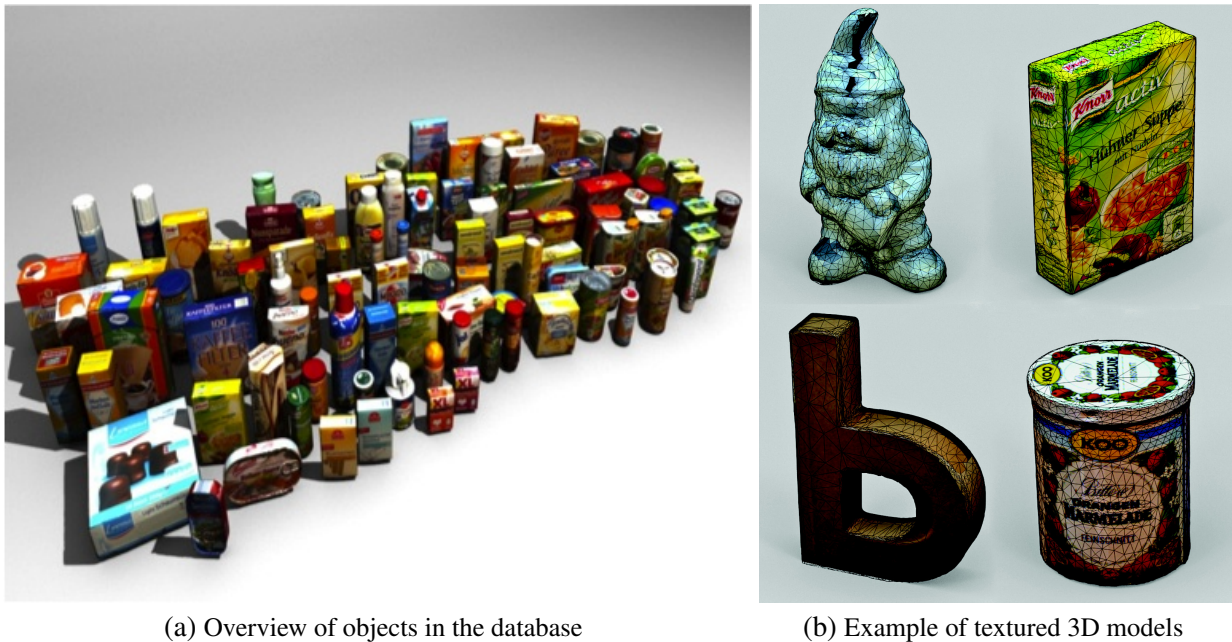
Finally, we show scalability of the PbD-system to different robot systems, e.g. Adero, Albert II, Armar IIIb, Justin and PR2, in Section 5.7.

5.1. Experiment Setup Overview

Experiments on key components, e.g. demonstration-based and robot-test-based generalization and specialization, were performed on two different robot systems, which will be described in Section 5.1.1. In order to learn and plan manipulation of everyday objects, precise 3D models of each object are required. We employ a database of 3D models, which will be described in Section 5.1.2. The components of the developed PbD-system as described in Chapter 3 and 4 as well as the overall architecture, see Figure 1.4 on page 8, are realized in software and fully integrated in a coherent way. The software implementation is described in Appendix C.

5.1.1. Robots

Figure 5.1 shows the robots Albert II and Adero, on which the key experiments were performed. Albert II has a 3-DOF mobile platform and a 6-DOF Amtec-arm with a 6D DLR-force-torque-sensor in the wrist and a 4-DOF Barrett Hand. Adero consists of two KUKA Lightweight Arms with 7 DOF with active compliance and two 13-DOF Schunk Anthropomorphic Hands. The second version of Adero supports additionally a mobile platform with 3 DOF.



(a) Overview of objects in the database

(b) Example of textured 3D models

Figure 5.2.: 3D-models in KIT Object Models database, [59].

The mapping and planning of manipulation strategies was also implemented on the robots Armar IIIb, PR2 and Justin. Experiments on Armar IIIb and PR2 were only performed in simulation.

5.1.2. Object Database

Kasper et al. [59] developed an interactive object modeling center to generate high resolution, textured 3D models of rigid objects, see Figure 5.2, in an efficient and intuitive way. We used the system to generate a 3D model of the human hand, see Section 4.1.1. The object models are stored in the (public) KIT Object Models database [58]. In the experiments, we use different objects from the database, e.g, OrangeMarmelade, YellowSaltCube, Sauerkraut, DanishHam, BathDetergent, CokePlasticSmall, CokePlasticLarge, VitalisCereals, SmacksCereals, Sprudelflasche. Additionally we use 3D models of different cups, a saucer, a measuring cup, a chips can, a chair, a chessboard, chess pieces, a crate, different plates, different spoons and different spatulas.

5.2. Evaluation of Planning Algorithm

In this section, we compare the developed planning algorithm with standard techniques on simple tasks, e.g. grasping of different objects and pouring in. First, we compare grasp quality and success rate of a set of learned grasping strategies with a grasp database on sets of fixed and random object poses, see Section 5.2.1. In Section 5.2.2, we compare the flexibility of the planning algorithm to different object and obstacle arrangements with different execution mechanisms.

Task	Min	Max	Average	Success
CokePlasticsLarge	0	0.48	0.16	0.95
CokePlasticsSmall	0	0.49	0.12	0.95
CokePlasticsSmall (top)	0	0.16	0.01	0.99
DanishHam	0	0.24	0.11	0.92
OrangeMarmelade	0	0.26	0.03	0.96
Pringles	0	0.44	0.16	0.95
Mildessa	0	0.22	0.05	0.96
Smacks	0	0.17	0.03	0.94
VitalisCereals	0	0.26	0.08	0.93
YellowSaltBox	0	0.23	0.03	0.96

Table 5.1.: Result: learned grasping strategies, fixed object poses

5.2.1. Grasp Strategies

We use the KIT Object Models database [59] as a reference to evaluate learning of grasping strategies from the human teacher. In the database each object is annotated by a set of grasps for different robot hands, e.g. Schunk Anthropomorphic Hand or Schunk Dexterous Hand (SDH). A grasp in the database is defined as a sequence of actions:

1. The hand is placed approximately 10cm away from the object with a certain preshape.
2. The hand is moved in direction of the object.
3. The hand is closed until contact.
4. Force is applied in the fingertips, i.e. the fingers are closed a bit further.
5. The object is lifted.

The following objects were used: CokePlasticsLarge, CokePlasticsSmall, Pringles, DanishHam, OrangeMarmelade, VitalisCereals, Smacks and YellowSaltBox. For each object, we demonstrated a grasping strategy in the sensory environment and learned a planning model, see Figure 5.3. The first constraint set was used. Each planning model consisted of two nodes and one arc. In the first experiment, we placed each object at (the same) fixed position and orientation on the table in front of the robot at position $(900, -400, 720)^T$ with orientation $(0, 0, 0)^T$ and planned each learned planning model 100 times, see Figure 5.4. In Table 5.1, the minimum, maximum, average grasp quality and success rate of the planning process is shown. A query is successful if a robot trajectory is generated, e.g. the approach motion is in the workspace. In this case, the grasp quality can still be 0.

Additionally, we executed each grasp using the sequence of actions in the database and calculated the grasp quality, see Table 5.2. The execution failed if one of the motions was not in the workspace of the robot arm. In both approaches, the identical grasp quality measure was used.

For all objects, the learned grasping strategies had a larger maximum grasp quality. The reason is that in the database, a small set of distinct hand poses is used but if we plan a grasping strategy, a large number of grasps is generated automatically in the search space defined by the learned

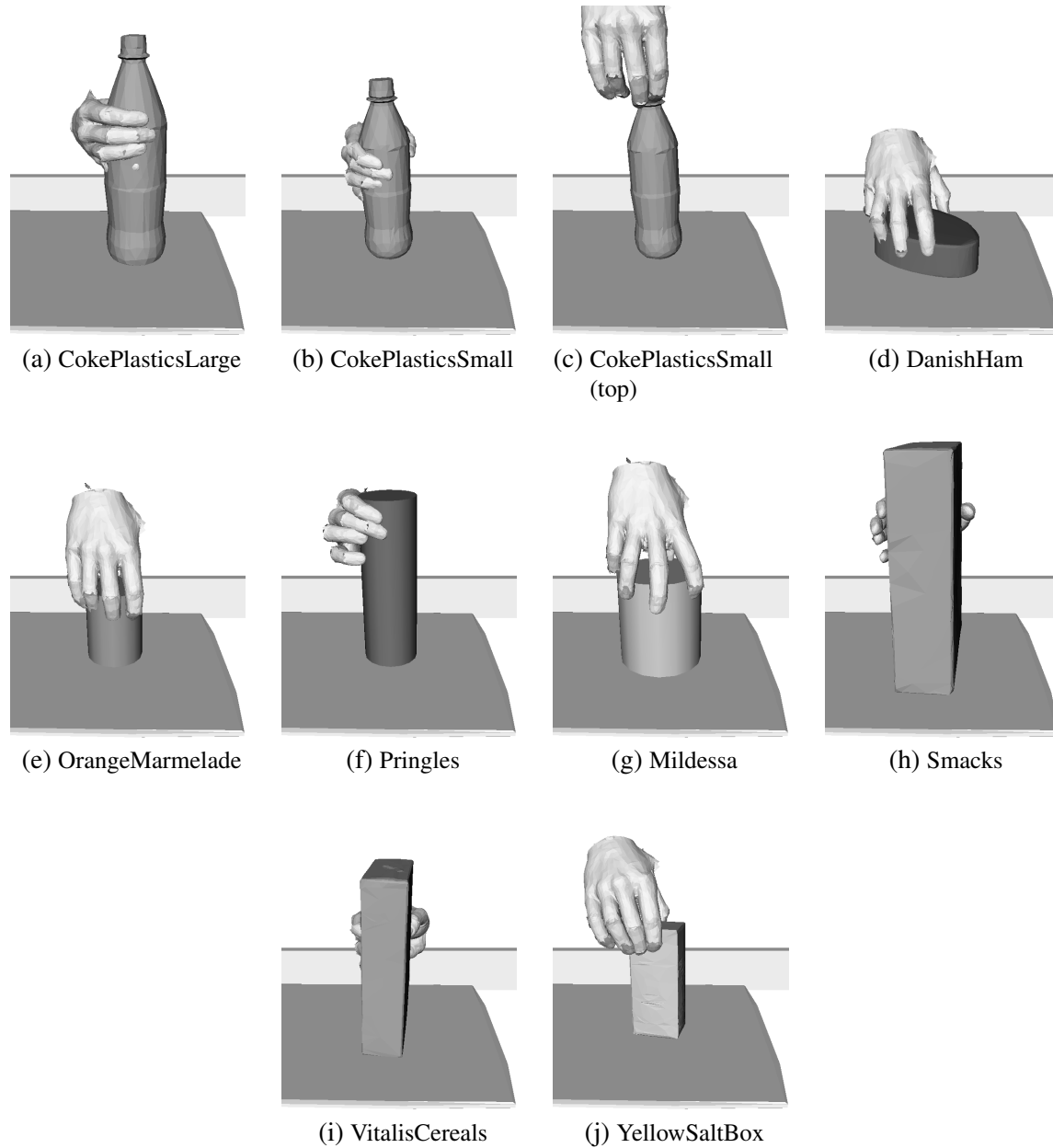


Figure 5.3.: Examples of human grasp demonstrations for the selected objects.

constraints. Even if the search space is small, large differences in grasp quality can be obtained due to local differences on the object surface. Nevertheless, the performance of the grasp database is good. On average, 0.29 of all grasps could be applied and only for the object Mildessa a grasp with a grasp quality close to zero was found. The average success rate of the learned grasping strategies is 0.95. We would have to draw 9 random grasps from the database to achieve a similar success rate, i.e. $(1 - 0.29)^9 < 1.0 - 0.95$.

In the second experiment, a more realistic setup was used. We randomized the object position with $(\pm 300, \pm 300, 0)^T$ and the orientation with $(0, 0, \pm 45)^T$. The result of the learned grasping strategies is shown in Table 5.3. The grasps in the database were executed in random order and the first successful grasp was taken. In contrast to the first experiment, the success rate for the database

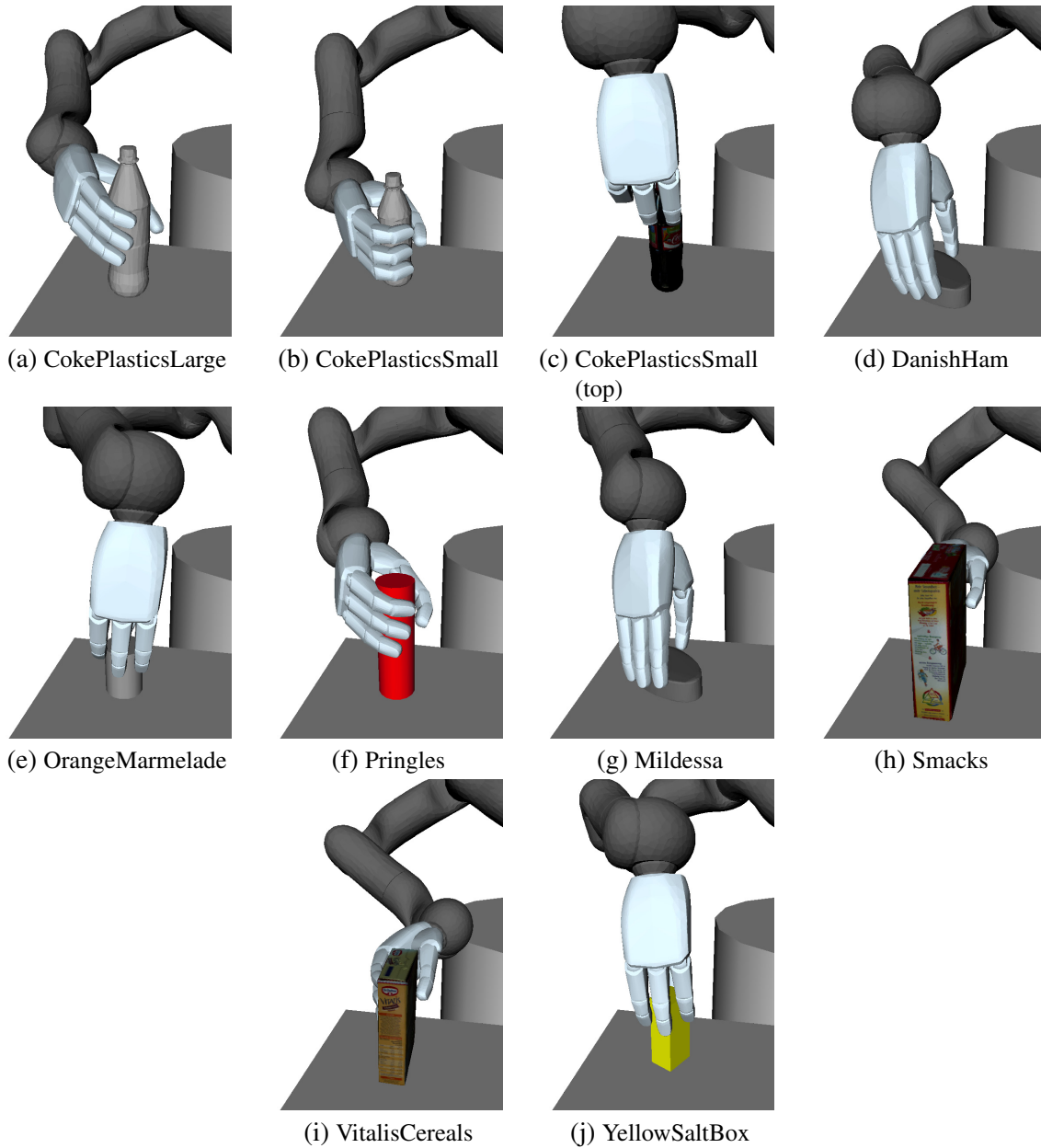


Figure 5.4.: Examples of planned grasp strategies for the selected objects.

reflects if a successful grasp is found. The result is summarized in Table 5.4. Both approaches were executed 100 times.

In this experiment, both approaches produced grasps with similar high (averaged) grasp qualities (PbD: 0.33, database: 0.2) and success rates (PbD: 0.95, database: 0.78). We can expect that the slight advantage of PbD will diminish with a larger grasp database. The main advantage is that task-dependent grasps are learned. In Table 5.3, two different grasping strategies for CokePlasticsSmall were learned, one from above and one from the side. The average grasp qualities show that it is more difficult to generate grasps from above (average 0.01) than from the side (average 0.13) but still a high quality grasp (maximum 0.17) can be generated.

Task	Min	Max	Average	Success
CokePlasticsLarge	0	0.37	0.16	0.24
CokePlasticsSmall	0	0.38	0.09	0.26
DanishHam	0.23	0.23	0.23	0.25
OrangeMarmelade	0.14	0.25	0.20	0.50
Pringles	0	0.35	0.17	0.26
Mildessa	0	0.09	0.03	0.30
Smacks	0	0.24	0.08	0.29
VitalisCereals	0	0.44	0.07	0.26
YellowSaltBox	0	0.22	0.06	0.22

Table 5.2.: Result: grasp database, fixed object poses

Task	Min	Max	Average	Success
CokePlasticsLarge	0	0.47	0.17	0.94
CokePlasticsSmall	0	0.49	0.13	0.94
CokePlasticsSmall (top)	0	0.17	0.01	0.99
DanishHam	0	0.23	0.10	0.90
OrangeMarmelade	0	0.24	0.03	0.96
Pringles	0	0.47	0.14	0.94
Mildessa	0	0.27	0.05	0.96
Smacks	0	0.25	0.03	0.94
VitalisCereals	0	0.34	0.09	0.93
YellowSaltBox	0	0.30	0.02	0.96
Becher	0	0.38	0.06	0.95

Table 5.3.: Result: learned grasping strategies, random object poses

In general, the grasp quality measure is sufficient to find grasps that can be executed successfully on the real robot [54], see Figure 5.5.

The results show that grasping strategies can be learned efficiently using the developed PbD process. The generated grasp qualities are similar to grasp qualities in the precomputed database. A clear advantage is that task-dependent grasps can be learned.

5.2.2. Planning without Physics Simulation

In this section, we evaluate flexibility of 5 different planning and execution mechanisms. The task is to pour in consisting of multiple steps: grasping the cup and bottle, moving the cup in front of the robot and pouring in, see Figure 5.6a. The (sub-)planning model to pour in contains 4 nodes

Task	Min	Max	Average	Success
CokePlasticsLarge	0	0.25	0.12	1.00
CokePlasticsSmall	0	0.28	0.09	1.00
DanishHam	0.03	0.24	0.18	0.45
OrangeMarmelade	0.06	0.27	0.16	0.64
Pringles	0	0.34	0.23	1.00
Mildessa	0	0	0	0.18
Smacks	0.02	0.17	0.10	0.91
VitalisCereals	0	0.22	0.09	1.00
YellowSaltBox	0	0.09	0.03	0.82

Table 5.4.: Result: grasp database, random object poses

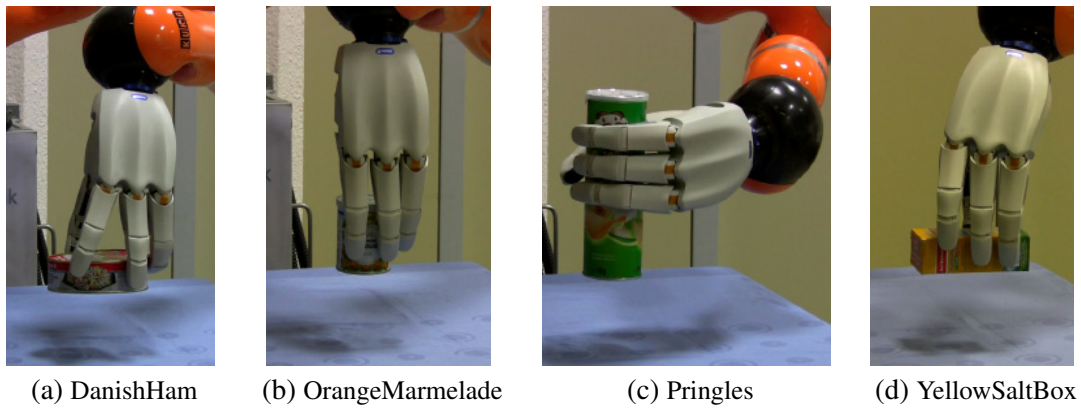


Figure 5.5.: Execution of planned grasps on Adero. - [54] .

and 3 arcs. The grasps are not predefined but planned flexibly leading to a high variance in grasp postures, i.e. how the hand is placed relative to the bottle. The cup is placed in front of the robot with a high variance in position and orientation. The goal is to evaluate, how the mechanisms adapt to collisions, self-collisions, variance in reference poses, e.g. cup in front of the robot, and variance in grasp postures, e.g. hand relative to bottle.

In general, we executed the mechanisms 5 times on different settings. In each setting, the mechanisms were executed in multiple trials. The planning and execution mechanisms are

1. Planning: a trajectory is planned using the planning model.
2. Planning & search heuristics: prior to the trials, we learn a search heuristic. In each trial, a trajectory is planned using the planning model and learned search heuristic.
3. Planning & full path: prior to the trials, we generate a trajectory for the left and right arm using the planning model. In each trial, we plan a path to the first configuration of the generated trajectory and execute it.
4. Search heuristics only: prior to the trials, we learn a search heuristic. In each trial, we plan a trajectory using the planning model until the cup has been moved in front of the robot. We execute the learned search heuristic directly using the controller to pour in.
5. Planning & partial path: prior to the trials, a trajectory is planned. We store the Cartesian trajectory of the bottle relative to the cup during pouring in. In each trial, we plan a trajectory using the planning model until the cup has been moved in front of the robot. In order to pour in, inverse kinematics are used to generate a trajectory based on the Cartesian trajectory of the bottle relative to the cup.

We evaluated three different settings: random cup and bottle position, random position of one and random position of two obstacles, see Figure 5.6. The basis is a 3x3 set of positions in front of the robot. In the first setting, we placed the cup and bottle one after another on all combinations of

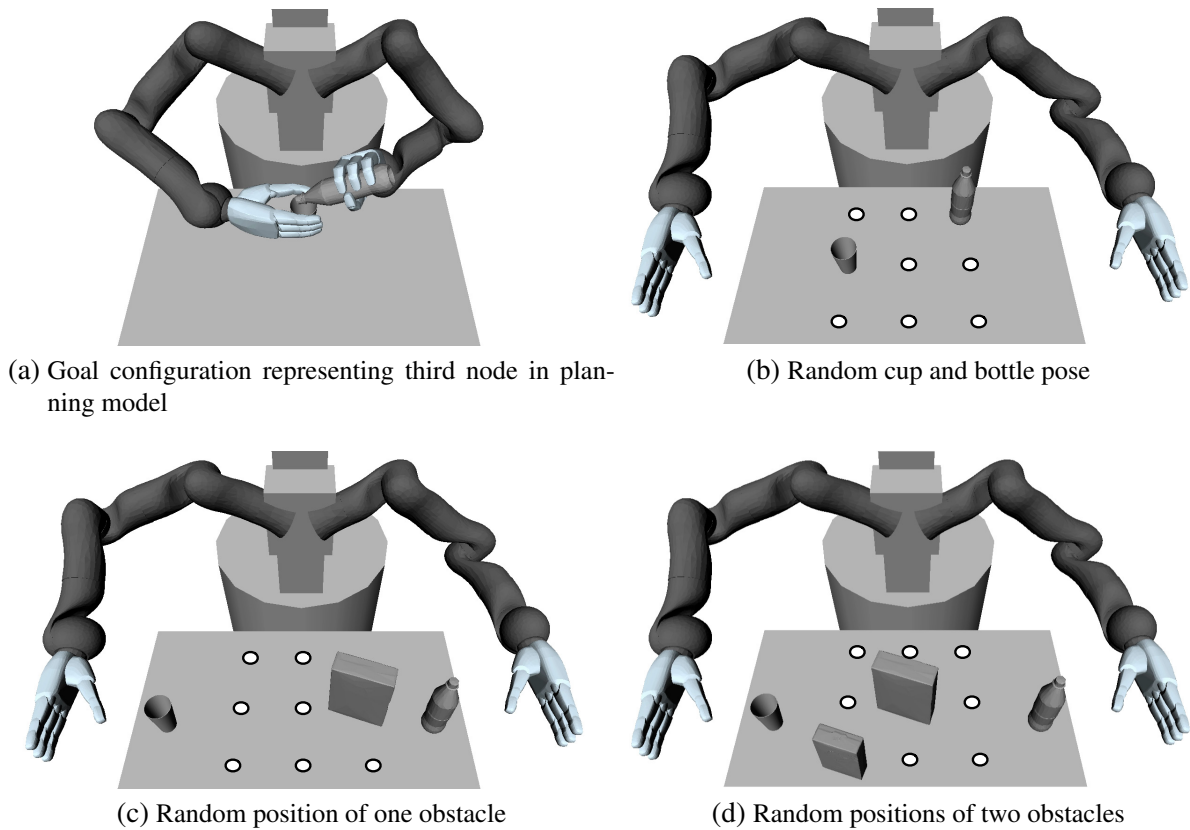


Figure 5.6.: Planning: pour-in with variable cup, bottle and obstacle poses.

positions (except the same), see Figure 5.6b. In the second setting, see Figure 5.6c, a large obstacle was placed on all positions. In the last setting, see Figure 5.6d, we added a smaller obstacle and placed both obstacles on all combinations (except the same).

Each mechanism was executed 5 times on each setting. For instance, in the first setting 72 combinations of cup and bottle are possible and each mechanism is executed 5 times each with 72 trials. Table 5.5 shows a summary of the success rates for each mechanism. Planning and planning with search heuristics show higher success rates compared to the last three mechanisms. Figure 5.7 visualizes the results in the first setting. In each (small) map the red patch shows, where the bottle is placed. The other patches visualize the success rate, when the cup is placed on the corresponding position. A dark green indicates a success rate of 100%, white a success rate of 0%. Intermediate values are interpolated. Since the cup is grasped with the right hand and the bottle with the left, patches on the right side of the bottle are often blocked. The result are tight spaces with a high collision potential, in which the first three mechanisms are more successful. For all mechanisms, results are best, when the bottle is on the right side. Direct execution of search heuristics and partial paths fail more often: (self-)collisions cannot be avoided sufficiently and the motion cannot be adapted to different cup positions in front of the robot as well as different hand postures since the workspace is left. The higher success rate of planning with the

Method	Setting 1	Setting 2	Setting 3
Planning	77.1	97.2	95.5
Planning & search heuristics	75.5	96.3	99.1
Planning & full path	65.6	50.0	21.8
Search heuristics only	20.3	20.6	9.7
Planning & partial path	16.1	4.4	13.0

Table 5.5.: Summary of results: success rates in % in different settings.

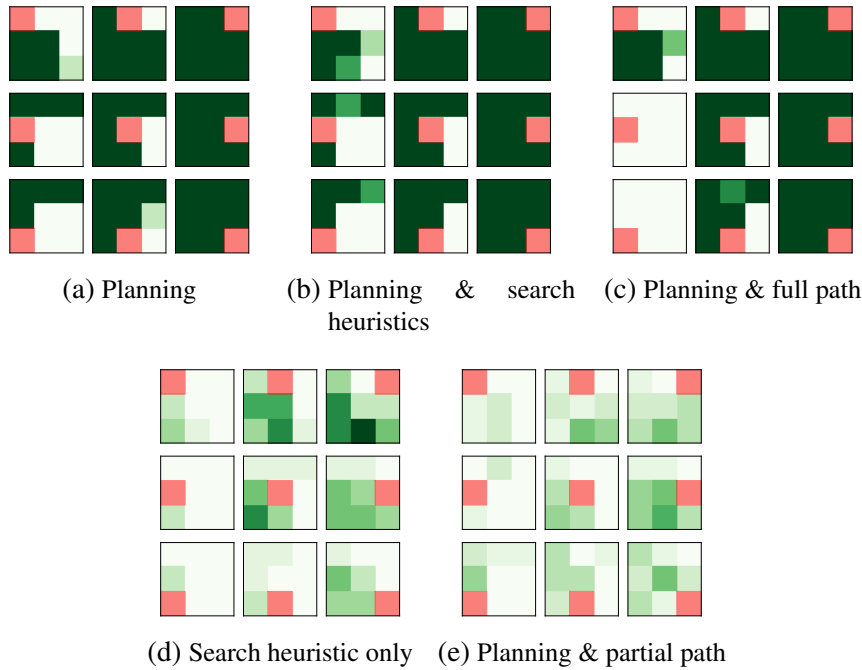


Figure 5.7.: Visualization of success rates for setting 1: random cup and bottle position. The bottle is placed on the red position. The cup on the remaining positions on the table. Dark green is 100%, white is 0%.

full path compared to planning with the partial path shows that cup and hand postures have to be reproduced to increase the success rate.

In setting 2, the cup and bottle pose were placed left and right of the 3x3 set of positions and a single obstacle was placed on the table on the nine different positions. The first and second mechanism can easily adapt to the new situation. Figure 5.8c shows the result for planning with the full path. The white patches indicate that the first configuration of the generated trajectory is blocked by the obstacle, leading to a success rate of 0%. In the last two mechanisms, success rate is small since the controller and inverse kinematics often failed due to workspace violations and collisions.

Figure 5.9 shows the last setting, in which two obstacles are placed on the table. Planning with search heuristics is superior to planning since the time threshold is reached less often. The success rates of method 1 and 2 remain stable. Success rate of method 3 drops to 21.8% since the start configuration is blocked more often. Method 4 and 5 show similar results.

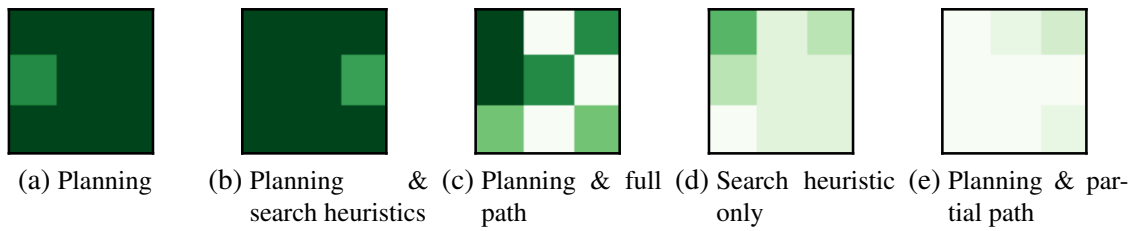


Figure 5.8.: Visualization of success rates for setting 2: random pose of one obstacle. Dark green is 100%, white is 0%.

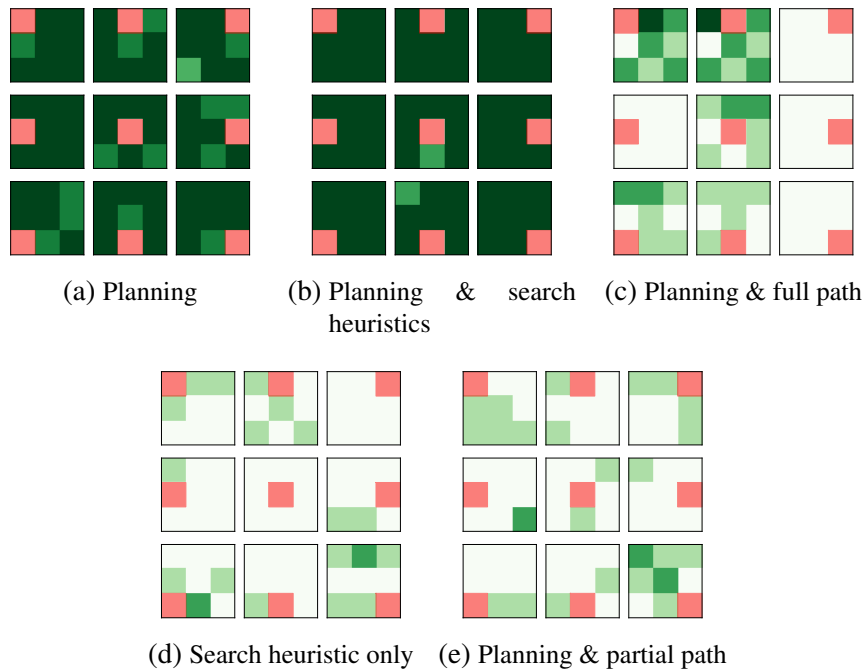


Figure 5.9.: Visualization of success rates for setting 3: random pose of two obstacles. The first obstacle is placed on the red position. The second obstacle on the remaining positions on the table. Dark green is 100%, white is 0%.

Planning based approaches show higher success rates in all settings. The success rates of method 4 and 5 show that adapting a single (Cartesian) trajectory or search heuristic to different grasp postures or goal poses is problematic. The results of method 3 in setting 1 indicate that generating trajectories or search heuristics for the complete task, i.e. grasps, cup motion and pour-in, is advantageous since the intermediate grasp postures and cup motions are compatible. The problem of method 3 is less flexibility, which is underpinned by the low success rates in setting 3. Method 2 represents a flexible combination of planning and search heuristics. The generated maps of method 3, 4 and 5 have more patches, i.e. obstacle or object arrangements, which contain no solutions, i.e. 41.2%, 43.0% and 55.5% patches with less than 1% solutions, see Table 5.6. The planning-based methods 1 and 2 are more flexible with only 6.93% respectively 7.40% patches with less than 1%.

The results indicate that planning with the original planning model and planning with search heuristics lead to a high flexibility with respect to different object and obstacle arrangements.

Method	Setting 1	Setting 2	Setting 3	Average
Planning	20.8	0	0	6.93
Planning & search heuristics	22.2	0	0	7.40
Planning & full path	33.3	33.3	56.9	41.2
Search heuristics only	44.4	11.1	73.6	43.0
Planning & partial path	34.7	66.7	65.2	55.5

Table 5.6.: Summary of execution results: percentage of environments with success rate $< 1\%$.

5.3. Evaluation of Planning Model Generalization

In this section, we evaluate demonstration-based and robot-test-based generalization of the preliminary planning model.

5.3.1. Demonstration-based Generalization

The goal of the demonstration-based generalization is to remove irrelevant constraints from a learned planning model. In this section, we analyze the approach in a qualitative way in two different tasks: chess knight move and bottle closing. The experiments were presented in [52]. We discuss the results in more detail in the next subsections.

Chess knight move

The goal is to learn the chess knight move in a single direction, e.g. two squares straight and one to the right, and to be able to execute it in a realistic chess match, i.e. with non-empty chess board. Calinon et al. [23] evaluated the same task and used PCA to reduce the number of features. The execution with a controller was limited to empty chessboards.

The chessboard and chess knight were each modeled with a single coordinate frame. The human teacher demonstrated the chess knight move 5 times from each corner and the center of the chessboard, see Figure 5.10. The teacher introduced only minor variations in the L-shaped motion. Based on these demonstrations, the initial planning model was learned. It contained 40 node and 72 arc constraints. The position constraints are visualized in Figure 5.11. The L-shaped motion is approximated by a large number of overlapping cylindrical constraints. The large cubical constraint restricts the position of the chess piece to stay about 3cm above the chessboard.

The planning model was planned on Albert II with a non-empty chessboard. The human teacher observed that the planning process failed since the chess knight could not be rotated, which is required to place it at the goal position. The teacher demonstrated the first additional demonstration: the chess knight stands initially with a different orientation on the chess board, the chess knight is moved and placed with a different orientation on the chessboard, see Figure 5.10. The result is that all constraints, which restrict the orientation of the chess knight were removed and only direction constraints remained. Due to the different orientation of the chess knight in the beginning, arc constraints which enforce an L-shaped motion were removed. The resulting planning model was executed again, see Figure 5.12. The task was executed successfully, i.e. a collision-free path

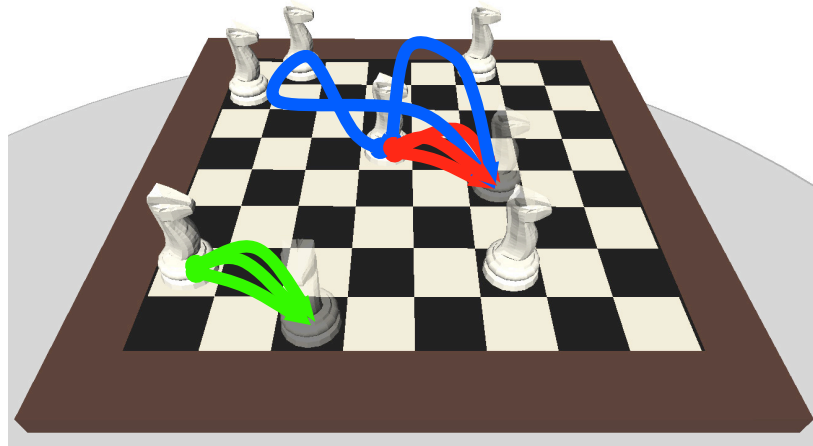
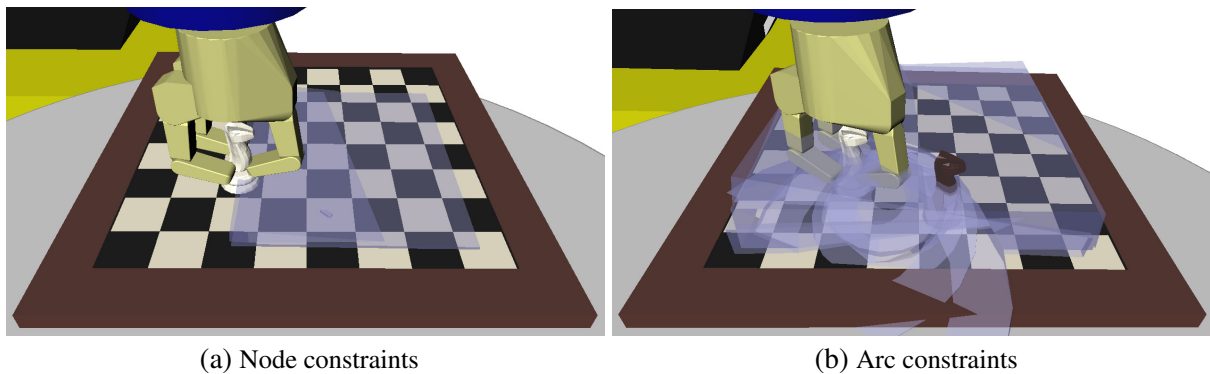


Figure 5.10.: Human demonstrations of chess knight move. In order to learn the initial planning model, the human teacher demonstrated the chess knight move 5 times in the corners and the center of the chess board. Only small variance in the actual, L-shaped motion was demonstrated (red). In the first additional demonstration, the chess knight was rotated during the motion (green). The human teacher lifted the chess knight above the chess board in the last additional demonstration (blue). - [52] .



(a) Node constraints

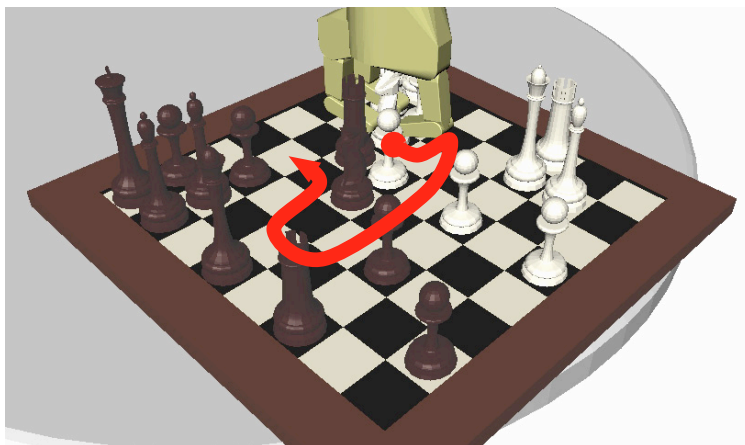
(b) Arc constraints

Figure 5.11.: Constraints in the initial planning model. - [52]

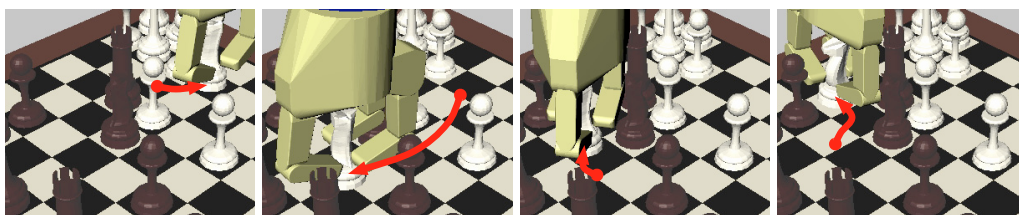
through all chess pieces was found by the motion planner. The execution is complicated since the chess knight is moved only a few centimeters above the chess board through the other chess figures to reach the goal.

The human teacher made the second additional demonstration to increase flexibility: lift the chess knight 20cm high above the chessboard, which removed the large cubical constraint.

The resulting planning model contained only a single direction node constraint, which restricts the chess knight to stay upright, and a single position node constraint, which restricts the chess knight relative to the orientation of the chess board and the position of the chess knight. The position node constraint ensures that the chess knight is placed two squares ahead and one to the right. Figure 5.13 shows the final planning results. The robot lifts the chess knight above the chessboard, rotates it slightly and places it without collision on the goal position.

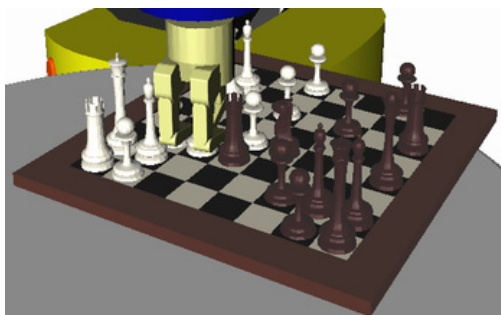


(a) Visualization of collision-free path through all chess pieces



(b) Visualization of different path segments showing how the chess piece is rotated

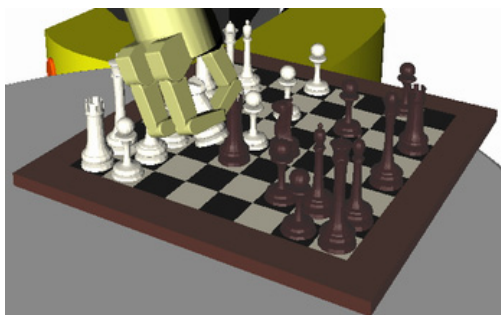
Figure 5.12.: Execution on non-empty chessboard after first additional demonstration. - [52]



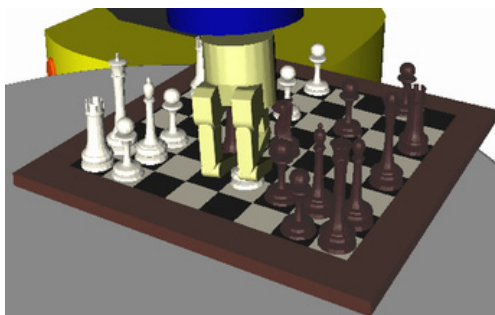
(a) Grasping



(b) Lifting

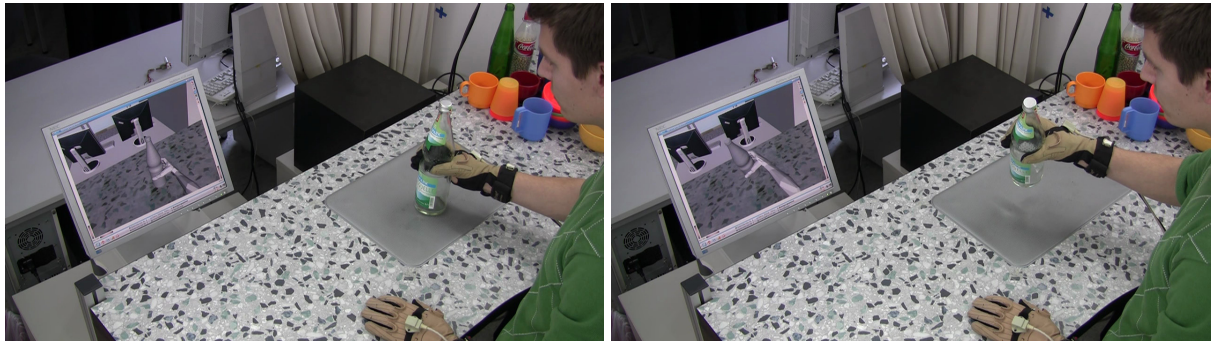


(c) Moving

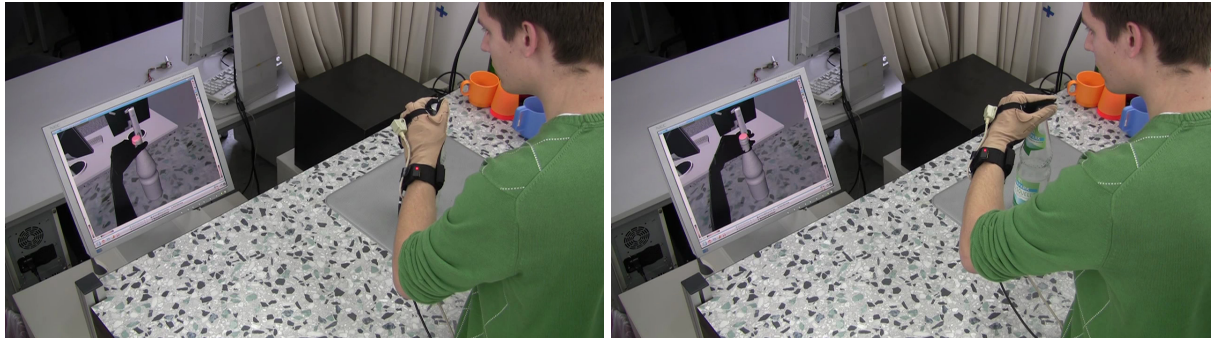


(d) Placing

Figure 5.13.: Execution on non-empty chessboard using all additional demonstrations.



(a) Example of human demonstrations to lift the bottle



(b) Example of human demonstrations to rotate the bottle cap

Figure 5.14.: Human demonstrations of the bottle closing task. The task was separated into two steps: lifting (a) and rotating (b) to limit influence of sensor noise. - [52]

Bottle closing

In this task, the goal is to grasp the bottle with the right hand, lift it, grasp the bottle cap with the left hand and rotate it clockwise to close the bottle. The lifting and rotating parts were demonstrated separately to reduce noise in the observation process. The hand and finger positions have to be as accurate as possible to be able to reproduce the rotation axis of the cap. The human demonstrated each part 8 times, see Figure 5.14. For both objects a cylindrical grasp preshape was detected.

The initial planning model could be executed with the same bottle on Adero in computer-aided simulation, see Figure 5.15.

The human teacher demonstrated different lifting motions to remove constraints, which restrict the motion of the bottle. Orientation and direction constraints remained, i.e. the bottle has to be kept upright. Execution with a different, smaller bottle failed. In this case, the visualization of goal configuration candidates generated in the planning process showed that the robot tried to grasp the bottle either near or above the top. In both cases, no configuration could be found, in which all goal constraints were obeyed indicating that wrong reference frames were contained in the constraint set of the initial planning model.

The human teacher demonstrated the task with a third, larger bottle. Based on these additional demonstrations, constraints were removed, which restrict the wrist pose relative to the bottle open-

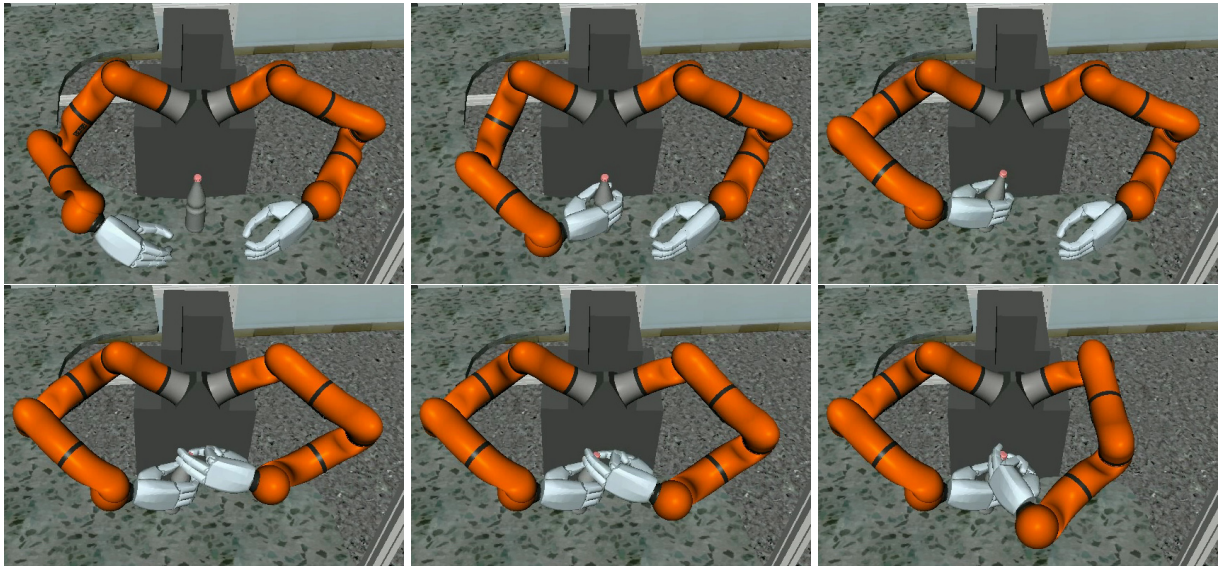


Figure 5.15.: Planning result of the bottle closing task with the bottle, which was used to learn the initial planning model. - [52]

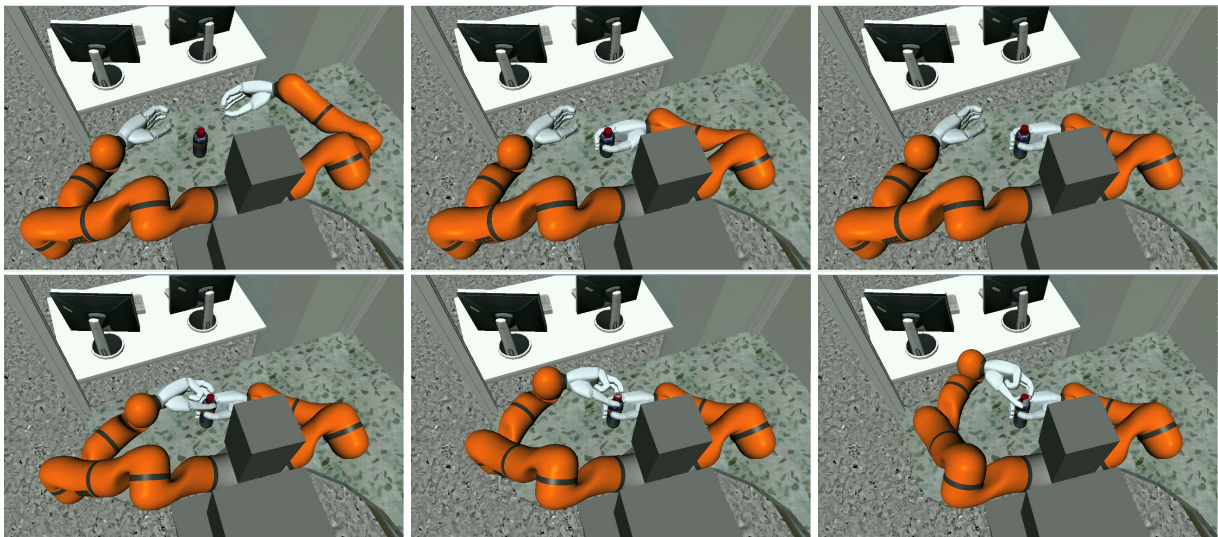


Figure 5.16.: Planning result of the bottle closing task with a smaller bottle and the planning model obtained using all additional demonstrations.

ing during lifting and relative to the bottle bottom during rotating. The resulting planning model was executed successfully with the smaller bottle, see Figure 5.16.

Conclusion

The qualitative results of the experiments show that visual feedback, i.e. the execution of a preliminary planning model in simulation in a different environment, enables the human teacher to interactively remove irrelevant constraints by making additional demonstrations. Nevertheless, the

Experiment	Initial Constraints	Reduced Constraints	Chosen solution	Tests	Validation set	Success (%)
Bottle in fridge	5	2	1	2	10	100
Pour-in	174	60	2	2	15	67
Bottle in crate	15	13	1	3	20	100
Pressing a key	362	28	1	3	15	100
Cup on saucer	26	15	1	4	24	88
Bottle opening	16	8	1	1	20	90
Lifting a chair	12	6	1	1	10	100

Table 5.7.: Results of robot-test-based generalization: *Initial* and *Reduced Constraints* is the number of constraints in the state before and after generalization. *Chosen solution* is the index of the selected solution. *Test* is the number of robot tests. *Validation set* is the number of robot test to determine the generalization error of the optimization result. *Success* is 1 - generalization error.

human teacher requires experience with the learning system to interpret and resolve the failure cause using additional demonstrations.

The resulting planning models were flexible. The chess knight move was executed on a chessboard with a large number of chess pieces, which reduced the free space drastically. In the bottle opening task, the resulting planning model was executed with two different bottles.

5.3.2. Robot Tests

In this subsection, we discuss the evaluation of the robot-test-based generalization algorithm with 7 experiments: bottle in fridge, pour-in, bottle in crate, pressing a key, cup on saucer, bottle opening and lifting a chair. The planning model for the first experiment served as the illustrative example in Section 4.5 and was defined manually. The remaining planning models were learned based on multiple human demonstrations with a single set of objects, i.e. learned constraints encode geometric properties of the objects, which limits generalization to different objects. Focus is on the generalization to new objects. The experiments were published in [49].

The human teacher demonstrated a set of robot tests, which were divided into two sets to calculate the generalization error: training and validation. We executed the generalization algorithm to deduce a maximal set of constraints, which admits a successful solution to the training set. The generalization algorithm was executed on 24 processes in parallel. We used Albert II and Adero in these experiments. The result of the generalization algorithm are multiple planning models sorted by fitness. The planning result of each hypothesis is visualized and the human teacher picks the first solution, which reproduces the task at least visually. The resulting planning model was executed on the validation set. We define a failure as either the maximum planning time of 5 minutes is reached or the human teacher labelled the execution as failed. The latter might happen, if constraints, which are relevant to the task, were removed. Table 5.7 shows a summary of the results. We discuss each experiment in more detail in the following subsections.

Bottle in Fridge

The goal is to grasp a bottle and place it in a fridge door. This task served as an illustrative example in Section 4.5. The position of the bottle in the fridge door was restricted by one constraint relative to the door and three constraints relative to the shelves in the fridge. A contact constraint restricted that the bottle had to be in contact with the fridge door. In the robot tests, the teacher demonstrated different door angles. The planning process failed since the four position constraints were inconsistent. These ambiguities were resolved since only the correct reference, i.e. the fridge door, results in a bottle position, in which the contact constraint is obeyed.

Pour-in

The task was to lift a cup and pour liquid from a bottle into the cup. The robot test was to execute the learned planning model with a cup and a measuring cup. The planning model contained constraints with references to coordinate frames in the robot base, in the opening and bottom of the cup and in the opening and bottom of the measuring cup. The grasping motion for the cup and measuring cup were defined ambiguously by constraints, which restrict the wrist coordinate frame relative to the opening and bottom coordinate frames (since only a single object was used during learning). The inconsistency statistics showed that these constraints were inconsistent in the robot test. Since the false reference coordinate frame led to configurations, in which the grasp quality constraint was violated, the correct constraints were identified.

Problematic are constraints, which implicitly restrict that no fluids are spilled, e.g. by restricting the measuring cup opening to stay above the cup opening. Since the measuring cup is considerably larger than a normal bottle, a larger number of collisions occurs and often no configuration is found, in which these constraints are obeyed. These constraints were deactivated in the beginning and were reactivated one after another by the mutation operator.

The optimization process did not converge and was stopped after 8 hours computation time. The human teacher chose the second solution, which produced the visually correct result in simulation. Nevertheless, the generalized planning model failed on 33% of the validation set. The reason is that a single orientation arc constraint was missing restricting the measuring cup to be only slightly rotated while pouring in. The missing constraint was added after additional 5 hours computation time showing that more elaborate stopping criteria have to be investigated in the future. Figure 5.17 visualizes the constraints before and after generalization.

Bottle in Crate

In this experiment, a bottle had to be grasped and placed in a crate. Coordinate frames were defined in the crate and in the opening and bottom of the bottle. The human teacher demonstrated robot tests with bottles of different sizes and varying bottle and crate poses. The latter is shown in Figure 5.18. During grasping ambiguities similar to the pour-in experiment existed and were

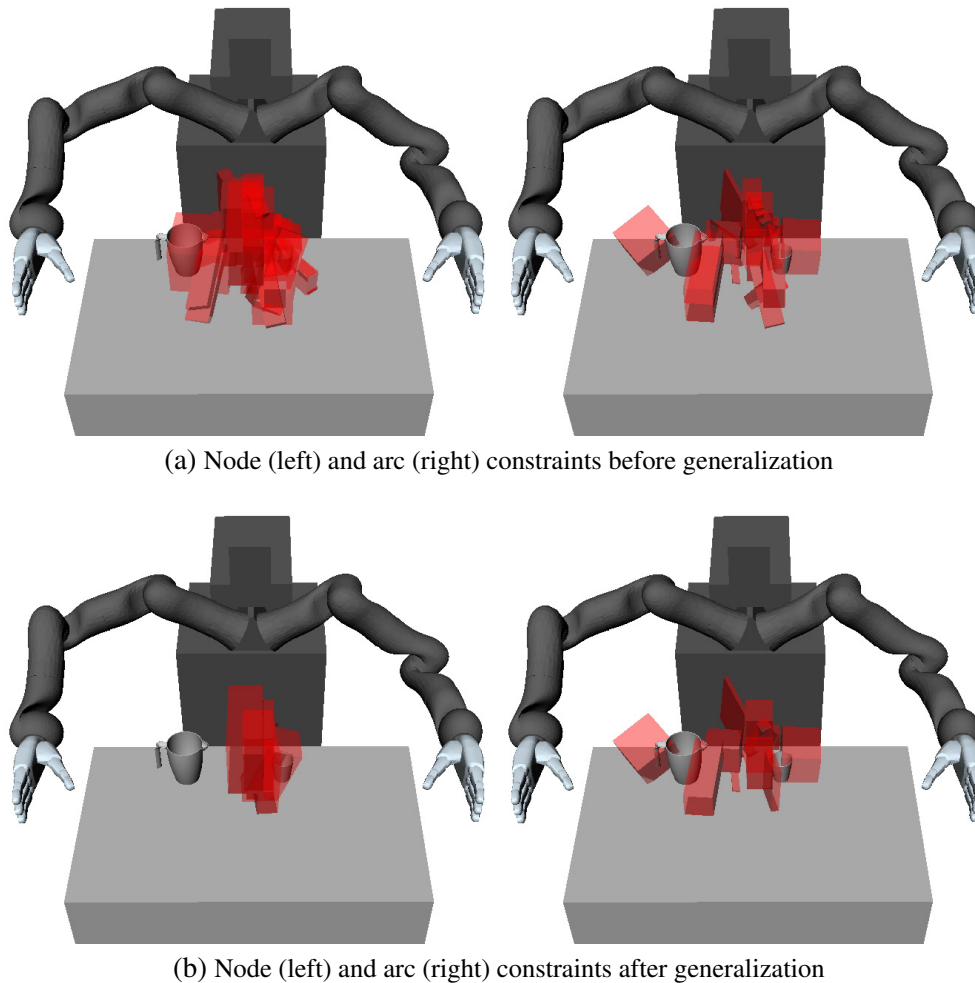


Figure 5.17.: Pour-in: constraints before and after generalization. - [49]

removed successfully, i.e. if the wrist pose is restricted relative to the opening or bottom of the bottle. Orientation constraints for the bottle were removed. Constraints, which restrict the bottle opening relative to the crate were removed since they could not be obeyed due to collisions with the crate. The generalized planning model could be executed successfully with bottles of different size.

Pressing a Key

In this experiment, the goal is to press a key on the keyboard. We modeled the keyboard with 45 keys. In the human demonstrations, the human teacher pressed only the 27th key. The initial planning model contained 362 constraints. In the robot tests, the robot had to press a different key. Constraints, which restrict the finger relative to different keys, were inconsistent, see Figure 5.19. The result is that all position constraints but one are deactivated by the mutation operator. The state is mutated until the position constraint with the correct reference is found. Incorrect references lead to collisions with other keys, contact with the wrong key or no contact at all. The mutation is random, i.e. trial-and-error, leading to high computation times. In the result, the relevant position

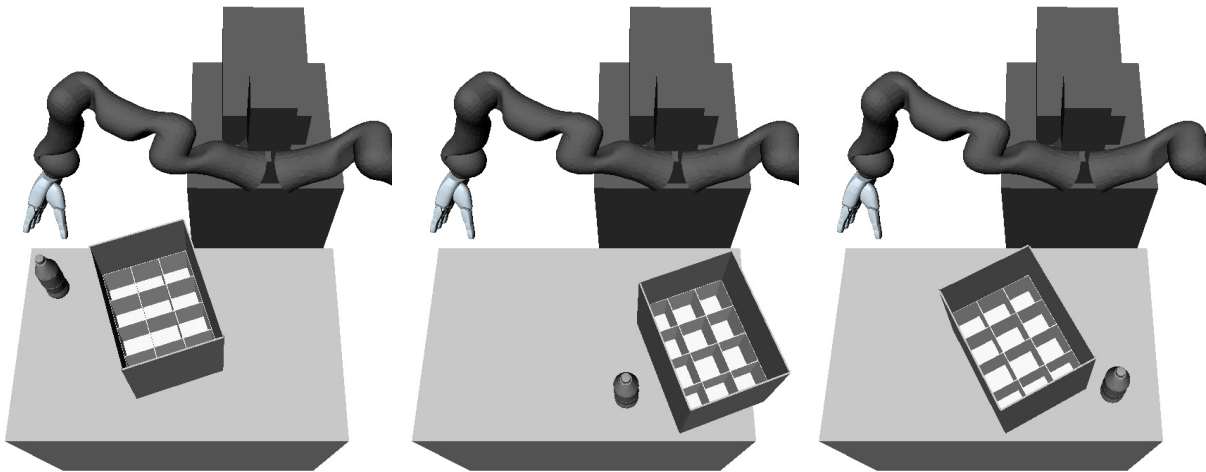


Figure 5.18.: Bottle in crate: examples of robot tests.

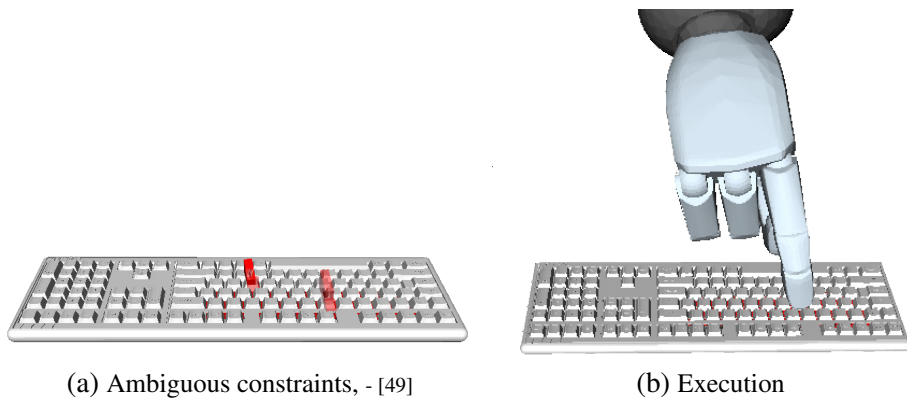


Figure 5.19.: Constraint and execution of generalized planning model to press a key.

constraints, i.e. that the motion of the wrist and finger is relative to the 27th key, remained. The generalized planning model could be executed with arbitrary keys on the same keyboard. Generalization to a keyboard with different keys was not possible since orientation constraints remained, which restricted the wrist and finger relative to different keys on the keyboard.

Cup on Saucer

In this experiment, a cup has to be placed on a saucer. The teacher demonstrated a limited number of cup orientations assuming that the symmetry of the cup is used. To test this assumption, he defined robot tests with arbitrary cup orientations. Constraints restricting the cup orientation were removed and only direction constraints remained approximating the symmetry axis of the cup. Cup and saucer were placed at a large distance to each other, which effectively removed all constraints restricting the cup position relative to the start pose of the cup, and all arc constraints restricting the cup position relative to the saucer. The result contained only arc constraints, which restrict that

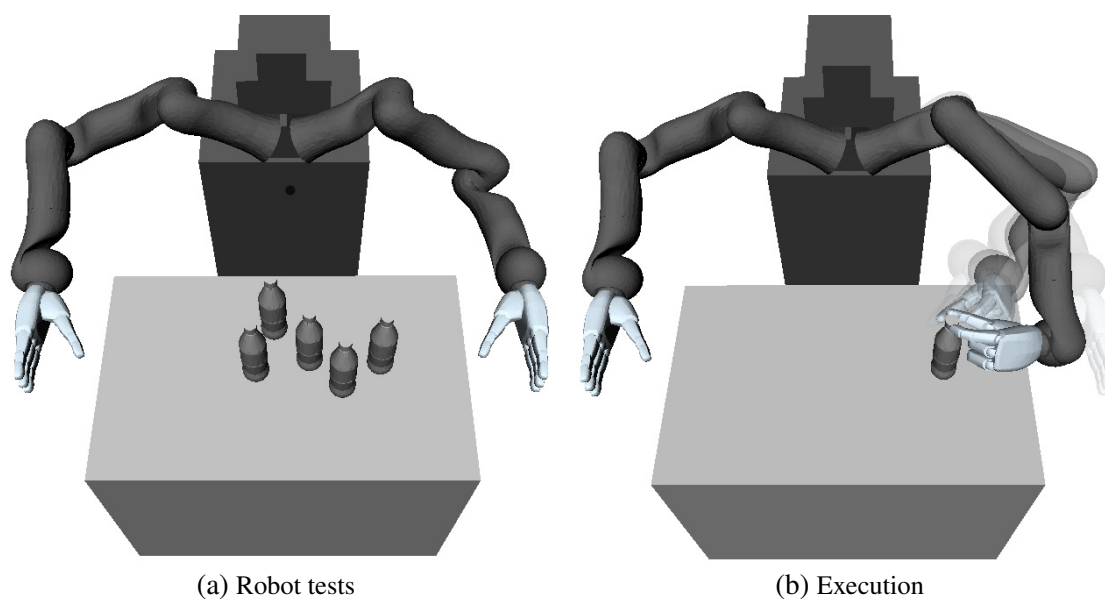


Figure 5.20.: Experiment: bottle opening. - [49]

the cup is transported upright, and node constraints, which restrict that the cup is placed upright on the saucer. The generalized planning result could be executed successfully on the validation set.

Bottle Opening

Similar to Section 5.3.1, a bottle cap had to be grasped and rotated counterclockwise to open the bottle. The human teacher defined robot tests with bottles of varying height, i.e. where the coordinate frames opening and bottom are different. The first step was to grasp the bottle cap. Constraints restricted the wrist position and orientation relative to the bottom and opening of the bottle as well as the center of mass of the cap. In the robot tests, these constraints were inconsistent and only constraints with the correct reference, i.e. the bottle opening and center of mass of the cap, remained. During the rotation, a high number of collisions and workspace violations occurred. Consequently, constraints, which restrict the cap to stay at the same position and only rotate around the z-axis, were initially removed but later reintroduced by the mutation operator. Figure 5.20 shows different robot tests. The arc constraints in the optimization result restricted that the cap had to stay at the same position and had to be rotated around the z-axis. The node constraints restricted that the orientation of bottle had to be in a specific range and the position had to remain unchanged. The generalization error of 10% has to be attributed to the difficulty of the planning problem.

Lifting a Chair

In this experiment, the goal was to grasp a chair and lift it. Coordinate frames were defined in the robot base and chair. Still ambiguities existed, e.g. has the grasping motion to be restricted relatively to the robot base? By using robot tests with different chair poses relative to the robot,

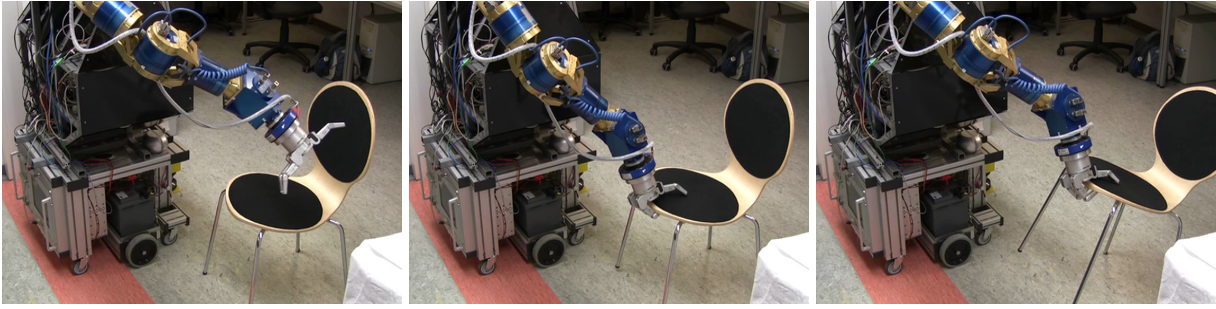


Figure 5.21.: Execution of generalized planning model to lift a chair. - [49]

position constraints, which reference the robot base coordinate frame, were removed. The optimal solution was found, consisting of constraints restricting the wrist position and orientation relatively to the chair. The task was executed on Albert II, see Figure 5.21.

Conclusion

The experiments showed that a small number of robot tests, 2.3 on average, is sufficient to generalize a planning model. In all experiments a planning model was found that solved the robot tests. The average success on the validation set is 92%. The pour-in experiment showed that a better stopping criteria is required for problems with hundreds of constraints.

The human teacher had to verify only 1.1 planning models on average until a solution was found. Since the human teacher is not involved in the time-demanding optimization process but only in the definition of robot tests and verification of optimization results, user interaction is limited. The results show that planning models can be generalized with limited user interaction. The small generalization error indicates that the generalized planning models can be executed in a flexible way in different environments.

5.4. Evaluation of Planning Model Specialization

In this section, learning and planning of search heuristics is analyzed. The results in Section 5.4.1, 5.4.2 and 5.4.3 were published in [55] and [112].

Three manipulation tasks were evaluated: placing a cylinder in a crate (A), placing a bottle in a fridge door (B) and placing a bottle in a box on a rack (C). In each task, three different subtasks were defined (A1, A2, A3, B1, B2, B3, C1, C2, C3), e.g. to analyze scalability to environments with different object poses or obstacles. We explain each subtask in the following subsections.

Experiments were performed both on Albert II and Adero. In each subtask, 100 trials were performed. Maximum planning time was 30 seconds with 2 seconds to generate at most 100 goal configurations in each trial. We ran all experiments with different distance thresholds for clustering $P_d = 100, 150, 200$ and number of trajectories in each cluster $P_c = 3, 5, 7$. The number of Gaussians was fixed to $P_g = 6$. We measured mean planning time μ and standard deviation σ of planning time

Experiment	w/out search heuristics			with search heuristics					s
	μ	σ	e	μ	σ	e	t_0	n	
A1	3.7	1.8	0	0.33	0.01	0	7	1	11.3
A2	5.3	7.8	1	0.73	1.1	0	3	1	7.2
A3	5.7	7.5	4	0.97	2.0	3	4	3	5.9
B1	2.5	0.02	0	0.54	0.08	0	3	1	4.5
B2	3.0	1.2	0	0.89	0.26	0	75	4	3.4
B3	2.7	3.6	0	1.7	3.7	2	3	3	1.6
C1	20.7	13.8	4	0.91	1.3	0	26	1	22.9
C2	4.5	2.6	0	0.92	0.55	0	3	1	4.9
C3	12.6	3.8	2	0.76	1.2	0	20	1	16.6

Table 5.8.: Experiments: results obtained with $P_d = 100$ and $P_c = 3$. Mean μ (in s) and standard deviation σ (in s) and error rate (in %) of planning time. Iteration t_0 , when first search heuristic is learned. Number of learned search heuristics n . The last column (s) shows the speed-up. - [112]

in seconds, the percentage of failed planning attempts e , the number of generated search heuristics n , the time point t_0 , when the first search heuristic was generated and the speed-up s . The speed-up is the quotient of mean planning time without and with search heuristics. Additionally, we evaluated some subtasks multiple times to analyze mean speed-up. Table 5.8 shows a summary of results obtained for parameters $P_d = 100$ and $P_c = 3$. All experiments were executed independently. C3 has a shorter mean planning time as C1 and C2 since different search heuristics were learned resulting in different speed-ups.

5.4.1. Bottle in Crate

The goal of task A is to grasp a bottle and place it in a crate, see Figure 5.22a. The task was executed on Albert II. The goal constraint is visualized in Figure 5.22b. It restricts the coordinate frame in the bottom of the bottle to reside in a cubical region relative to the crate. The main planning problems are to find a collision-free goal configuration in the crate and plan a collision-free path through the narrow passage in the crate.

We considered three subproblems: a fixed bottle and crate pose (A1), random bottle and crate poses (A2) and the following sequence, each executed 100 times before the next: A1, A2, A1 with 1, 2, 3, 4 obstacles in the crate (A3). The sequence of obstacles is shown in Figure 5.23.

Task A1

In A1, the bottle and crate had a fixed position and orientation on the table. Figure 5.24 shows an example of the execution of task A1 using a single search heuristic. A small clustering distance threshold P_d led to search heuristics, which covered only a single compartment in the box, see Figure 5.25a to 5.25c. Therefore, the probability to generate a configuration which collides with the crate was small resulting in a high speed-up. With larger P_d a single search heuristic covered either multiple, see Figure 5.25d, 5.25e, 5.25h and 5.25i, or a single compartment, see Figure 5.25f

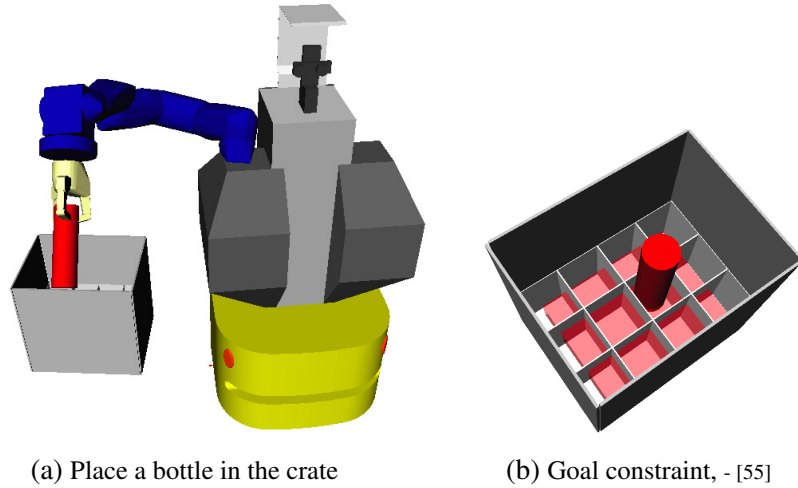


Figure 5.22.: Task A: visualization of task (a) and used goal constraint (b).

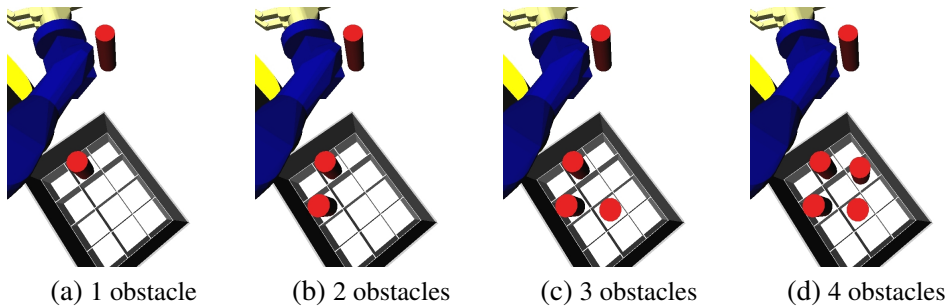


Figure 5.23.: Task A3: obstacle arrangements. - [112]

and 5.25g, depending on the randomly generated training examples resulting in a higher variance of the obtained speed-up.

Table 5.9 shows the planning results with and without search heuristics in task A1. The experiment was repeated 5 times and the average speed-up shows that a smaller P_d led to tighter search heuristics and shorter planning times, see Table 5.10.

Task A2

In subtask A2, the cylinder pose was randomized within $\pm 10cm$ and the crate pose within $\pm 10cm$ and $\pm 45^\circ$. The higher variance in the generated training examples, compare Figure 5.26a and 5.26b, shows the importance of clustering to determine similar training examples.

In the first run of the experiment, planning time increased slightly, see Table 5.11, but a high speed-up of up to 8.7 remained. We repeated the experiment 5 times, see Table 5.12. The maximum average speed-up was 14.1 with $P_d = 100$ and $P_c = 3$, which is significantly larger than the maximum average speed-up of A1 with 9.56. The reason for this increase is the higher average planning time without search heuristics of 5.3s (A2) compared to 3.7s (A1) and the similar plan-

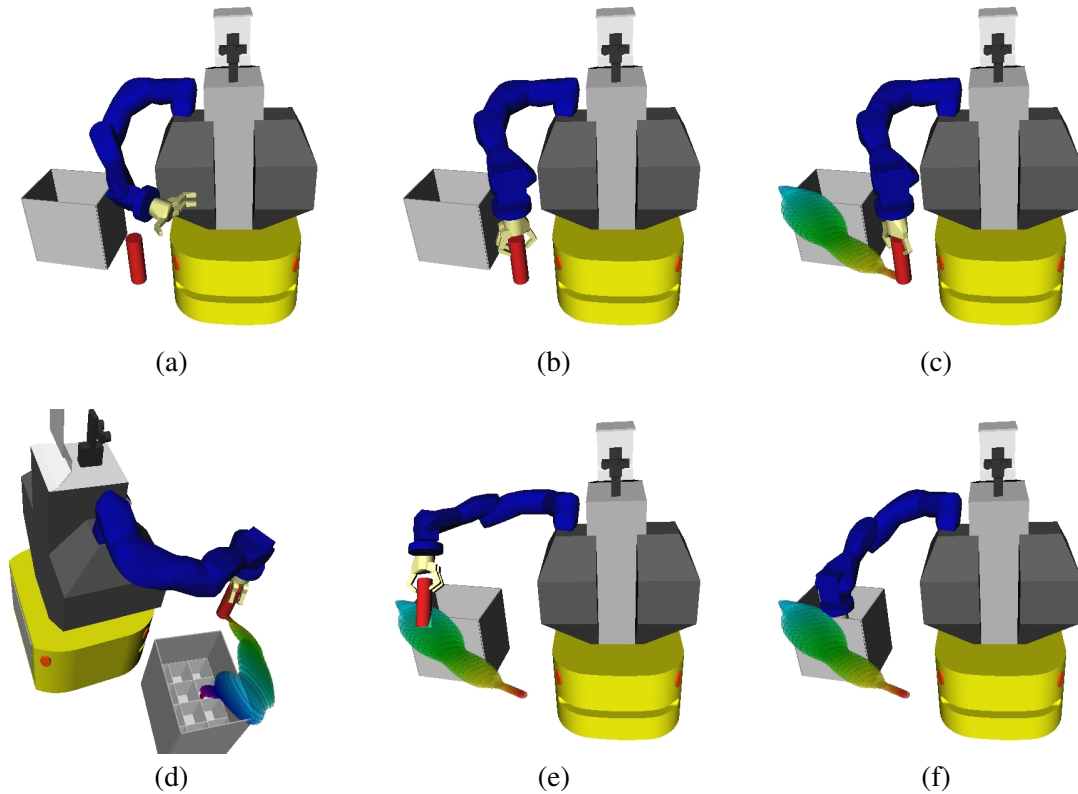


Figure 5.24.: Task A1: execution of task using a single search heuristic. - [112]

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
μ	0.33	0.45	0.37	1.7	0.86	0.37	0.42	0.83	0.69	3.7
σ	0.01	0.29	0.10	2.49	0.98	0.02	0.13	1.43	0.74	1.8
t_0	7	12	39	3	5	12	3	5	7	
s	11.32	8.32	10.17	2.19	4.33	10.08	8.85	4.48	5.39	

Table 5.9.: Task A1: fixed crate and bottle pose. μ is the mean (in s) and σ the standard deviation (in s) of the planning time. t_0 is the iteration, when the first search heuristic is learned. s is the speed-up, P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics. - [112]

ning time with search heuristics, e.g. 0.27s (A2) compared to 0.33s (A1) in the best of 5 runs with $P_d = 100$ and $P_c = 3$. The planning model was executed successfully on Adero, see Figure 5.27.

Task A3

In this subtask, we evaluated the incremental learning of search heuristics by changing the environment during the experiment. A3 started with A1, i.e. a fixed crate and cylinder pose, which is executed 100 times. Followed by A2, i.e. random crate and cylinder poses. Then A1 was executed again but with one, two, three and four obstacles in the crate. Figure 5.28 visualizes the learning process with different parameter settings.

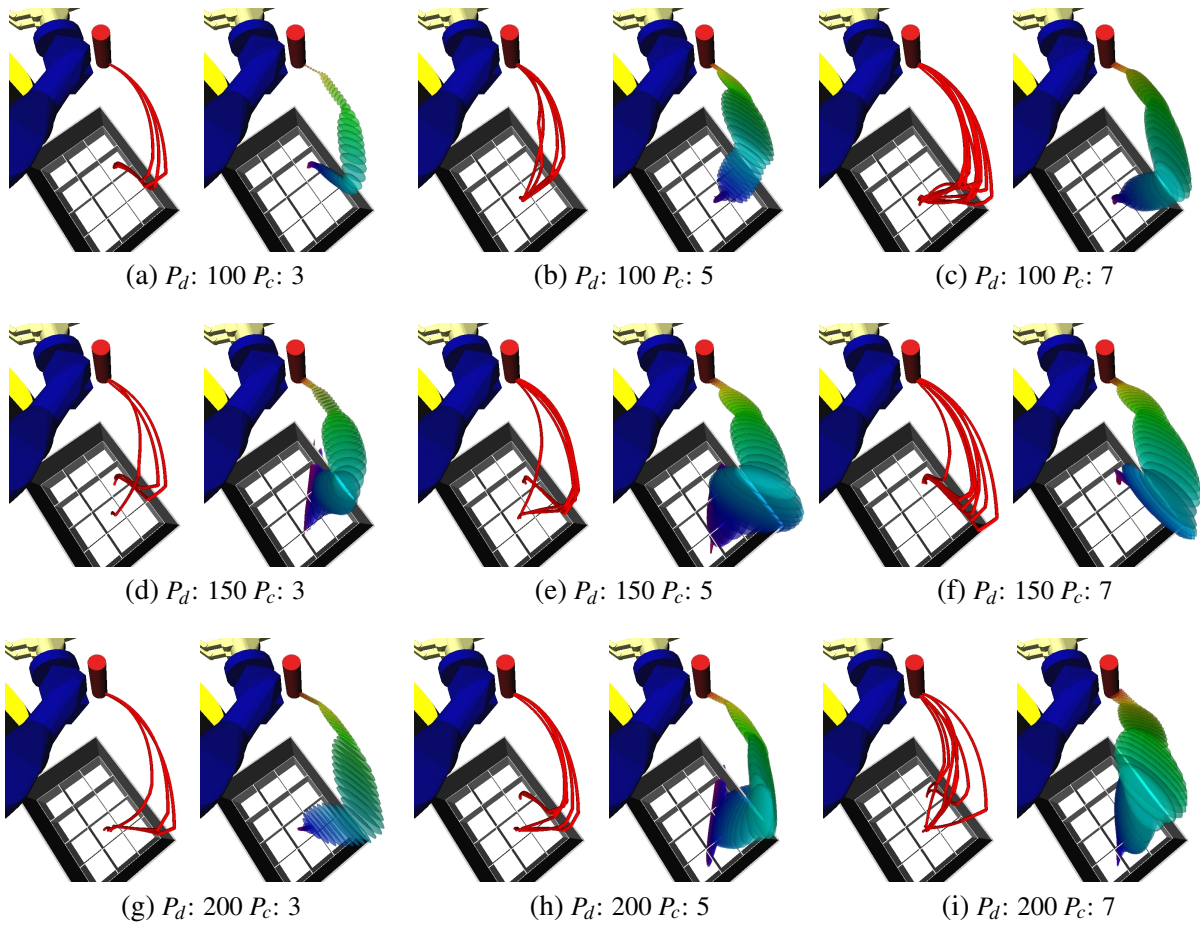


Figure 5.25.: Task A1: training examples and learned search heuristics with different parameters. - [112]

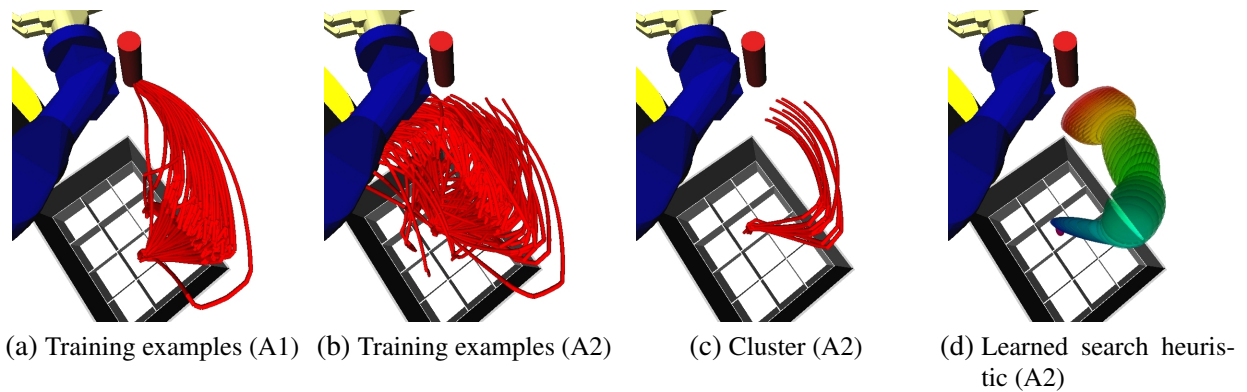


Figure 5.26.: Comparison of training examples with fixed (A1) and random object poses (A2). Example of generated cluster and learned search heuristic. - [112]

P_g	6								
P_d	100			150			200		
P_c	3	5	7	3	5	7	3	5	7
s_1	11.32	8.32	10.17	2.19	4.33	10.08	8.85	4.48	5.39
s_2	11.23	11.04	7.77	3.02	3.80	8.01	11.23	2.91	6.12
s_3	7.89	10.49	5.64	2.80	10.55	8.43	2.99	3.64	2.98
s_4	7.18	6.23	10.46	9.77	5.49	4.52	6.96	4.06	2.35
s_5	10.16	7.15	9.27	3.65	1.82	10.76	1.35	6.64	3.17
μ_s	9.56	8.65	8.66	4.29	5.20	8.36	6.27	4.34	4.00
σ_s	1.92	2.08	1.99	3.11	3.27	2.43	4.08	1.41	1.65

Table 5.10.: Task A1: speed-ups of five runs with each 100 trials. s_i is the speed-up of run i . μ_s the mean of the speed-ups and σ_s the standard deviation of the speed-ups. P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. - [112]

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
μ	0.73	0.73	0.87	0.61	0.66	1.20	1.32	1.14	1.41	5.3
σ	1.11	1.33	2.53	0.18	0.31	1.58	1.76	1.44	2.89	7.8
t_0	3	5	23	3	5	8	3	8	12	
s	7.21	7.25	6.07	8.69	8.03	4.37	3.99	4.61	3.72	

Table 5.11.: Task A2: random crate and bottle pose. μ is the mean (in s) and σ the standard deviation (in s) of the planning time. t_0 is the iteration, when the first search heuristic is learned. s is the speed-up, P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics. - [112]

P_g	6								
P_d	100			150			200		
P_c	3	5	7	3	5	7	3	5	7
s_0	7.21	7.25	6.07	8.69	8.03	4.37	3.99	4.61	3.72
s_1	12.84	12.60	8.90	3.02	6.37	4.48	5.93	9.40	4.94
s_2	14.44	14.56	9.11	8.55	2.80	8.16	8.77	9.01	12.58
s_3	16.70	10.67	14.58	8.56	6.65	7.14	12.72	5.80	6.13
s_4	19.32	11.51	17.87	6.79	9.26	9.77	5.10	12.47	2.97
μ_s	14.10	11.32	11.30	7.12	6.62	6.78	7.30	8.26	6.07
σ_s	4.56	2.70	4.79	2.43	2.43	2.35	3.51	3.12	3.83

Table 5.12.: Task A2: average speed-up. s_i is the speed-up of run i . μ_s the mean of the speed-ups and σ_s the standard deviation of the speed-ups. P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. - [112]



Figure 5.27.: Task A2: execution on Adero. - [55]

We discuss the example in Figure 5.28a. During the execution of A1 the first search heuristic was learned. The search heuristic could still be applied, when the problem changed to A2 and A1 with one and two obstacles, see Figure 5.29a. The third obstacle blocked the learned search heuristic. The planning process failed and planning without search heuristics was used to generate new solutions. After a few iterations, a new search heuristic was learned and planning time decreased, see Figure 5.29b. The novel search heuristic was blocked by the fourth obstacle and a new search heuristic was learned, see Figure 5.29c.

Nearly identical results were obtained with $P_d = 100$ and $P_c = 5$, see Figure 5.28b. Figure 5.28c shows the results with $P_d = 100$ and $P_c = 7$. In this case, a search heuristic was learned which could be applied successfully until it was blocked by the fourth obstacle. A second search heuristic was learned.

In this experiment, a smaller but still significant speed-up was obtained, e.g. a speed-up of 5.9 with $P_d = 100$ and $P_c = 3$. In all trials, at most 3 search heuristics were learned indicating that search heuristics can be used flexibly in different environments.

5.4.2. Bottle in Fridge

In this experiment, the goal was to grasp a bottle and place it in a fridge door, see Figure 5.30. In contrast to Task A, an arc orientation constraint was included, which restricted the bottle to stay upright during the motion. Additionally, the bottle had to be placed in a continuous space in the fridge door, see Figure 5.30a.

Three subtasks were evaluated: no obstacles (B1), random obstacle pose in fridge door (B2) and the sequence of both (B3).

Task B1

Figure 5.31 shows learned search heuristics in B1. Since no obstacle blocked the goal and the initial bottle pose was fixed the planner produced similar trajectories in each run. Therefore, the variance in the learned search heuristics was low compared to task A and the speed-up was stable for different parameter sets. The best average planning time with search heuristics was 0.53 seconds

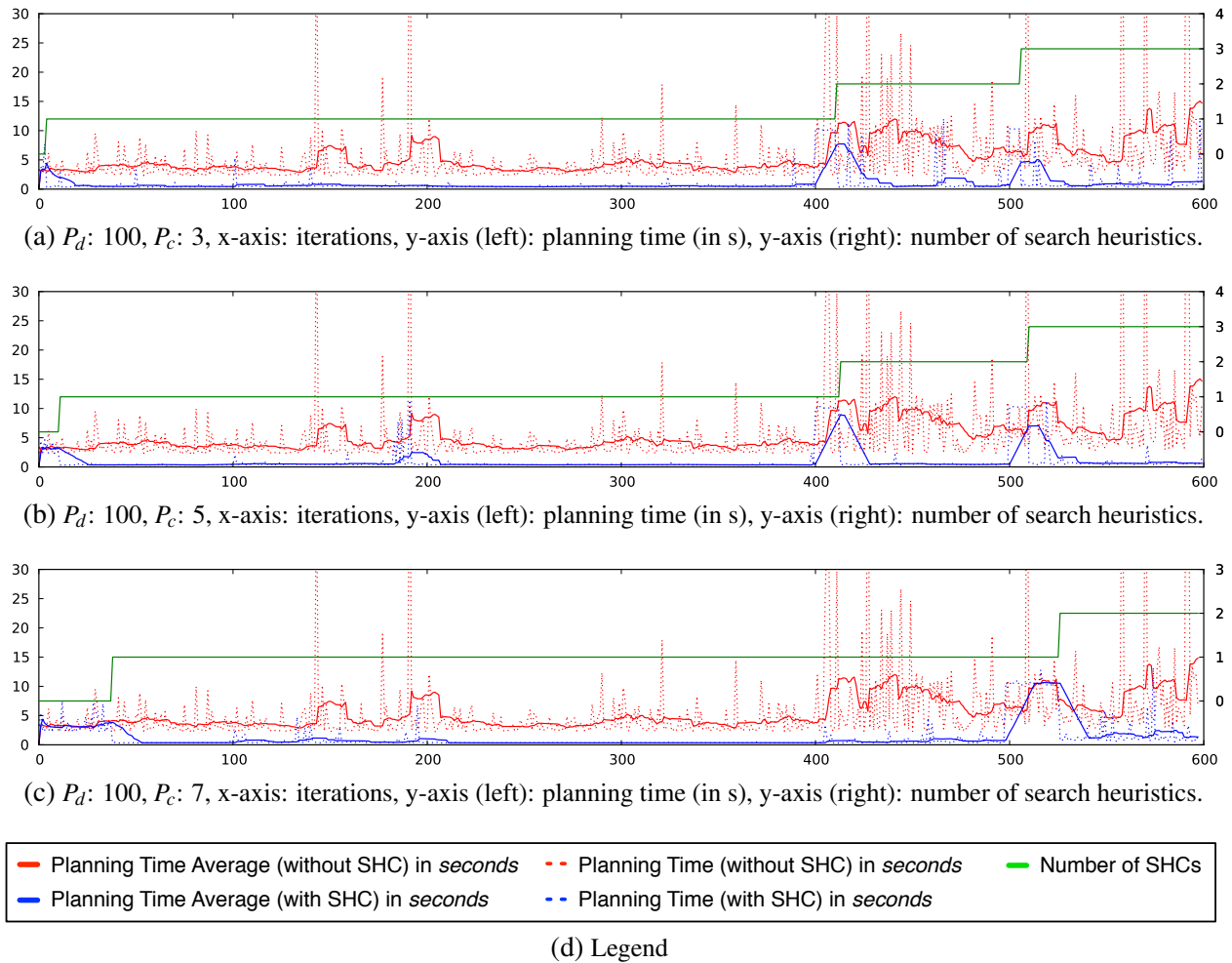


Figure 5.28.: Task A3: incremental learning of search heuristics with different parameters. - [112]

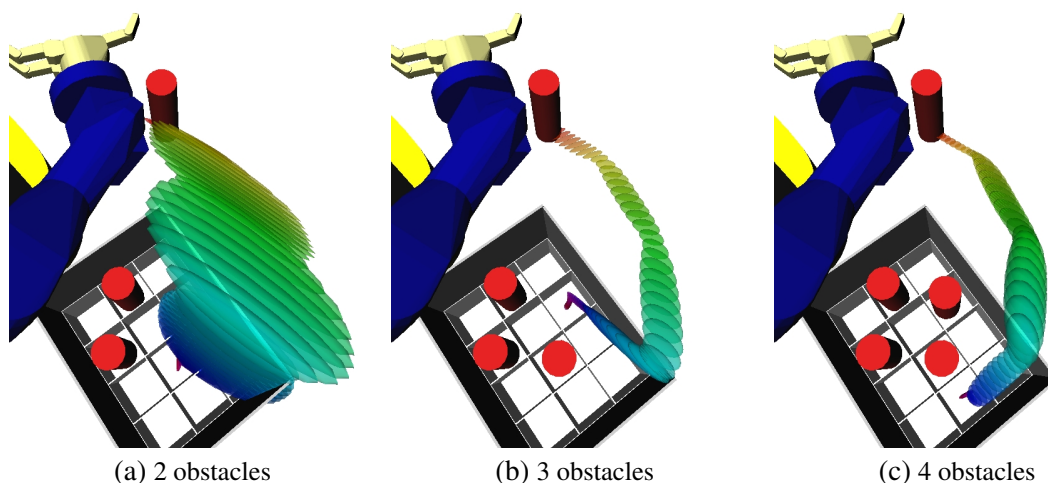


Figure 5.29.: Task A3: a search heuristic was learned and used successfully with two obstacles (a). In (b), a third obstacle blocked the search heuristic in (a) and a new search heuristic is learned. The process repeated with the fourth obstacle (c). - [112]

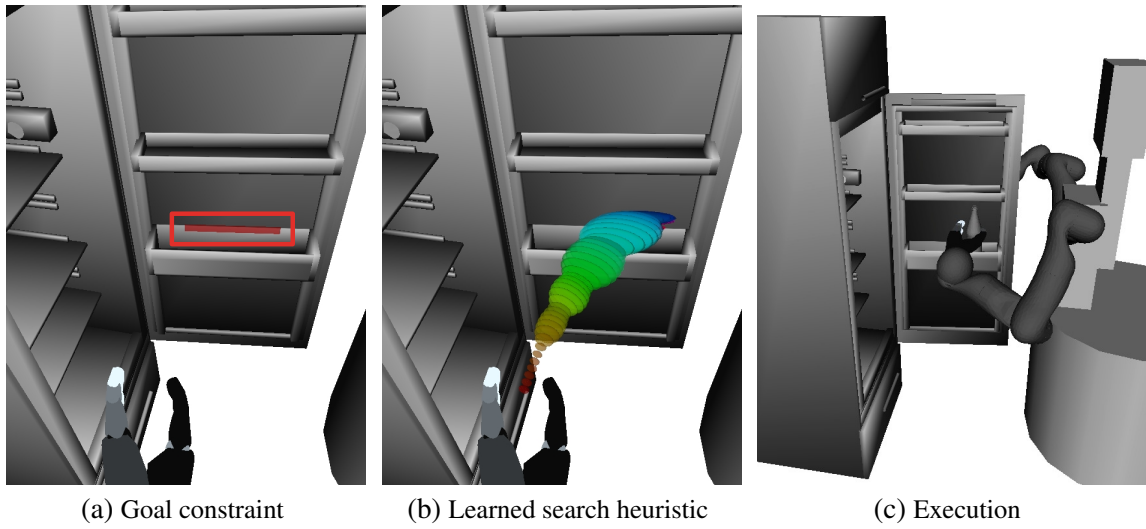


Figure 5.30.: Task B: grasp a bottle and place it in the fridge door. - [112]

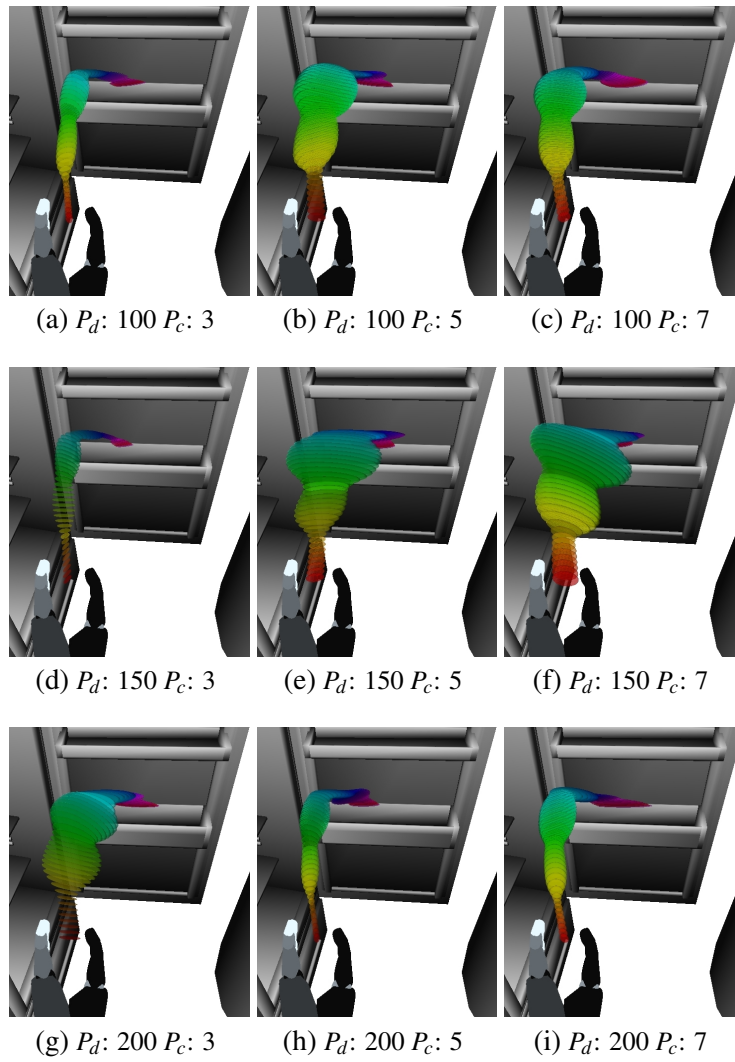


Figure 5.31.: Task B1: learned search heuristics with different parameters. - [112]

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
μ	0.54	0.60	0.61	0.57	0.58	0.54	0.53	0.78	0.61	2.5
σ	0.08	0.16	0.11	0.08	0.06	0.06	0.08	0.22	0.18	0.15
t_0	3	5	11	3	5	7	3	5	7	
s	4.53	4.08	4.04	4.29	4.22	4.53	4.60	3.14	4.06	

Table 5.13.: Task B1: no obstacles. μ is the mean (in s) and σ the standard deviation (in s) of the planning time. t_0 is the iteration, when the first search heuristic is learned. s is the speed-up, P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics. - [112]

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
μ	0.89	1.72	2.11	1.18	1.11	2.04	2.47	1.64	1.09	3.0
σ	0.26	2.66	4.80	1.42	1.57	4.61	4.09	2.40	0.54	1.18
t_0	75	37	26	24	21	34	17	20	33	
s	3.35	1.74	1.41	2.54	2.68	1.46	1.21	1.82	2.75	

Table 5.14.: Task B2: random obstacle poses. μ is the mean (in s) and σ the standard deviation (in s) of the planning time. t_0 is the iteration, when the first search heuristic is learned. s is the speed-up, P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics. - [112]

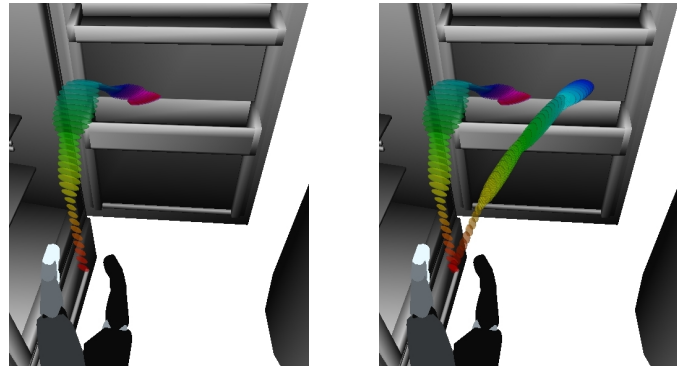
and without 2.5 seconds indicating that the planning problem is simple. The highest speed-up was 4.6, see Table 5.13.

Task B2

In this subtask, a second bottle was placed at a random position in the fridge door as an obstacle. The planning time without search heuristics was slightly higher compared with B1 since the obstacle was small and therefore blocked only a small portion of the free space, see Table 5.14. Although two search heuristics were learned at most, see Figure 5.32, planning time was not affected by the time to select a search heuristic. It could be identified efficiently when a search heuristic was blocked: no goal configuration was found, which lay in the intersection of the last ellipsoid of the search heuristic and the goal constraint. A maximum speed-up of 3.35 was obtained.

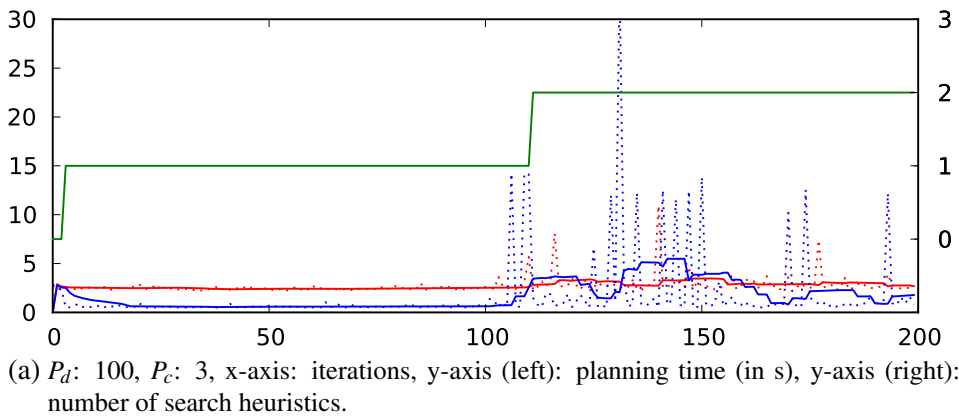
Task B3

In the last subtask, we executed B1 100 times followed by B2 100 times. Figure 5.33 shows the resulting planning times. After a few iterations the first search heuristic is learned and the planning time is reduced. When the random obstacle is added, the search heuristic is blocked multiple times, new training data is generated and the second search heuristic is learned after a few iterations. The average speed-up of 5 runs is between 1.6 and 2.3, see Table 5.15.

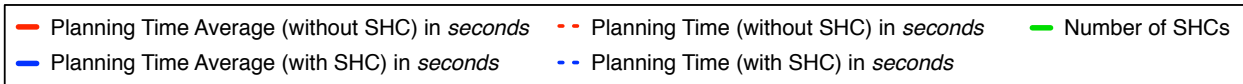


(a) First search heuristic (iteration 5) (b) Second search heuristic (iteration 12)

Figure 5.32.: Task B2: learned search heuristics ($P_c: 3, P_d: 100$). - [112]



(a) $P_d: 100, P_c: 3$, x-axis: iterations, y-axis (left): planning time (in s), y-axis (right): number of search heuristics.



(b) Legend

Figure 5.33.: Task B3. B1 is executed (1-100) followed by B2 (101-200). - [112]

P_g	6								
P_d	100			150			200		
P_c	3	5	7	3	5	7	3	5	7
s_0	3.2	1.7	1.4	2.4	2.6	1.4	1.2	1.7	2.6
s_1	3.1	1.4	2.7	1.3	1.3	1.8	2.5	2.5	2.8
s_2	1.7	1.5	1.5	2.5	1.1	2.3	1.3	2.6	1.9
s_3	1.7	1.7	2.4	1.4	1.9	1.7	1.6	2.1	1.4
s_4	1.6	2.2	2.4	2.0	2.1	2.0	1.7	1.5	1.9
μ_s	2.3	1.7	2.1	1.9	1.8	1.8	1.6	2.1	2.1
σ_s	0.84	0.30	0.61	0.56	0.61	0.35	0.51	0.45	0.59

Table 5.15.: Task B3: average speed-up. s_i is the speed-up of run i . μ_s the mean of the speed-ups and σ_s the standard deviation of the speed-ups. P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. - [112]

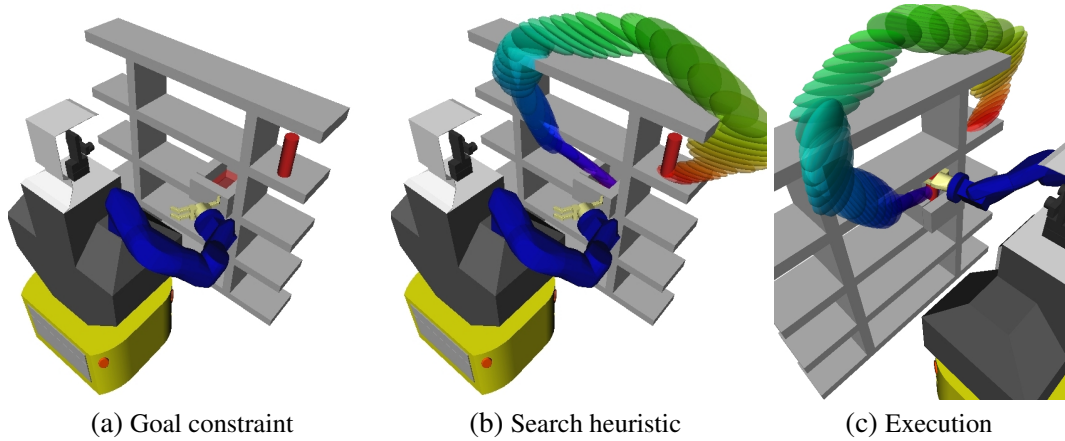


Figure 5.34.: Task C: grasp a chips can and place it in a box on a rack. - [112]

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
μ	0.90	0.40	1.9	0.78	0.91	0.79	0.57	0.64	1.2	21
σ	1.3	0.04	3.5	1.5	0.49	1.9	0.17	1.1	2.8	14
t_0	26	24	36	6	8	25	7	22	20	
s	22	52	11	27	23	27	37	33	17	

Table 5.16.: Task C1: fixed box position in rack. μ is the mean (in s) and σ the standard deviation (in s) of the planning time. t_0 is the iteration, when the first search heuristic is learned. s is the speed-up, P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics. - [112]

5.4.3. Chips Can to Rack

In task C, the goal is to grasp a chips can and place it in a box on a rack, see Figure 5.34. In the first subtask, the box pose is fixed (C1). In the second subtask, the box is placed at random positions with the same orientation on the rack (C2). In C3, C1 and C2 are executed one after another, each 100 times.

Task C1

Table 5.16 shows the planning results for C1. The average planning time without search heuristics was 21 seconds. We deduce that the large rack in a small distance to the robot restricts its workspace severely. Figure 5.35 shows the learned search heuristics for different parameter sets. A high variance of learned search heuristics indicates a high variance in the planned trajectories. Each search heuristic represents an alternative solution to the task. The effect on the learning process can be observed on t_0 , i.e. the time point, in which the first search heuristic was learned, see Table 5.16. t_0 was significantly higher than for task A and B. Since the planning problem was difficult, a high speed-up between 11 and 52 was reached. The task was executed successfully on Albert II, see Figure 5.36.

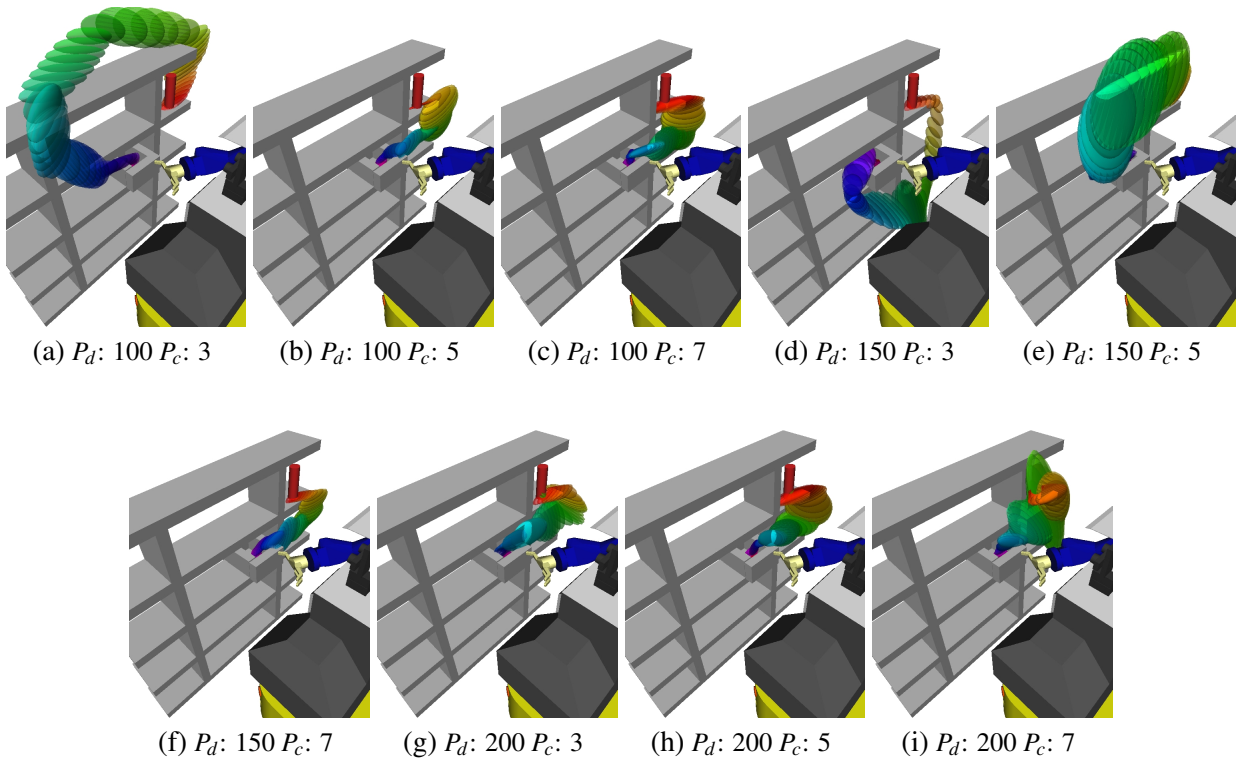


Figure 5.35.: Task C1: learned search heuristics with different parameters. - [112]

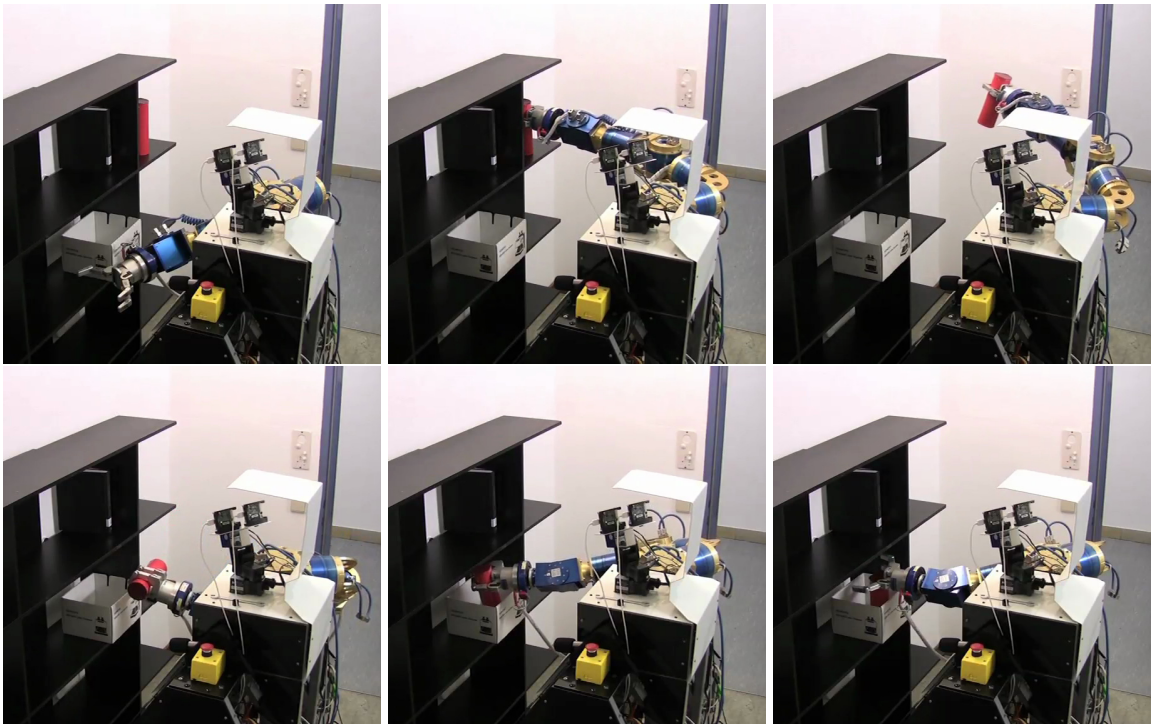


Figure 5.36.: Task C1: execution on Albert II. - [55]

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
μ	0.92	0.64	0.87	0.64	0.86	0.73	0.68	0.99	0.93	4.5
σ	0.55	1.1	0.82	0.52	1.4	1.2	0.54	1.7	1.4	2.6
t_0	3	8	16	4	7	13	3	6	18	
s	4.9	7.0	5.1	7.0	5.2	6.1	6.6	4.5	4.8	

Table 5.17.: Task C2: random box positions in rack. μ is the mean (in s) and σ the standard deviation (in s) of the planning time. t_0 is the iteration, when the first search heuristic is learned. s is the speed-up, P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics. - [112]

Task C2

In this subtask, the box was not fixed but placed at random positions with the same orientation on the same shelf. Since the box position in C1 was extremely close to the side of the rack the average planning time without search heuristics was significantly smaller for C2 with 4.5s compared to C1 with 21s, see Table 5.17. Planning time with search heuristics was nearly identical. The result was a smaller speed-up compared to C1 between 4.5 and 7.0 depending on the parameter set. A single search heuristic was sufficient in all trials.

The robot had to move the chips can in a complex way near to the box, i.e. rotate the chips can while moving it inside. The controller reproduced this motion near to the object based on the learned search heuristics, resulting in shorter planning time. The larger part of the motion, i.e. to move around the rack pieces to avoid collisions, was generated using the planner but planning time was relatively small due to the large free space.

Task C3

The result of C3 is shown in Figure 5.37. C1 was more difficult than C2 resulting in high mean and variance of the planning time without search heuristics. After a few iterations, a search heuristic was learned and a high speed-up was achieved. At iteration 100, the settings switched to C2. The mean and variance of the planning time dropped but remained significantly higher than the planning time with search heuristics. The variance in the planning time was reduced greatly by using the learned search heuristic. The main reason is that the last part of the motion, which is difficult to plan without search heuristics, could be reproduced using the search heuristics resulting in a small variance.

5.4.4. Comparison with Control-Approach

We compared two mechanisms: planning with search heuristics and the execution of the control algorithm without planning. Two tasks were used: place a bottle in a crate with random poses and place a bottle in the fridge with a random obstacle. In both tasks, a single search heuristic was learned and used with both mechanisms.

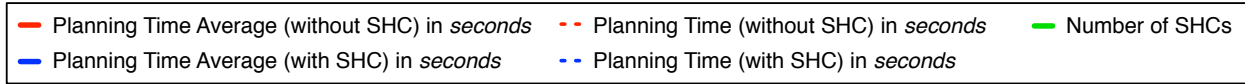
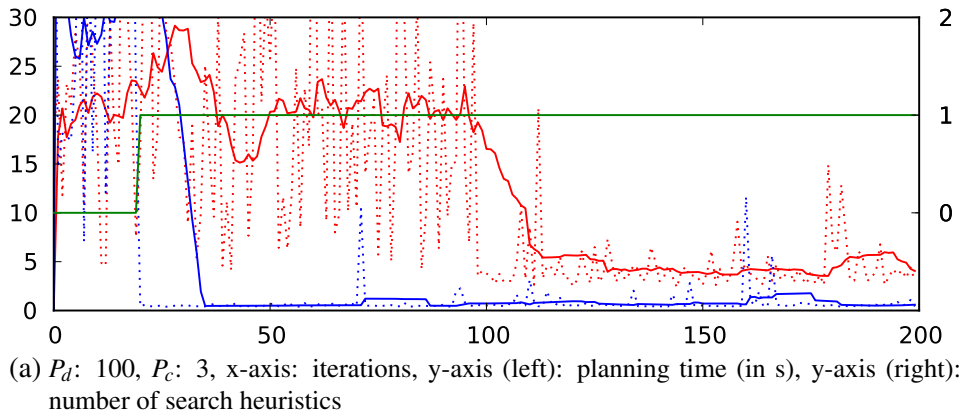


Figure 5.37.: Task C3. C1 is executed (1-100) followed by C2 (101-200). - [112]

	Task	Time (success) (s)	Time (s)	Success (%)
Planning	Bottle in crate	1.94	4.43	80
	Bottle in fridge	2.12	2.12	100
Control	Bottle in crate	1.96	12.6	51
	Bottle in fridge	3.86	14.8	56

Figure 5.38.: Comparison with Control Algorithm: planning results show a higher success rate in both tasks.

We executed the control algorithm, which is used in the constraint-based motion planner, see Section 4.6.5, starting from the current robot configuration, i.e. we did not plan to the beginning of the search heuristic. In the control algorithm, the robot motion is adapted depending on the deviation of the current configuration to the mean of the search heuristics in the next time point. The motion will follow the mean the closer the smaller the variance in the next time point.

Figure 5.38 summarizes the results. We executed each task 100 times. In the first task, the execution time is similar (if successful) but the success rate is higher $80\% > 51\%$ when using planning with search heuristics. In the second task, planning with search heuristics has also superior success rate $100\% > 56\%$ and planning time $2.12s < 3.86s$. The results indicate that the combination of planning and control-based execution of search heuristics is advantageous if collisions in the environment and the workspace limitations require to deviate from the trajectories, which are encoded in the search heuristic.

5.4.5. Conclusion

Table 5.18 shows an overview of the average speed-ups obtained in the tasks A1, A2, B1, B2, C1 and C2. A3, B3, C3 are omitted since their purpose was to analyze the incremental learning. Each experiment was executed 5 times. The maximum speed-up was obtained with $P_d = 100$ in 67% of the experiments including the 3 planning problems with highest planning time without search

P_g	6									no Shc
P_d	100			150			200			
P_c	3	5	7	3	5	7	3	5	7	
s_{A1}	9.5	8.7	8.6	4.3	5.3	8.4	6.2	4.4	4.0	11.3
s_{A2}	13.9	11.4	11.0	7.2	6.9	6.6	7.2	8.2	6.0	7.2
s_{B1}	4.4	4.5	4.2	4.2	4.3	4.5	4.6	4.1	4.3	4.5
s_{B2}	2.3	1.7	2.1	1.9	1.8	1.8	1.6	2.1	2.1	3.4
s_{C1}	16.7	32.4	33.9	30.6	28.4	23.4	32.9	24.7	30.8	22.9
s_{C2}	5.0	6.3	5.7	5.6	6.1	6.6	5.4	4.2	6.5	4.9

Table 5.18.: Overview of average speed-up s_i in experiments A1, A2, B1, B2, C1, C2. Best speed-up (red) and individually for each P_c (blue). P_g the number of Gaussians, P_d the distance threshold for clustering and P_c the number of trajectories in a cluster. *No Shc* refers to planning without search heuristics.

heuristics: A1, A2, C1. The minimum speed-up was 1.63 and the maximum 33.92. For each P_d , the best speed-up was obtained with $P_c = 3$, i.e. a minimum number of three elements in a cluster, in 50% of experiments, compared to 17% with $P_c = 5$ and 33% with $P_c = 7$.

The results indicate that high speed-ups to plan manipulation tasks can be achieved using search heuristics: 9.5 (A1), 13.9 (A2), 4.6 (B1), 2.3 (B2), 33.9 (C1) and 6.6 (C2). In general, the distance threshold parameter P_d had the biggest impact on the obtained speed-ups. With $P_d = 100$ and $P_c = 3$, the average speed-up was 86.8% of the maximum speed-up with all parameters. The lowest speed-ups in each experiment were 4.0 (A1), 6.0 (A2), 4.1 (B1), 1.7 (B2), 16.7 (C1) and 4.2 (C2). This corresponds to 42.1% (A1), 43.2% (A2), 89.1% (B1), 70.0% (B2), 49.3% (C1) and 63.6% (C2) of the maximum speed-up with all parameters. The latter shows that a high speed-up can be obtained even with suboptimal parameters.

5.5. Evaluation of Learning and Planning of Dexterous Manipulation Tasks

In this section, we evaluate two simple dexterous manipulation tasks: opening a bottle with two fingers and lifting a spoon for grasping with a single finger. Preliminary results for both tasks were published in [53] but with different experimental data. In the first task, we created the planning model manually. The goal is to evaluate planning time and learning of search heuristics to consider the correspondence problem for manually created planning models. The second task was learned based on constraint set 2. We attached one of the magnetic field motion trackers to each finger to get accurate fingertip measurements, which do not suffer from occlusions. The motion of the spoon was measured with the third magnetic field motion tracker.

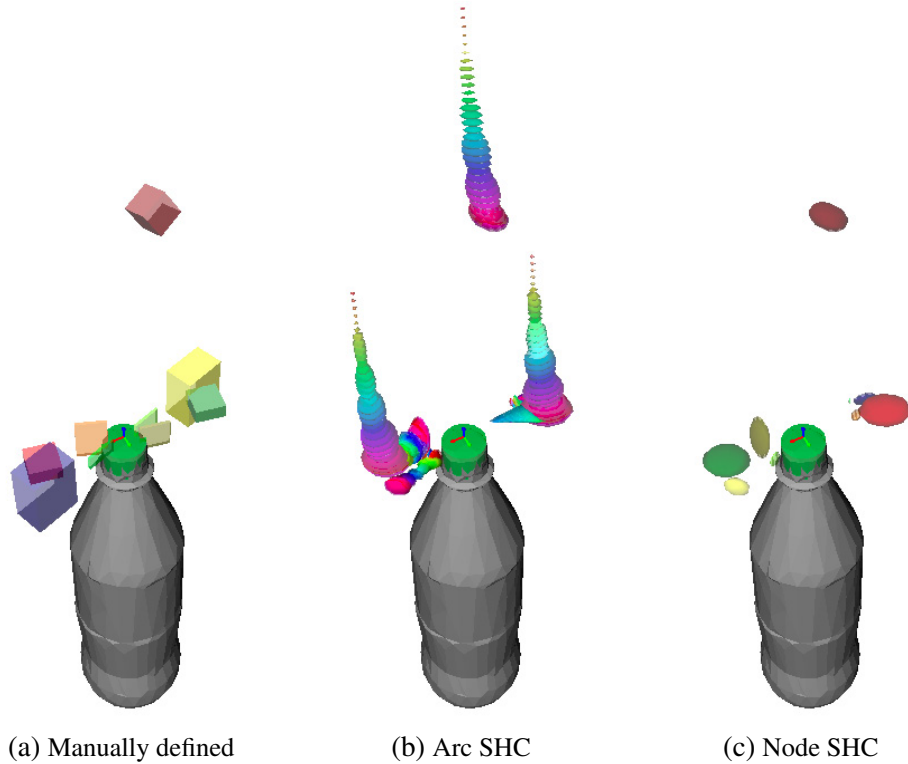
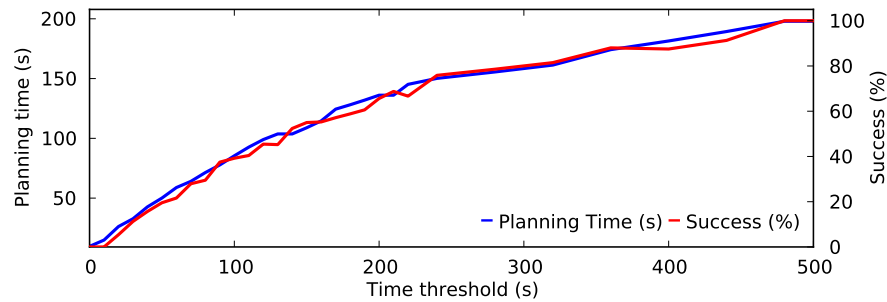


Figure 5.39.: Opening a bottle: visualization of constraints in manually generated planning model and learned search heuristic constraints (SHC).

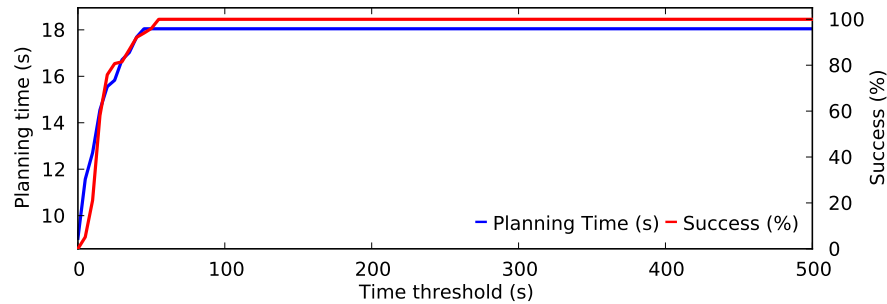
5.5.1. Opening a Bottle

In this task, the goal is to rotate a bottle cap with two fingertips only. The planning model was generated manually. We evaluate planning time for this dexterous manipulation task without and with learned search heuristics.

Figure 5.39a shows the constraints in the manually created planning model. Manual definition took about 3 hours by an expert. It proved difficult to define constraints for the wrist pose and fingertips, which can be reached by the robot arm and fingers. The planning model consists of 4 segments, i.e. 4 arcs and 5 nodes. The first segment describes the approach motion of the hand, in which only the hand motion relative to the object was considered. On the second segment, the fingers are closed until contact with the bottle cap. Since the hand is held still, only constraints describing the finger motion relative to the cap were used. Object constraints describe the cap motion relative to the start pose of the cap, i.e. the cap rotates around the z-axis and stays at the same position. The third segment represents the motion of the fingers to rotate the cap. Finger constraints and object constraints were added as in the previous segment. The object node constraint defines the relevant task goal: the cap has to stay at the same position, $\pm 5mm$, and it is rotated around the z-axis relative to the cap between 35° and 65° . On the last segment, the fingers are opened and finger as well as object constraints are added.



(a) Manually defined planning model



(b) Planning model with learned search heuristics

Figure 5.40.: Opening a bottle: planning time (s) and success rate (%) without and with learned search heuristics with different time thresholds (s).

In the execution environment, the task was planned with the same bottle, which was used during the definition of the planning model. Constraints for the fingertips and wrist were extended automatically to consider the correspondence problem. In the physics simulation, a simplified model of the cap was used: the 3D model of the bottle is `CokePlasticsLarge` in the KIT Object Models database, the cap is a cylinder. The cap is allowed to rotate freely around the z-axis and $\pm 1^\circ$ around the x-axis and the y-axis. The position can change by $\pm 1\text{mm}$ in each direction. Joint friction was added to simulate the friction of the bottle screw. No collisions between the cap and the bottle are considered. Since no contact constraints between the robot fingers and the bottle were defined, only contacts between the robot fingers and the bottle cap were allowed.

The planner ran in a loop until either a planning solution was found or the overall time threshold was reached after the planner had finished. The generated planning solution was executed in physics simulation. If a constraint is violated, i.e. the bottle cap was not rotated sufficiently, the execution fails. The average planning time was 58.8s and success rate 21.6% with a time threshold of 60s. With a time threshold higher than 500s success rate was 100%. The average planning time of a solution with successful execution was 33.7s. The planning problem is difficult since the fingers have to be moved in a coordinated way, the cap rotation is simulated based on the forces applied through the finger tips and a final cap orientation has to be in a specific range of $\pm 15^\circ$. Figure 5.40 shows the dependency of the planning time on the time threshold.

After learning search heuristics, the average planning time dropped to 18.0s with success rate 100% (time threshold 60s) or 16.7s with success rate 81.5% (time threshold 30s). Figure 5.39b

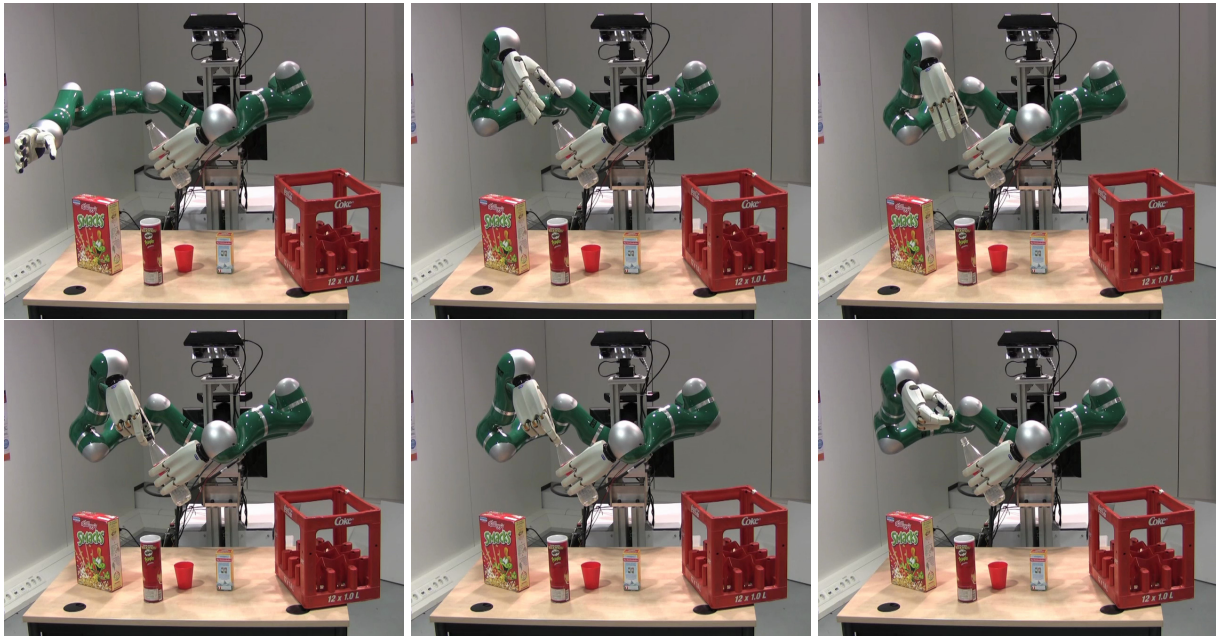


Figure 5.41.: Opening a bottle: execution in similar setting on Adero. - [53]

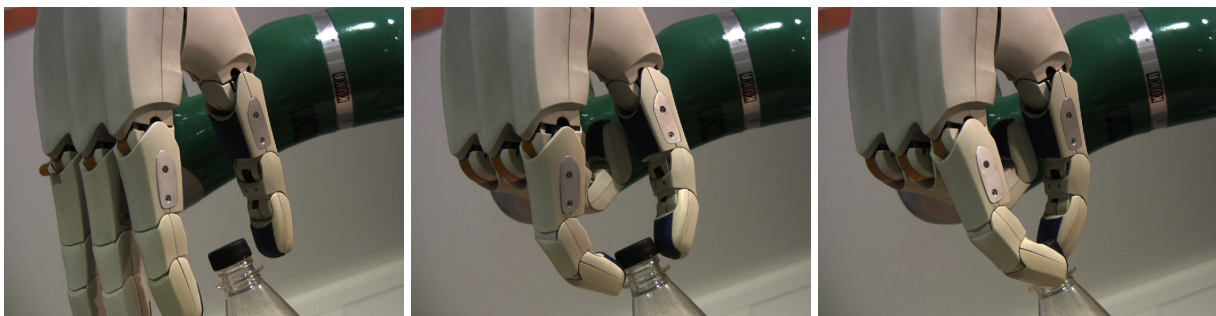
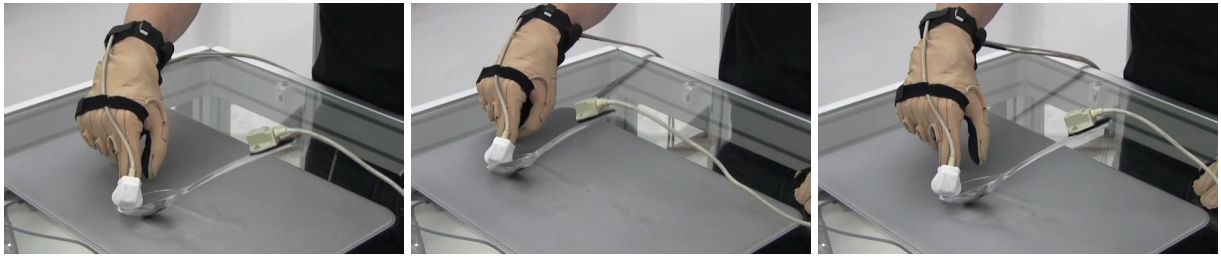


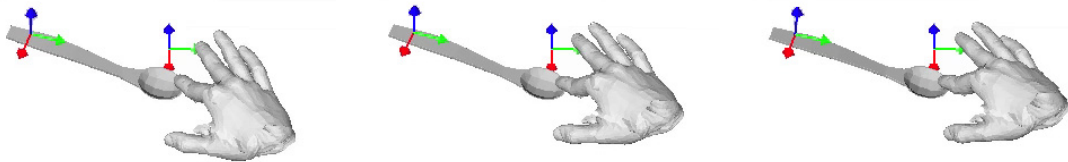
Figure 5.42.: Opening a bottle: close-up of execution in similar setting on Adero. - [53]

shows a set of learned search heuristic constraints. The three identical sequences of ellipsoids correspond to the fingers, which are held still while moving the arm towards the bottle. We project the set of search heuristics to the manually defined node constraints. Figure 5.39c shows the specialized set of constraints. Most specialized constraints are included in the manually defined constraints but some constraints, e.g. green and red, are slightly larger showing the necessity to relax constraints to consider the correspondence problem. Figure 5.41 and Figure 5.42 show the execution of the task in a similar setting on Adero.

The results indicate that manually defining a planning model for dexterous manipulation tasks is time-demanding. The planning time was high due to suboptimal constraint regions, which had to be relaxed to consider the workspace restrictions of the robot fingers. Based on the automatically learned search heuristics, a significant reduction in planning time and an increase in success rate were obtained.

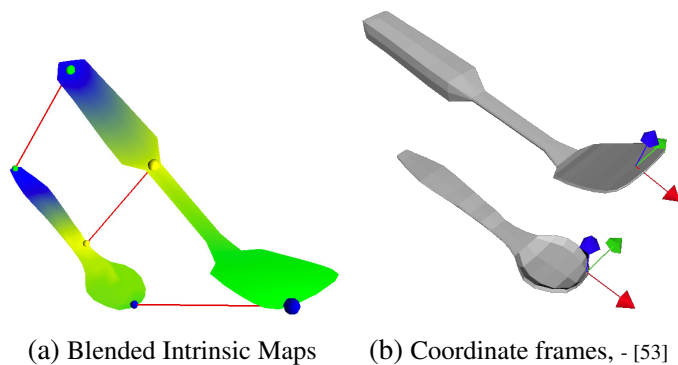


(a) Human demonstrations in sensor environment, - [53]



(b) Visualization of three human demonstrations with different hand postures

Figure 5.43.: Lifting a spoon: human demonstrations.



(a) Blended Intrinsic Maps

(b) Coordinate frames, - [53]

Figure 5.44.: Lifting a spoon: mapping of contact coordinate frames.

5.5.2. Lifting a Spoon

In this task, the goal is to push with one finger on a spoon to lift it and grasp the handle with the other hand. This manipulation strategy is required since the robot was not able to grasp the spoon while it lay on the table. Figure 5.43 shows the human demonstrations. The human teacher demonstrated the task 7 times with a small spoon. The elastomer-based tactile sensors provided sufficient measurements of the contacts and forces between the fingertip and the spoon. In the learning algorithm, two contact coordinate frames relative to the center of mass of the spoon were generated. The initial planning model contained 140 constraints.

We applied the robot-test-based generalization algorithm to remove irrelevant constraints. The robot-test consisted of a different spoon, i.e. a large spatula, placed in front of the robot with a fixed pose. We applied blended intrinsic maps, see Section 3.3.1, to map the contact coordinate

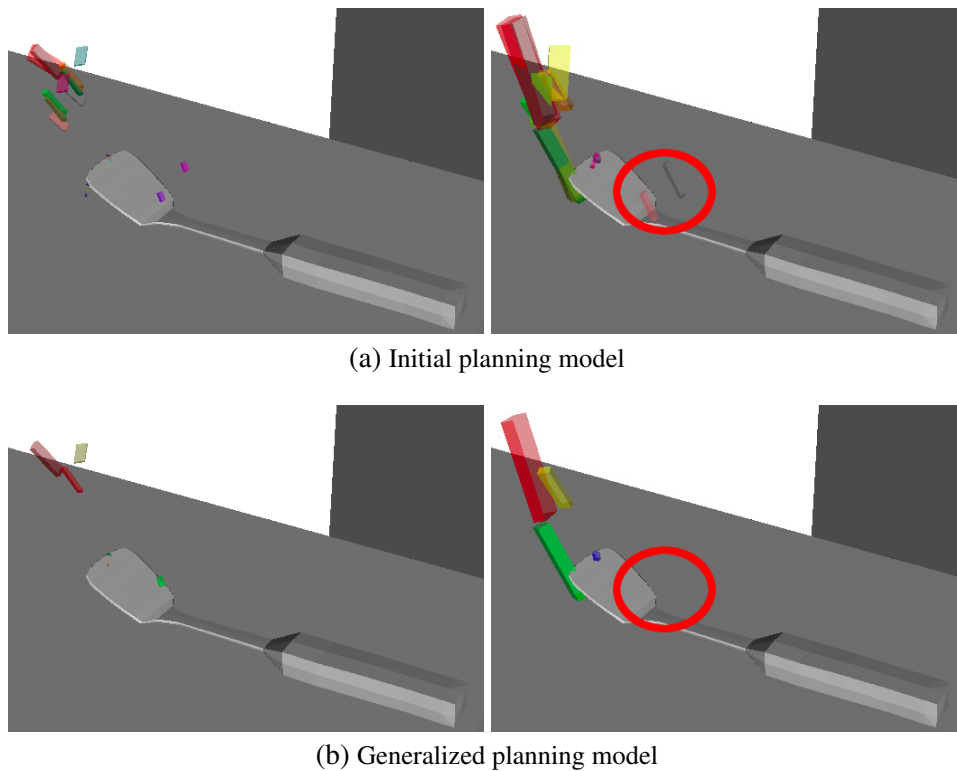


Figure 5.45.: Lifting a spoon: node (left) and arc (right) positions constraints. The circle highlights two inconsistent constraints, which are relative to the center of mass of the spoon (a). Both constraints are removed in the generalization algorithm (b). Position constraints, which are relative to the generated contact coordinate frames remain. - [53]

frames to the 3D model of the spatula, see Figure 5.44. The generalized planning model contained 55 constraints.

Figure 5.45 shows the set of learned and generalized constraints. A special set of constraints in the initial planning model is highlighted. It restricts the motion of the center of mass of the spoon, when it is lifted upwards, i.e. the robot applies force to it. Since the spatula is larger than the original spoon, no motion of the spoon can be generated, in which the set of constraints is obeyed. Subsequently, this set of constraints is removed in the generalization algorithm. Important constraints remained, e.g. the spoon has to have a contact with the table, the fingertip has to make contact with the spoon and the coordinate frame in the contact point of the fingertip has to be slightly rotated.

Figure 5.46 shows the validation of a planned trajectory in physics simulation. The planning process is identical to the previous experiment. Planning of the generalized planning model took 11.2s on average with a success rate of 85%. The overall planning problem was less difficult compared to the previous experiment. During contact a robot finger (3 DOF) and the robot arm (7 DOF) were moved resulting in a 10 dimensional configuration space but with a large workspace. In the previous experiment, the robot arm was static while two fingers moved during contact re-

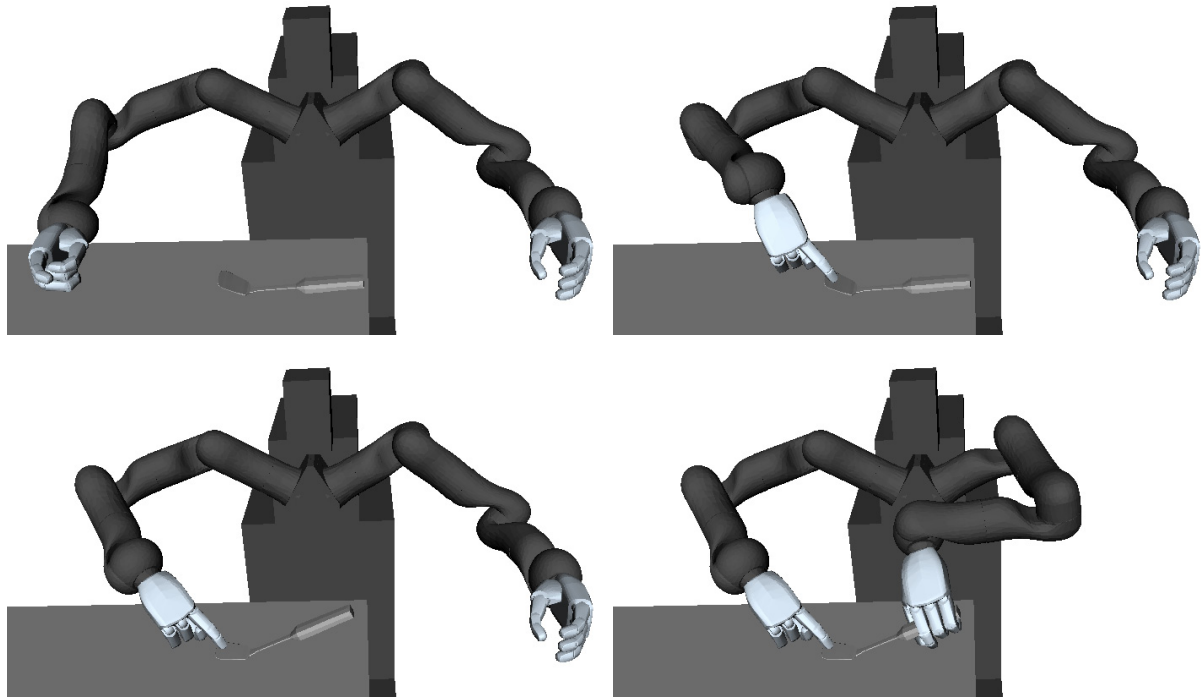


Figure 5.46.: Lifting a spoon: validation of planning result in physics simulation. - [53]

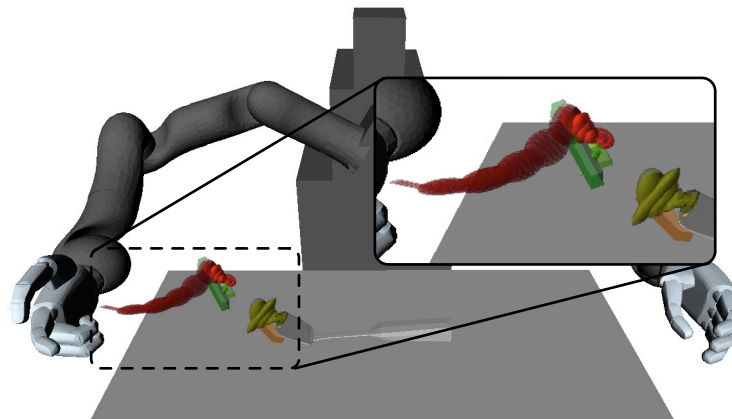


Figure 5.47.: Lifting a spoon: visualization of search heuristic 1.

sulting in a 6 dimensional configuration space. Since the finger motions had to be synchronous and the workspaces were highly restricted, the constraint manifold was more complex.

We learned search heuristic constraints to reduce planning time and consider the correspondence problem. Figure 5.47 shows an example of a learned search heuristic. Planning with search heuristics resulted in an average planning time of 6.54s with success rate 85% corresponding to a speed-up of 1.71.

Figure 5.48 shows the execution of the task on Adero. The handle of the spoon was enlarged to detect it using the vision system and to grasp it securely.



Figure 5.48.: Lifting a spoon: execution on Adero. The handle of the spatula was enlarged to be able to detect the spoon visually and grasp it securely. - [53]

The results indicate that simple dexterous manipulation tasks can be learned and planned efficiently using the developed PbD process. Learning of search heuristics played an important role to counteract the negative effects of relaxing finger and hand constraints, i.e. the increase in planning time, to consider the correspondence problem.

5.5.3. Conclusion

The first experiment showed that manual definition of a planning model for dexterous manipulation tasks is time-demanding, even for an expert user. The constraints had to be relaxed by the user since it is difficult to find hand poses, in which the rotation motion of the fingers does not violate the workspace boundaries. The result is a high planning time.

Task	Number of Search Heuristics	Volume Reduction to (%)
Force closure grasp of cap	23	35.73
Pressing a key	2	3.37
Large bottle in fridge	18	1.99
Small bottle in fridge (low grasp)	11	2.69
Small bottle in fridge (high grasp)	43	3.96

Table 5.19.: Overview of constraint specialization results. The volume reduction is the quotient (in %) of the volume of the specialized constraint region and the volume of the original constraint region.

We learned search heuristics to decrease planning time. By projecting the search heuristic constraints on the manually defined constraints, the constraints were successfully tightened. The planning results show that the tightening decreased planning time significantly but did not affect the generalization capabilities of the planning algorithm, i.e. both processes converged to 100% planning success with sufficient time thresholds.

The results were confirmed by the second experiment. In this case, the planning model was learned from human demonstrations and generalized with the robot-test-based approach. Due to the large kinematic differences between the human hand and the SAH, constraints for the wrist and point finger were relaxed resulting in an average planning time of 11.2s (without smoothing). By learning search heuristics the average planning time dropped to 6.54s. Similar to the first experiment, the success rates converged to similar high values (85%, 82%).

5.6. Evaluation of Constraint Specialization to Consider the Correspondence Problem

In this experiment, we evaluate the projection of generated search heuristics to learned constraints. The constraints are initially relaxed to consider the correspondence problem, i.e. the differences between human and robot morphology. The goal is to measure the reduction in volume of the constraint regions, which can be achieved. We used Monte Carlo Integration to measure the volume reduction with 10^6 samples. Five different tasks were evaluated: Force closure grasp of a cap, pressing a key, placing a large bottle in the fridge, placing a small bottle in the fridge with a low grasp and placing a small bottle in the fridge with a high grasp. We ran each task 1500 times to generate search heuristics and evaluated the volume reduction of a single node position constraint in each task. Table 5.19 summarizes the results.

Grasping a Bottle Cap

Figure 5.49 shows the first task. The robot hand had to be placed above a bottle cap in a way that a grasp quality constraint is obeyed. Since the bottle cap is small and the workspace of the fingers is restricted, the initially large position constraint for the wrist is tightened. We placed the bottle at a random pose in front of the robot above the table. The sampling distribution was uniform on

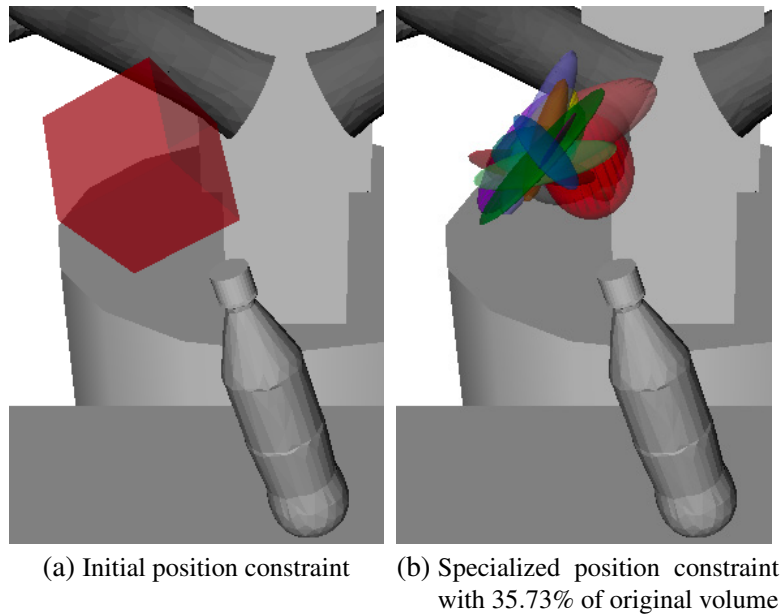


Figure 5.49.: Constraint specialization: grasping a bottle cap

the cube with center $(900, 0, 900, 0, 0, 0)$ and sizes $(400, 400, 100, 90, 90, 90)$, with the convention $(x, y, z, roll, pitch, yaw)$. The reduction in volume, i.e. the quotient of new and old volume, is 35.73% with a total of 23 search heuristics. The number of search heuristics is high compared to Section 5.4 since a large number of bottle poses were generated randomly, which do not occur when the robot lifts the bottle to open it.

Pressing a Key

In the second task, a finger had to be placed on a key. We evaluated the reduction of the position constraint, which restricts the wrist pose. A contact constraint ensured that the point finger has only contact with a single key, see Figure 5.50. Since the position constraint for the wrist is not very restrictive, the problem to find a robot configuration, in which the finger can be placed on the key, was difficult. The keyboard was placed randomly in a cube with center $(960, -65, 724, 0, 0, 90)$ and sizes $(400, 400, 0, 0, 0, 90)$. The reduction in volume was 3.37% with only 2 search heuristics learned showing that it was difficult to place the robot finger, which is thicker than the human one and has a more restricted workspace, on the key without colliding with other keys.

Placing a Bottle in the Fridge

In the last three experiments, we evaluated the result of the constraint specialization using a single task with multiple objects. Figure 5.51a shows a position constraint, which restricts how a bottle has to be placed in the shelves of a fridge door. The constraint region of the position constraint does not represent the geometry well but a contact constraint ensures that the bottle bottom has to be in contact with a shelf. In all experiments, we placed four cylinders (with height $235mm$ and

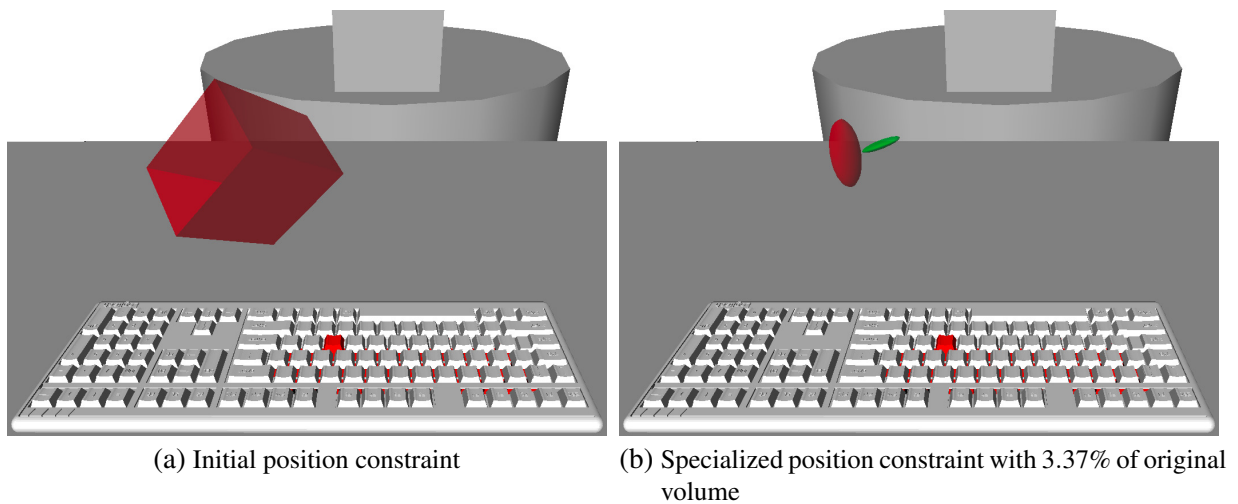


Figure 5.50.: Constraint specialization: pressing a key

radius 36mm) at random positions in the fridge. Two of the cylinders are placed in the lower shelf (represented by a cube with center $(0,0,115,0,0,0)$ and sizes $(6,300,8,0,0,0)$). The remaining cylinders are placed in the upper shelf (represented by a cube with center $(0,0,455,0,0,0)$ and sizes $(6,300,8,0,0,0)$).

In the first trial, we used a large bottle, which does not fit in the top shelf. Therefore, the learned 18 search heuristics covered only the lower shelf. The volume was reduced to 1.99%.

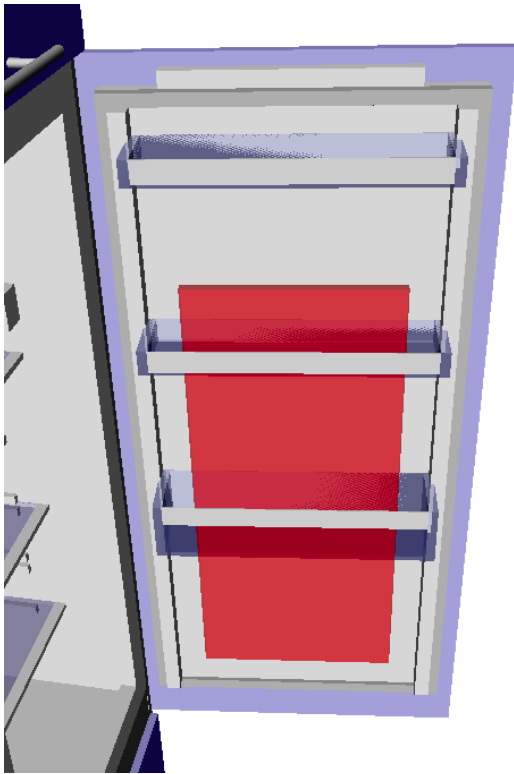
In the second trial, a small bottle was grasped from the side. If the bottle was in contact with the lower shelf, the hand collided with the shelf. The result is that the bottle could only be placed in the top shelf. 11 search heuristics were learned and the volume was reduced to 2.69%.

In the last trial, the small bottle was grasped higher and could be placed in both shelves. Consequently, the 43 learned search heuristics cover both shelves. The volume was reduced to 3.96%.

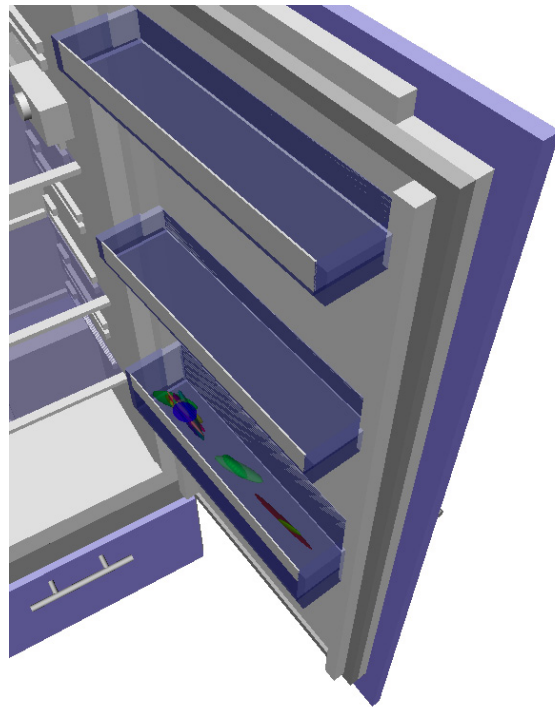
Conclusion

In the first and second task, the position of the robot wrist was constrained to execute a difficult grasp or move a finger to a specific point. The high reduction in the volume of the constraint regions indicates that the original constraint region was suboptimal. This is always the case if constraints are relaxed to consider the correspondence problem. With the specialized constraint, the tasks can be executed more efficiently. In the second task, the variance of the human could not be reproduced by the robot supporting the assumption that differences in morphology have to be considered explicitly when learning manipulation tasks from the human.

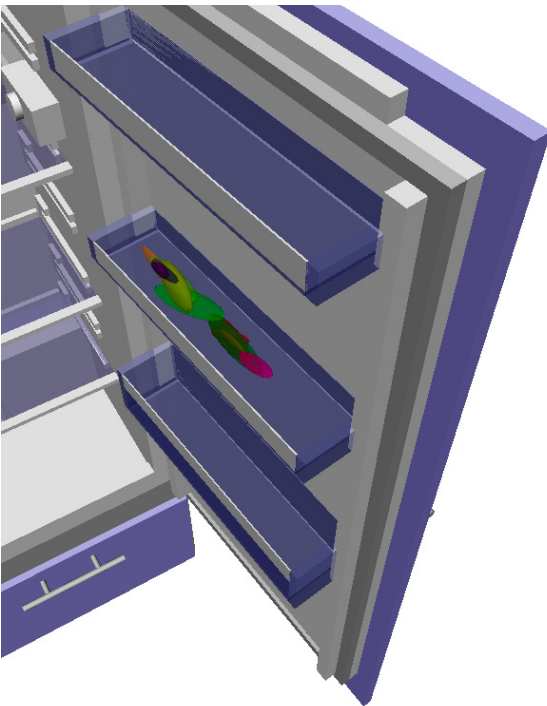
The last three tasks show that learned constraints may be a rough estimate of an optimal solution, e.g. due to learning with different objects. In this case, constraint specialization enables the robot to specialize a learned constraint to the geometry of the target object. The result is a significant reduction in planning time.



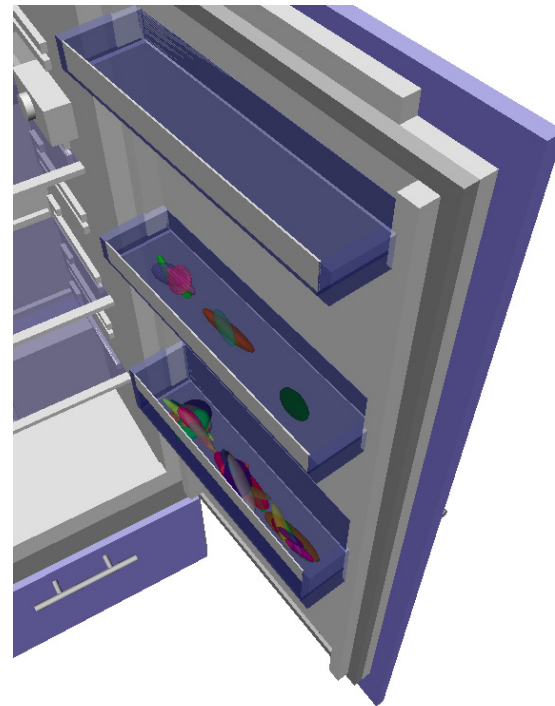
(a) Initial position constraint



(b) Specialized position constraint with 1.99% of original volume: large bottle, which does not fit in the top shelf.



(c) Specialized position constraint with 2.69% of original volume: small bottle, low grasp, hand collides with lower shelf.



(d) Specialized position constraint with 3.96% of original volume: small bottle, high grasp. Bottle fits in both shelves.

Figure 5.51.: Constraint specialization: placing a bottle in the fridge

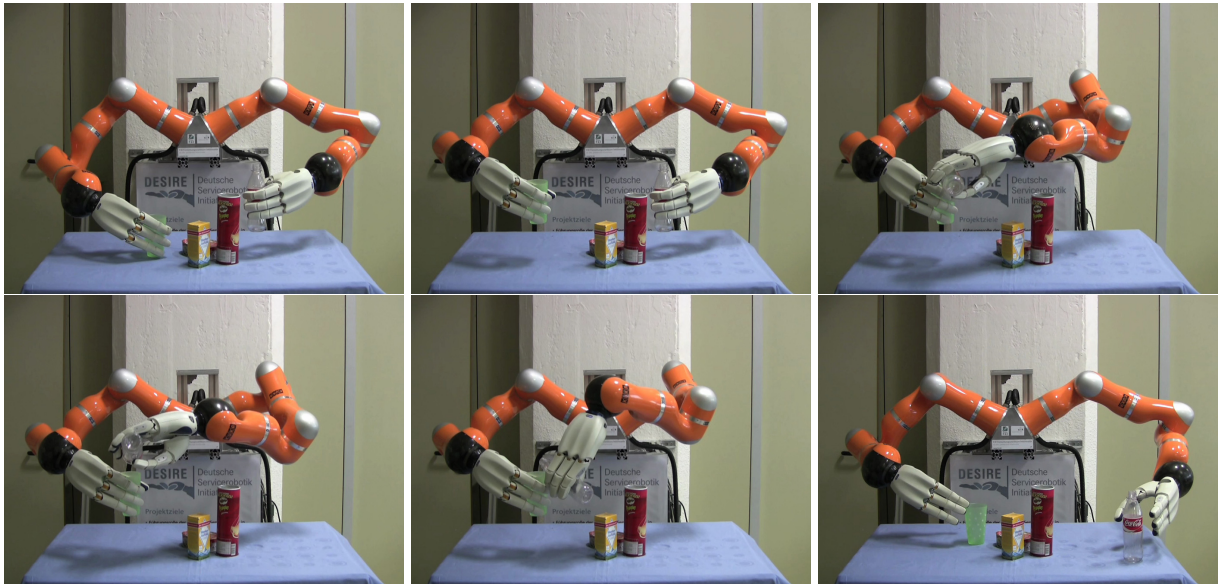


Figure 5.52.: Execution of pour-in task on Adero (version 1) with small bottle, large cup and multiple obstacles. - [51]

5.7. Evaluation of Scalability to Different Robot Systems

In this section, we evaluate the mapping of learned planning models to different robots in a qualitative way. The planning algorithm was tested with the real robots Adero, Albert II and Justin (DLR) and in simulation with Armar IIIb and PR2 (Willow Garage). All three versions of Adero (I, II, III) were used. All robots except Albert II have two arms.

The pour-in task was planned and executed. It consists of grasping a bottle and cup with each hand, pour-in and place the objects on the table. For Albert II and Adero version II and III only a subgraph of the learned planning model was used, i.e. only the bottle was grasped and the cup remained on the table. Planning and execution of the task were successful on all robots. The execution is shown in Figure 5.52 to 5.58.

This qualitative result shows that the abstraction of low-level human trajectories to coordinate frame-based constraints is sufficient to reproduce the task on different robots.

The generated robot trajectories respect the geometry, e.g. self-collisions, and kinematics of each individual robot. Since the learned constraints are robot-independent (before specialization), constrained motion planning is sufficient to reproduce the goals and constraints of the task while considering robot-dependent constraints, e.g. joint limits and self-collisions.

Additionally, different environments were used with no obstacles, see Figure 5.58, small obstacles, see Figure 5.54, a large obstacle in front of the robot, see Figure 5.56 or in a restricted workspace, see Figure 5.57. The latter shows the flexibility of a single learned planning model to large differences in environments.

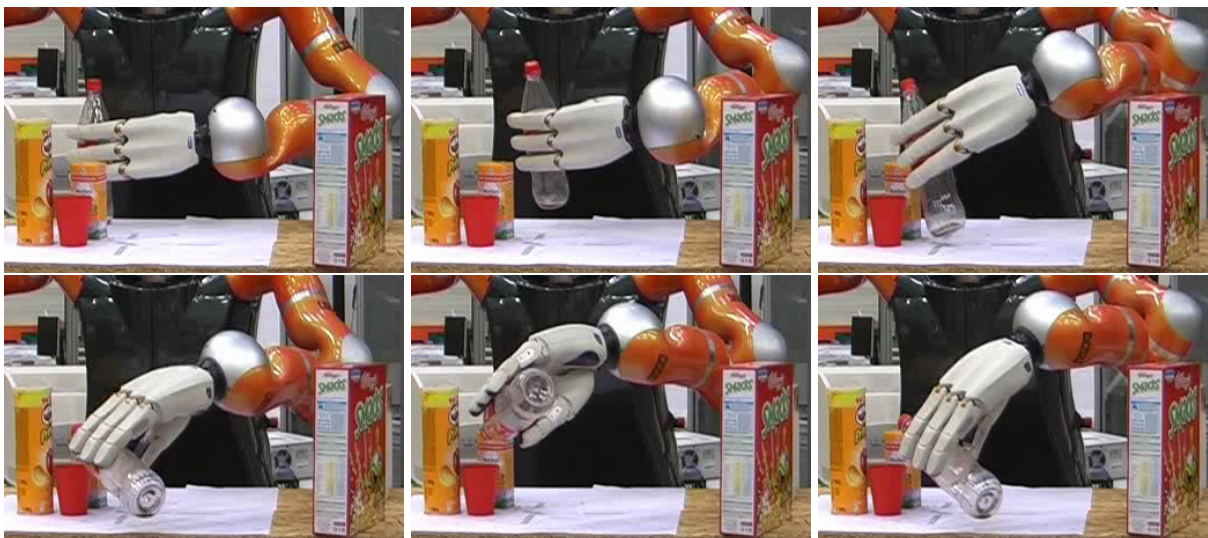


Figure 5.53.: Execution of pour-in task on Adero (version 2) with large bottle, small cup and multiple obstacles.

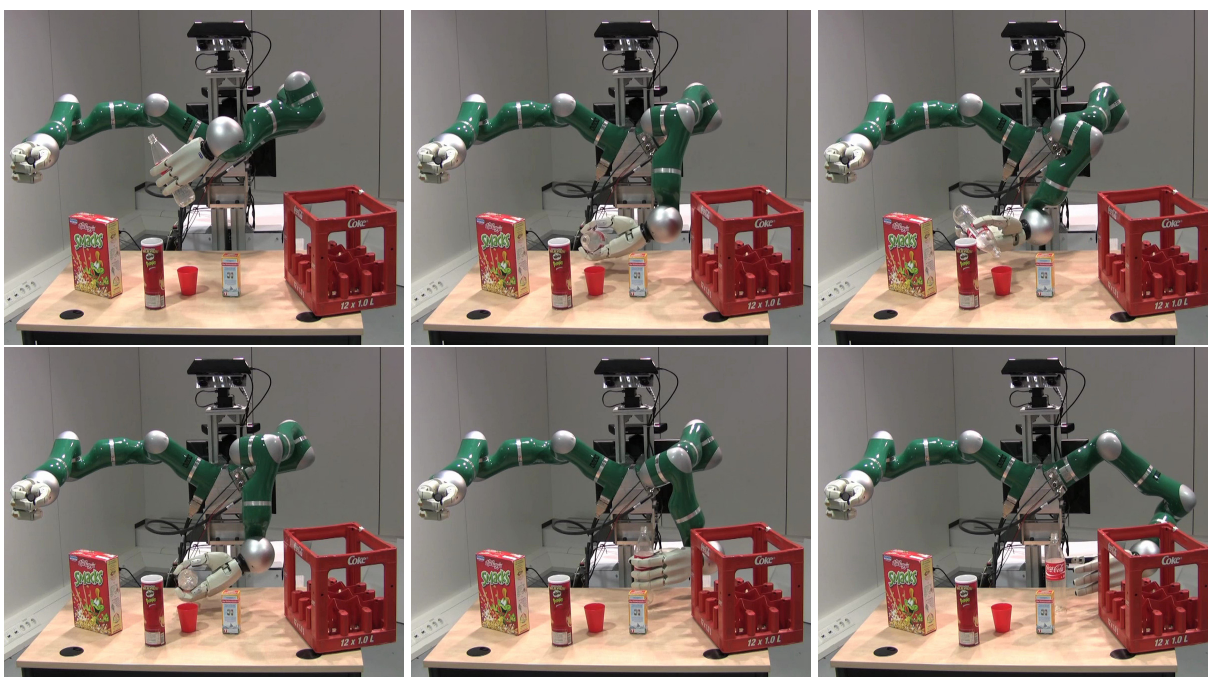


Figure 5.54.: Execution of pour-in task on Adero (version 3) with large bottle, small cup and multiple obstacles. - [50]

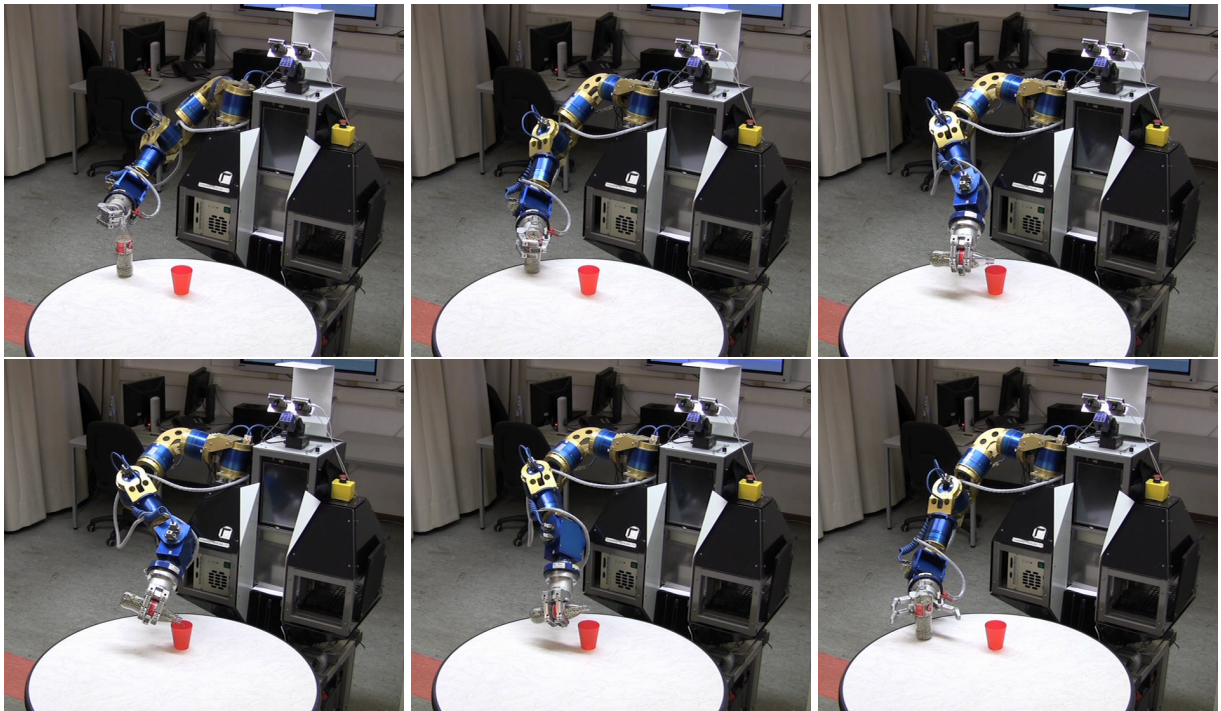


Figure 5.55.: Execution of pour-in task on Albert II with small bottle, small cup using one arm. Perls were used to simulate the fluid inside the bottle. No perls were spilled.

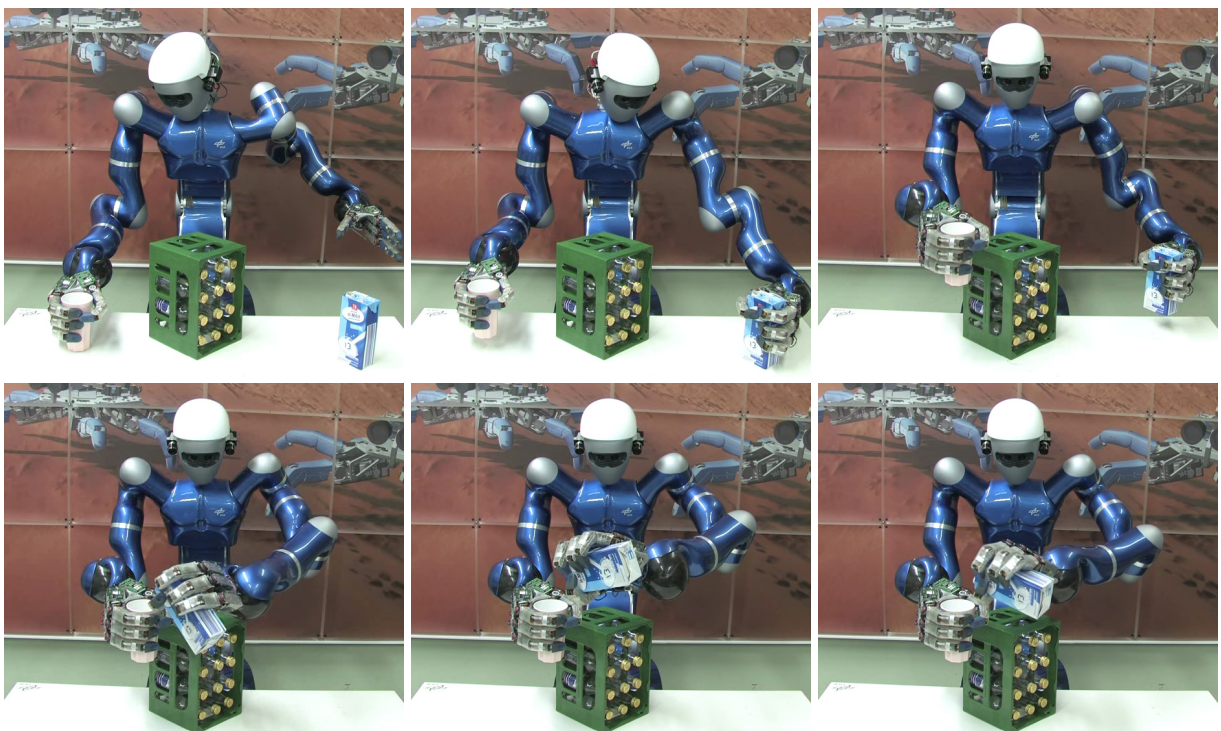


Figure 5.56.: Execution of pour-in task on Justin (DLR) with white cup, milk box and a large obstacle. The grasps of the white cup and milk box were planned and executed by DLR (top left, top center).

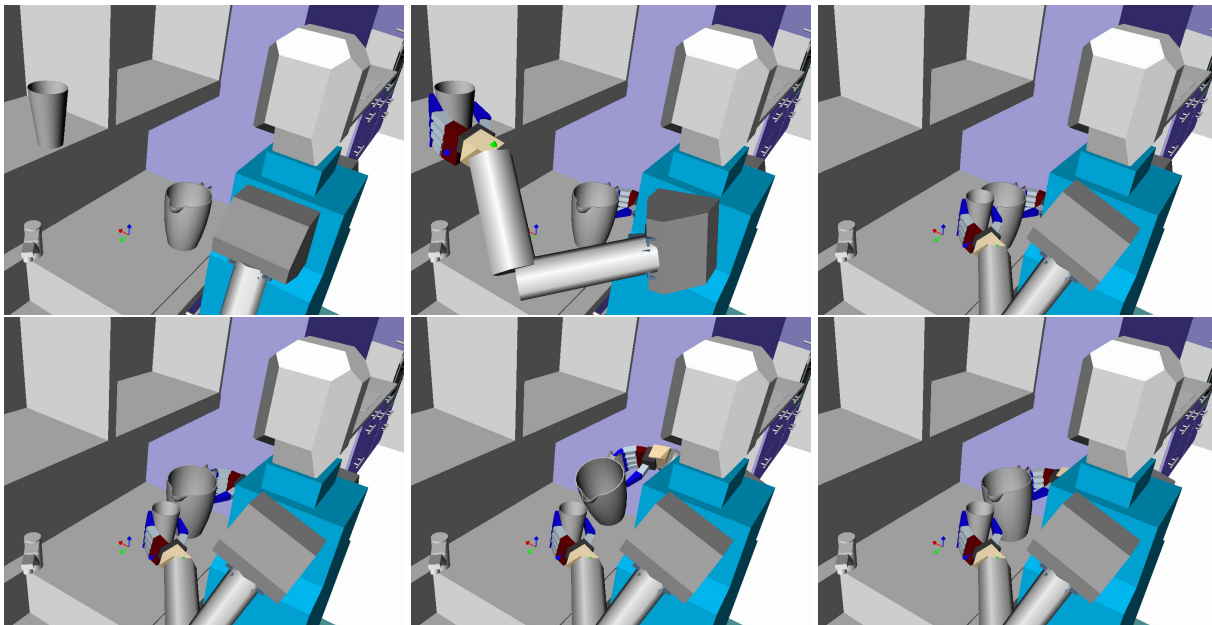


Figure 5.57.: Execution of pour-in task on Armar IIIb with large cup, measuring cup and cupboard using simulation.

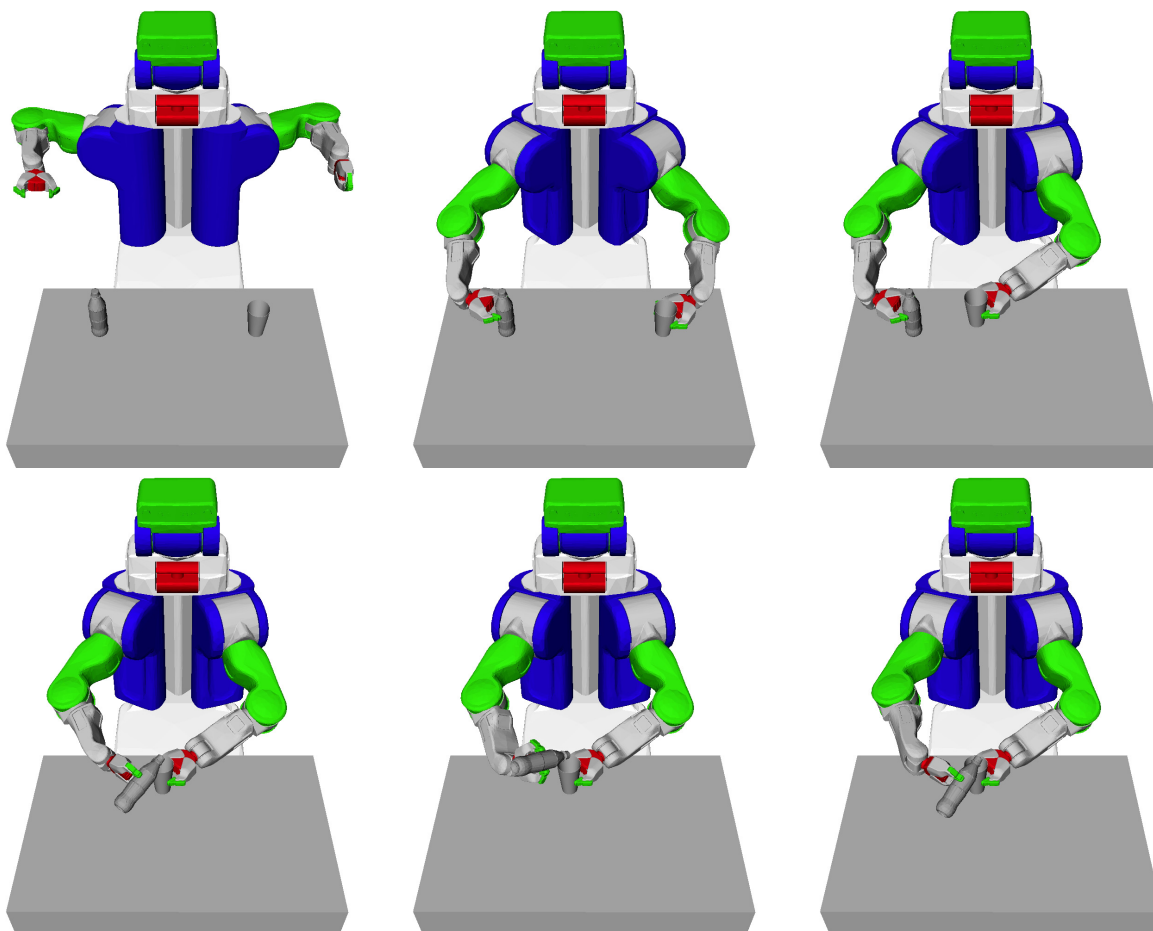


Figure 5.58.: Execution of pour-in task on PR2 with small cup and bottle using simulation.

5.8. Conclusion

We evaluated the complete PbD system as well as individual components with different manipulation tasks, ranging from simple tasks like grasping to more involved tasks like opening a bottle with the fingertips. Applicability to real-world problems was shown on the real robots Albert II, Adero and Justin. We used simulation to provide a rich database to analyze generalization, specialization and planning of learned manipulation strategies. Scalability to different robots was shown using two more robots in simulation.

The experiments show that manipulation tasks can be learned efficiently based on a limited set of human demonstrations. The analysis of the dependency on the experience of the teacher requires extensive user studies and optimized user interfaces, which was out of the scope of this thesis.

In literature, fully-integrated learning and execution systems, which scale to different manipulation tasks and require limited user interaction, are sparse. One reason is the tremendous amount of work required to set up real robot and sensor systems. Recent developments about standard robot operation systems, e.g. ROS [85], and availability of fully-integrated service robotics with enhanced manipulative and perceptive capabilities might lead to better comparability of results.

In the experiments, available controllers were used to execute the planned robot trajectories. In general, goals and constraints can be used directly on the control level to monitor the execution, e.g. using visual servoing, to increase robustness. Development and integration of advanced control algorithms, which allow to monitor sets of constraints, was not in the scope of this thesis.

6. Summary

The automation of everyday manipulation tasks with an anthropomorphic service robot poses major challenges for robotics research. In this thesis, we concentrated on manipulation challenges. In the human environment, a huge variety of tasks and objects exists and grows over time. Additionally, the maneuverability of the robot is restricted due to collisions with the environment or task-constraints, e.g. to hold a cup upright. The consequence is that the robot needs to be flexible. It has to adapt its manipulation knowledge to new situations and learn new manipulation knowledge in an efficient way. The basis is an abstract model of the manipulation task.

The first challenge is to find a representation of manipulation tasks, which captures all quantitative and qualitative aspects of the task that are necessary to generate a successful robot motion. We discussed state-of-the-art representations of manipulation tasks. Constraint-based representations allow to restrict different aspects of the task in a flexible way. The main limitation is the small number of constraints and lacking structure information, e.g. that a task might consist of multiple subgoals with different constraints.

We overcame this limitation by defining an extensive set of constraints to model different aspects of a manipulation task: position, orientation, direction, force, moment, contact, configuration, temporal and additional constraints like collision avoidance or minimal grasp quality. For example, a position constraint restricts the motion of one coordinate frame, e.g. in the fingertip, relative to a second coordinate frame, e.g. a button. In general, a manipulation task does not consist of a fixed set of constraints but multiple goals and constraints, which change over time. Based on this observation, we introduced *strategy graphs* to model manipulation tasks, in which nodes define (sub-)goals of the task and arcs define the constraints between two (sub-)goals. In principle, (sub-)goals can be reached in parallel. The set of temporal constraints defines a partial order, in which (sub-)goals have to be reached. State-of-the-art constraint-based motion planners assume a fixed set of constraints to plan efficiently. Since the set of constraints depends on the ordering of nodes, this assumption is violated. We interpreted the strategy graph as a Constraint Satisfaction Problem (CSP) and integrated standard tools from CSP-theory into the state-of-the-art motion planner RRT to solve the CSP. The complexity is very high, which led to an important simplification: we considered only linear planning models consisting of a sequence of nodes and arcs in the remainder of the thesis.

Even if the set of constraints is known, planning with a large number of diverse constraints is currently unsolved. The set of constraints defines a manifold of configurations, in which all constraints are obeyed. The manifold of collision-free configurations, the *free space*, highlights important problems: sampling a configuration from the manifold or calculating the distance to

the manifold may be intractable. We integrated projection techniques from Computer Vision into the state-of-the-art constraint-based motion planner CBiRRT to plan efficiently on the constraint manifolds defined by a strategy graph. The main advantage of motion planning is flexibility. The manipulation task can be executed in environments with different start configurations, varying object poses and obstacles.

Even simple tasks like opening a bottle require coordinated motion of multiple fingertips and the arm to induce a desired object motion. The result is a huge number of constraints. Manual definition is time-consuming, error-prone and requires expert knowledge. The second challenge is to acquire models of manipulation tasks in a flexible and efficient way. Related work in Programming by Demonstration (PbD) allows to learn manipulation knowledge based on a set of human demonstrations and a predefined feature space. The representations in the related work are suitable to reproduce the task in environments with minor deviations, e.g. using a simple controller. Flexibility to environments with high variance in objects and obstacles is limited. Additionally, more elaborate ways to remove irrelevant constraints, which reduce flexibility, had to be found.

We described a novel PbD-approach to learn strategy graphs efficiently based on a small number of human demonstrations. Important problems have been solved: occlusion-free observation of the human during manipulation, learning a preliminary planning model, identification and removal of irrelevant constraints to increase flexibility and refinement of constraints to consider the differences in morphology between human and robot.

The observation of the human during manipulation is problematic due to occlusions of fingertips or the manipulated objects by the human himself or the environment. Based on prior work in PbD we employed datagloves and magnetic field trackers to observe human hand and finger motion without occlusions. Contacts and forces applied in contact points represent relevant aspects to model a manipulation task, which cannot be measured with sufficient precision. We generated an accurate hand model of the human teacher using a Minolta VI-900 laser scanner to calculate contact points in 3D-simulation. Additionally, tactile sensors were employed to estimate forces in contact points.

In order to learn a (preliminary) planning model based on multiple demonstrations two problems have been solved: generating the structure, i.e. nodes and arcs, and generating constraints for each node and arc. The first problem has been solved using velocity-based segmentation, which allows the teacher to demonstrate goals explicitly by making slight pauses in the demonstration. The advantage is that semantically important time points can be identified, leading to less goals and shorter planning times. The second problem refers to one of the fundamental problems of pattern recognition: how to identify the best set of features to solve a classification problem. In our case, constraints are the features, from which to choose. In related work, mathematical tools like Principal Component Analysis (PCA) are applied to reduce dimensionality of the feature space. Inhomogeneous data, e.g. forces, orientations and positions, as well as the resulting mixture of different features is problematic. We followed a different approach using ideas from Curriculum Learning as well as optimization theory. The starting point was a predefined, broad set of con-

straints, which can be instantiated in different tasks. We defined two sets of constraints. One for Pick-Transport-Place tasks, which do not require to simulate force interaction, and the second for dexterous manipulation tasks, which require to capture force interaction with the fingertips. The constraint sets represent patterns, which allow to generate a large number of potentially relevant constraints automatically. The result is a preliminary planning model with a huge number, usually more than one hundred, of constraints.

In order to execute the planning model on an anthropomorphic robot system, the differences in morphology between the human and robot, the so called *correspondence problem*, have to be considered. In literature, different approaches exist, e.g. correspondence matrices represent a (weighted) mapping of joint values between the robot and the human. In the context of manipulation, the mapping of wrist and finger poses is problematic. Joint-based mappings are often inefficient due to different lengths and placement of fingers, even with anthropomorphic robot hands. A fixed cartesian mapping of wrist and finger poses is better suited to reproduce contacts in the fingertips but does not consider workspace boundaries of the hand. We followed a different approach and relaxed constraints, which restrict finger and hand motion, automatically based on the differences in workspace and geometry. The remaining constraints, e.g. which restrict the object motion, were not relaxed. In the planning process, aspects of the task, which are assumed independent of robot morphology, e.g. object motion, have to be reproduced exactly but, based on the relaxed constraints, the observed hand and finger motion can be adapted to do so. Nevertheless, the relaxed constraints represent an important restriction of the search space, which is necessary to solve high-dimensional planning problems.

Flexibility of the preliminary planning model is limited due to the inclusion of irrelevant constraints. We introduced two complementary approaches to identify irrelevant constraints and increase flexibility of the planning model. In the demonstration-based generalization, the human teacher sorts demonstrations into two sets according to complexity. Following the idea of Curriculum Learning, the preliminary planning model is trained with the first set and constraints, which are inconsistent with the second set, are removed. Following this approach, irrelevant constraints can be removed efficiently but the teacher requires experience. In the robot-test-based generalization, the teacher defines a set of robot-tests, which have to be solved by the robot system. We defined an optimization problem with the goal to find a maximum subset of constraints, which admits a successful solution to all robot-tests. The optimization problem is solved using a parallelized evolutionary algorithm. Hypotheses about irrelevant constraints are automatically modified based on statistics about inconsistencies of constraints, which are generated while planning and simulating the manipulation task. The advantage of using simulation-based optimization to identify irrelevant constraints, e.g. instead of PCA, is that even complex constraint interactions are considered, for instance we can identify an irrelevant position constraint, which prevents that a contact constraint is obeyed. Since the teacher does not have to demonstrate solutions to the manipulation tasks but only situations, in which the robot has to be able to solve it, user-interaction is reduced to a minimum and less experience is required.

Our goal is to execute the learned manipulation task with the robot in different environments. Due to the high number of controllable degrees of freedom of the robot, finding a solution to a given manipulation task might be time-consuming. Additionally, we expect a robot to learn from its experience and become better with time. The third challenge is to find execution mechanisms, which allow the robot to become faster over time and solve everyday manipulation tasks in a short amount of time. In the state-of-the-art, we described how optimal controllers can be derived to execute simple manipulation tasks. The approach does not scale to complex environments and manipulation tasks with complex constraints. Recent results in constraint-based motion planning allow to plan robot motions on complex constraint manifolds. But depending on the environment, planning time can be very high. We discussed different search heuristics to speed up the planning process. A common limitation is that search heuristics are fixed or learned prior to the execution. Incremental learning, e.g. to learn new search heuristics for situations, in which all other search heuristics fail, is not supported. Due to the large variety of objects and obstacles in the human environment, this kind of adaptivity is mandatory.

We extended the constraint-based motion planner to learn search heuristics incrementally. Each search heuristic encodes a cluster of robot trajectories into a sequence of ellipsoids based on Gaussian Mixture Regression. In the vicinity of an object, a robot trajectory often depends on the geometry of the object, e.g. when pushing a slider with the finger tip. Farther away from the object, the object geometry becomes less important and the influence of the start configuration and obstacles increase. Based on this assumption, we modified our planning algorithm to generate a robot trajectory in two parts: the object-dependent and -independent part. The first one is generated using a fast, local controller with the learned search heuristic. The advantage is short planning time but only local obstacles can be considered. The object-independent part represents the connection of the start configuration and the object-dependent trajectory. We generate the object-independent part using the constraint-based motion planner to plan the original strategy graph, which allows to efficiently consider different start configurations and arbitrary workspace restrictions, e.g. complex obstacles. The developed planning algorithm combines two major approaches in Programming by Demonstration in a flexible way: efficient encoding and reproduction of robot trajectories and flexible, goal-directed motion planning. In the experiments, we measured a significant reduction in planning time for everyday manipulation tasks.

Search heuristics represent a flexible restriction of the search space for motion planning. By projecting the search heuristic to the constraint manifold we effectively tightened the set of learned constraints. Since search heuristics are generated automatically in the planning process, which uses the geometric, kinematic and (potentially) the dynamic model of the robot, the robot morphology is considered explicitly. Looking back on the whole process, we see how the correspondence problem was considered. First, hand and finger constraints are learned, which incorporate human morphology. Second, constraints are relaxed based on the differences in geometry and kinematics. Finally, task solutions are generated on the robot and constraints are tightened, which incorporates

robot morphology. The learning result is a sophisticated task model specialized to the robot leading to high flexibility and efficiency in the execution process.

In the next section, we discuss limitations as well as open research questions. In Section 6.2, we conclude the thesis with the contribution and potential impact of the described work.

6.1. Limitations and Outlook

Manipulation tasks cover different important aspects, e.g. forces applied through fingertips or object motion. The observation is difficult due to occlusions or lacking sensor systems, e.g. forces between objects. In this thesis, we used a dedicated sensor environment with non-intrusive, occlusion-free sensors like data gloves and magnetic field trackers to observe the human during manipulation. Missing information, e.g. forces, was generated in dynamics simulation. In order to learn manipulation tasks directly using the robot sensors, different sensor systems have to be used providing less accurate information. In the context of the correspondence problem, we showed a way how to cope with inaccurate information about finger and hand motion. Inaccurate information is reflected by relaxed constraints, which are tightened using planning and simulation results. In combination with advanced sensor technologies, this approach could be the basis to learn manipulation tasks using onboard robot sensors.

The developed representation of manipulation tasks covers a large number of different task aspects represented by the strategy graph. Each constraint is represented by a geometric region. We investigated different types, e.g. hypercubes or ellipsoids, but limited our work to unimodal representations with a single connected component. The learning process is therefore limited to a continuous set of alternatives. Distinctive sets are considered by learning alternative strategy graphs. Since generalization and specialization are time-demanding processes, learning of distinctive alternatives in a single strategy graph has to be researched. We derived characteristics of geometric regions, which generalize also to multimodal representations. The difficulty is that either more training data is required or less flexibility can be achieved.

In the beginning, we assumed that models of the robot, objects and environment are available. In the human environment, this assumption is often violated, e.g. models of furniture and clothes are hard to obtain, and 3D models have to be generated efficiently. Kasper et al. [59] showed that 3D models of rigid objects can be generated using a high precision laser scanner. For most products, 3D models exist already due to the use of CAD tools. In order to use an object in a dynamics simulation, the physical properties of the object have to be retrieved. Sugaiwa et al. [103] demonstrate first results how a robot can autonomously figure out weight, friction and stiffness of objects. In future, we expect a robot to generate a 3D model of a given object using the robot sensors and to determine the physical properties by interacting with the object, which is the basis to learn manipulation tasks with previously unknown objects.

State-of-the-art dynamics simulation is either too inaccurate or too slow to be incorporated into a planning system, which limits applicability to simple manipulation tasks. In order to plan more

complex manipulation tasks, e.g. flipping a coin, future advances in this field are required. Additionally, complex manipulation tasks, e.g. tying a knot, require (fast) adaptation of the planned robot trajectory to react to changes in the environment and increase robustness. In order to react to forces or perturbations, a control algorithm with high frequency has to be applied. Different control strategies, e.g. position or impedance control, exist, which have to be chosen by the user right now. In general, the strategy graph holds necessary information, e.g. the allowed range of joint values, forces, position and orientations of specific coordinate systems. How to use this information automatically to set up an appropriate control strategy is an open research question.

The representation of robots, objects and tasks is only an approximation of the real world. Since planning relies on these models, deviations during the execution will occur and might lead to execution failure. The idea of supervised autonomy is that the robot operates autonomously but is supervised by a human operator, who can react to failures and choose appropriate actions. If a failure occurs, a common idea is to adapt the representation in an intuitive way to reenact autonomous execution in similar situations. In this thesis, results of robot-test-based generalization as well as learning of search heuristics indicate that the developed constraint-based representation can be adapted efficiently, which shows the applicability to supervised autonomy. Learning new constraints or adapting existing constraints using only a small number of demonstrations is the next step to flexible and robust manipulation.

6.2. Conclusion

Flexible manipulation planning is one of the key ingredients to automate everyday manipulation tasks in human-centered environments, where the robot faces a large variety of tasks, objects and obstacles. Manual definition of such planning models is time-consuming, error-prone and requires expert knowledge. Machine learning offers a way to include the human, who is an expert in manipulation, in a natural way into the programming process. In this thesis, we applied machine learning to learn a sophisticated description of manipulation tasks based on a set of human demonstrations. By using recent results in constraint-based motion planning, tasks were executed in a flexible way on different anthropomorphic robots in variable environments with different objects and obstacles. Based on this foundation the following contributions were made:

1. Manipulation task representation on the basis of goals and constraints to support flexible online motion planning.
2. (Semi-)automatic extraction of relevant constraints and goals of a manipulation task, i.e. the learning features, using an optimization process, which limits user interaction and leads to higher flexibility in the planning process.
3. Incremental learning of search heuristics based on prior planning results to restrict the search space for future planning attempts leading to shorter planning times in real world applications.

4. Consideration of the correspondence problem by relaxation of constraints, which refer to human morphology, and automatic tightening of constraints based on planning results, which incorporate robot morphology.

Other key ingredients to automate everyday manipulation tasks are robust execution in real world applications, accurate and robust perception of the real world and high-level decision making to enable real autonomy.

Robust execution requires further advances in robot technology to be able to resist external perturbations, e.g. impacts, and allow to rethink the paradigm of collision-free manipulation. The hand-arm-system (HASY) by DLR [41] shows significant progress in this direction.

Perception is a major topic in robotics research and fundamental to bridge the gap between industrial or laboratory environments and real-world applications. Recent works profit from the availability of low-cost but robust RGB-D cameras and future progress is to be expected.

The last ingredient is high-level decision making, which is required to solve complex tasks in the human environment. Tasks like refilling the fridge require reasoning on different levels of abstraction. The integration of these high-level decision making systems with low-level manipulation planning and execution is an open but intensively investigated research question.

Progress in these areas is promising and with the availability of fully-integrated service robots the automation of everyday manipulation tasks is on the verge of becoming reality.

A. Glossary and Notation

Term	Description	Reference
Adero	Anthropomorphic robot	Section 5.1.1
Albert II	Service robot	Section 5.1.1
Anthropomorphic robot	Service robot with two arms and multi-fingered robot hands with opposing thumbs	Section 1.1
Arc constraints	Constraints restricting the motion from one subgoal to the next of a manipulation task	Section 3.2
Armar IIIb	Humanoid robot	Section 5.7
Assignment	Specification of a time point for each subgoal in a planning model corresponding to a variable assignment in the associated CSP	Section 3.2.2
Cartesian trajectories	Trajectory defined in the Cartesian space, e.g. of the Tool-Center-Point	Section 4.6.5
CBiRRT	Constrained Bidirectional Rapidly-Exploring Random Tree	Section 3.4.3
Configuration	Representation of all relevant manipulation aspects, which are constrained in a planning model	Section 3.1.1
Consistency	Existence of an assignment in which all temporal and domain constraints are obeyed	Section 3.2.2
Constraint	Restriction of the search space for motion planning	Section 3.1
Constraint manifold	Set of configurations, in which a set of constraints are obeyed	Section 3.1.1
Constraint-based motion planning	Goal-directed planning of a robot trajectory in the presence of constraints	Section 2.3
Constraint relaxation	Automatic enlargement of constraint regions to consider the correspondence problem	Section 4.3.2
Correspondence Problem	The difficulty in mapping motions from one morphology to another	Section 4.3
CSP	Constraint Satisfaction Problem	Section 3.2.2

Term	Description	Reference
Data gloves	Sensor system to measure angles of human finger joints	Section 4.1.1
Demonstration-based Generalization	Interactive planning model generalization based on simulated robot trials and additional human demonstrations	Section 4.4
DMP	Dynamic Movement Primitive	Section 2.4.2
DOF	Degree of freedom	Section 1.1
Flexibility	Ability to execute a manipulation task in different environments with different objects, obstacles and start configurations	Section 1.1
Generalization	Process to identify and remove inconsistent constraints in a planning model	Section 4.4, Section 4.5
GMM	Gaussian Mixture Model. Multi-modal probability distribution	Section 2.4.1
GMR	Gaussian Mixture Regression. Transformation of a GMM into a representation suitable for control	Section 2.4.1
Hand model	Articulated 3D model of the human hand	Section 4.1.1
Irrelevant constraint	Artificial restriction of the search space, which can be removed to increase flexibility without lowering the task success	Section 4.4
Justin	Anthropomorphic robot by DLR	Section 5.7
KRL	KUKA Robot Language. Robot programming language for KUKA robots	Section 2.4
Magnetic field tracker	Sensor system to measure human wrist poses	Section 4.1.1
Motion planning	Goal-directed planning of a robot trajectory	Section 2.3
Morphing	Transformation of one 3D model into a second 3D model	Section 3.3.1
MPK	Motion Planning Kit. Motion planning framework	Appendix C
Node constraints	Constraint-based definition of the goal of a manipulation task	Section 3.2
ODE	Open Dynamics Engine. Physics simulation	Section 3.4.3
OpenRAVE	Robot simulation framework	Section 2.4
PbD	Programming by Demonstration	Section 2.4
PDF	Probability density function	Section 2.4.1

Term	Description	Reference
Planning	“The task of coming up with a sequence of actions that will achieve a goal”	[88, p. 375]
Planning-model-consistent Segmentation	Segmentation of a path, which is consistent with the goals and constraints of a planning model	Definition 5
PR2	Service robot by Willow Garage	Section 5.7
Preliminary planning model	Usually overspecialized planning model generated based on an initial set of human demonstrations	Section 4.2
PRM	Probabilistic Roadmap	Section 2.1.1
Projection techniques	Projection of a given configuration to a constraint manifold used to plan efficiently on complex constraint manifolds	Section 3.4.1
Robot-Test-based Generalization	(Semi-)automatic planning model generalization based on user-defined tests	Section 4.5
ROS	Robot Operating System	Section 2.4
RRT	Rapidly-Exploring Random Tree	Section 2.1.2
SAH	Schunk Anthropomorphic Hand. Robot hand with three fingers and one opposing thumb	Section 5.1.1
Search heuristic	Constraint-based encoding of motion planning results	Section 4.6
Segmentation	Specification of a number of time points to segment a continuous path	Definition 3
Strategy graph	Graph-like representation of manipulation tasks	Section 3.2
Symbol grounding	Mapping of a symbolic state to the corresponding set of continuous, real-world states	Section 1.1
Tactile gloves	Sensor system to measure forces in the human fingertips	Section 4.1.1
TCP	Tool-Center-Point. A user-defined coordinate frame rigidly attached to the last joint of the robot manipulator	Section 1.1
Temporal constraint	Constraints defining the temporal ordering of subgoals and duration of transitions between subgoals	Section 3.1.2
Workspace comparison	Analysis of the differences between the workspaces of human and robot fingers	Section 4.3.1

Symbol	Description
\mathcal{U}	Input state space
\mathcal{X}	Internal state space
Θ	Output state space, configuration space
θ	Configuration
τ, φ	Constraints
$\mathcal{M}(\tau)$	Constraint manifold of τ
$d_{\mathcal{M}}(\tau, \theta)$	Distance of configuration θ to constraint manifold of τ
\mathcal{E}	Set of entities
e_1, e_2, e_3	Entity
f	Constraint function
f_{cart}	Cartesian constraint function
f_{pos}	Position constraint function
f_{ori}	Orientation constraint function
f_{dir}	Direction constraint function
f_{for}	Force constraint function
f_{mom}	Moment constraint function
f_{con}	Contact constraint function
f_{cdis}	Contact position constraint function
f_{cdir}	Contact distance constraint function
f_{qual}	Grasp quality constraint function
f_{coll}	Collision constraint function
R	Constraint region
P	Planning model
\mathcal{X}	Variables in the planning model
Ψ	Set of domain constraints
Ψ_{node}	Domain node constraints in a planning model
Ψ_{arc}	Domain arc constraints in a planning model
Ψ_{SH}	Set of search heuristic constraints
$\Psi(t)$	Set of constraints overlapping time point t in a planning model
Υ	Set of temporal constraints
Υ_{arc}	Temporal arc constraints in a planning model
Υ_{node}	Temporal node constraints in a planning model
\mathcal{D}_i	Domain of variable x_i in constraint satisfaction problem
\mathcal{T}	Tree in RRT motion planner
\mathcal{O}	Set of objects
$o(e_1)$	Object assigned to coordinate frame e_1
\mathcal{F}	Set of coordinate frames

Symbol	Description
δ_{Hand}	Segmentation threshold for hand motions
δ_{Finger}	Segmentation threshold for finger motions
δ_{Force}	Segmentation threshold for fingertip forces
δ_{Time}	Threshold for temporal segmentation
$\delta_{Contact}$	Threshold for contact calculation
δ_f	Threshold for violation of constraint with constraint function f
\mathcal{H}_j	Discretization of workspace of human finger j
\mathcal{R}_j	Discretization of workspace of robot finger j
$\mathcal{B}(x, r)$	Closed sphere with center x and radius r
π_i, φ_j	Path in Cartesian space
$\theta(t)$	Path in configuration space
Π	Warped path calculated by Dynamic Time Warping
$d_{DTW}(\pi, \varphi)$	Distance function of Dynamic Time Warping with path π and φ
P_c	Minimum number of elements in cluster
P_d	Maximum clustering distance
P_g	Number of Gaussians to generate search heuristic
$\mathcal{U}(X)$	Uniform distribution on X
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean μ and covariance matrix Σ

B. Strategy Graph: Operators and Metrics

In the learning process, a database of planning models is obtained. In order to demonstrate different parts of a task separately and combine different planning models, we define metrics to compare different planning models and operators to combine planning models.

B.1. Strategy Graph Distance Metrics

The distance $d(\tau_1, \tau_2)$ between two constraints $\tau_i = (e_{i_1}, \dots, e_{i_{m_i}}, f_i, R_i)$ is defined as ∞ if the type is different, i.e. $f_1 \neq f_2$, or the set of entities differs, i.e. $(e_{1_1}, \dots, e_{1_{m_1}}) \neq (e_{1_1}, \dots, e_{1_{m_1}})$. Otherwise, the distance is computed based on the overlap of the regions R_i . We define it as the Hausdorff distance $d_H(R_1, R_2)$, which is often used in computer-vision for template matching [45]:

$$d(\tau_1, \tau_2) = d_H(R_1, R_2) = \max\left\{ \sup_{r_1 \in R_1} \inf_{r_2 \in R_2} \|r_1 - r_2\|, \sup_{r_2 \in R_2} \inf_{r_1 \in R_1} \|r_1 - r_2\| \right\} \quad (\text{B.1})$$

Constraints are the atomic elements of a planning model. Consequently, we define the distance between two planning models based on the distance of its constraints. In order to define the distance $d(P_1, P_2)$ of two planning models $P^i = (\mathcal{X}^i, \Upsilon_{node}^i, \Psi_{node}^i, \Upsilon_{arc}^i, \Psi_{arc}^i, \theta)$ a bijective mapping of the nodes, arcs and constraints is necessary. If no mapping exists, the distance is defined as ∞ . Otherwise, we calculate the sum of constraint distances for each mapping normalized by the number of constraints. The distance is defined as the minimum of all normalized sums of constraint distances.

The distance metric is very restrictive since the type and entities respectively the number and ordering of nodes have to be equal. They are sufficient to identify similar (sub-)graphs or (sub-)sets of constraints in two planning models.

In the learning process, we are interested in calculating the distance of a path, e.g. normally a human demonstration, to an already learned planning model. If we identify the planning model with its constraint manifold it is natural to define the distance as the maximum distance of the path to the constraint manifold. Since the constraint manifold is defined implicitly by a large number of constraints, the maximum distance of the path to all constraints has to be calculated. In order to determine the set of constraints, which have to be obeyed at a given time point, a *planning-model-consistent segmentation* of the path is necessary.

Definition 5 (Planning-model-consistent Segmentation) *A path is a continuous function $\pi : [0, 1] \rightarrow \Theta$. A segmentation of the path π is a tuple (t_0, \dots, t_n) with $t_i \in [0, 1], t_0 = 0, t_n = 1, t_i \leq t_{i+1}$.*

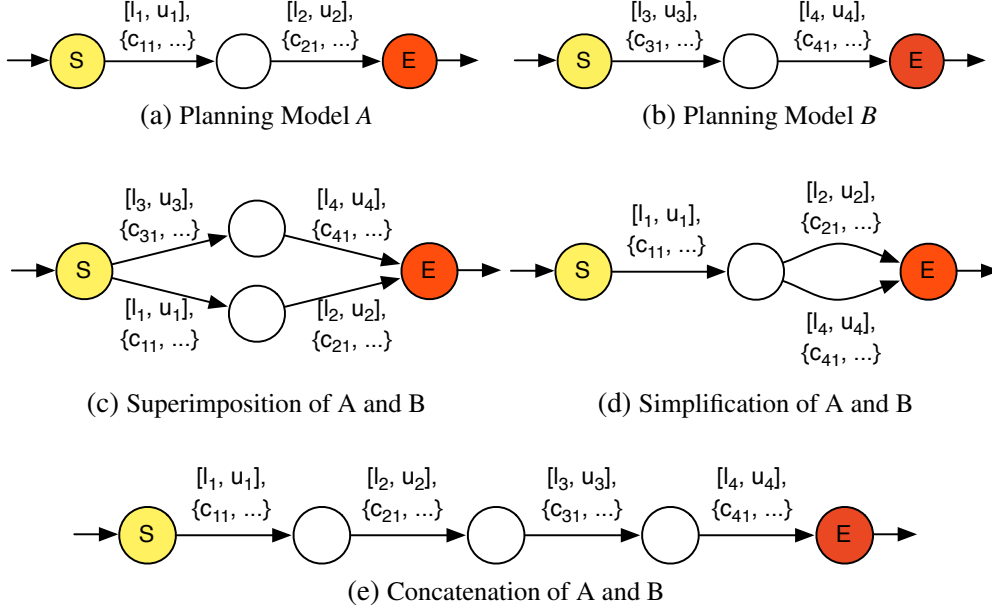


Figure B.1.: Superimposition, simplification and concatenation

The segmentation is consistent with the planning model $(\mathcal{X}, \Upsilon_{node}, \Psi_{node}, \Upsilon_{arc}, \Psi_{arc}, \theta)$ if a permutation $(t_{(0)}, \dots, t_{(n)})$ exists, for which all temporal constraints are obeyed with $x_i = t_{(i)}$.

The distance $d(\pi, P)$ of a path π with segmentation (t_0, \dots, t_n) to the constraint manifold defined by the planning model $P = (\mathcal{X}, \Upsilon_{node}, \Psi_{node}, \Upsilon_{arc}, \Psi_{arc}, \theta)$ is less restricting than the distance metric. It can be used to map human demonstrations to existing planning models, e.g. to learn hierarchies of planning models. If the number of segmentation points t_i and number of variables differ or $x_i = t_i$ violates the temporal constraints, the distance is ∞ . Otherwise, the distance is the maximum of all constraint distances:

$$d(\theta', t_k, P) = \max \left\{ \begin{aligned} &\max_{\tau \in \Psi_{node}(i)} d_{\mathcal{M}}(\tau, \theta'(t_i)), \\ &\max_{\substack{\varphi \in \Psi_{arc}(i,j) \\ t_i \leq t \leq t_j}} d_{\mathcal{M}}(\varphi, \theta'(t)) \end{aligned} \right\} \quad (\text{B.2})$$

B.2. Strategy Graph Operators

An important consequence of the strategy graph definition is that unique nodes S , the source, and E , the sink, exist representing the start time point and stop time point of the manipulation task. In Figure B.1a and B.1b two planning models are shown. For simplicity, node constraints are hidden. The closed interval $[l_i, u_i]$ represents the temporal constraint and c_{ik} the set of domain constraints of the corresponding arc.

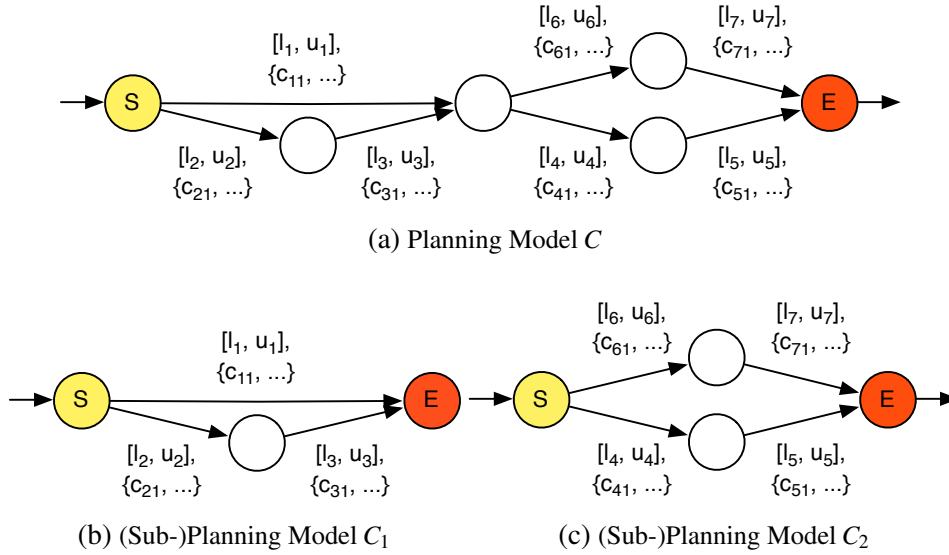


Figure B.2.: Decomposition

We superimpose two planning models by joining the set of nodes (without S and E) and the sets of temporal and domain constraints, see Figure B.1c. The resulting planning model represents the intersection of both planning models.

The simplification of two planning models is shown in Figure B.1d. We join two nodes if the temporal and domain constraints are identical. Two arcs are combined if the start- and end-node and the temporal and domain constraints are equal, e.g. $l_1 = l_3, u_1 = u_3, c_{1i} = c_{3i}$ in Figure B.1d.

The concatenation of two planning models is obtained by replacing the sink of the first planning model with the source of the second planning model, see Figure B.1e.

In order to decompose a planning model into two (sub-)planning models we identify a node (not S or E), whose removal leads to two unconnected subgraphs. This splitting node serves as source respectively sink of the two (sub-)planning models, see Figure B.2.

B.3. Strategy Graph Linearization

The goal is to transform a planning model $(\mathcal{X}, \Upsilon_{node}, \Psi_{node}, \Upsilon_{arc}, \Psi_{arc}, \theta)$ into a linear planning model $(\mathcal{X}, \Upsilon'_{node}, \Psi'_{node}, \Upsilon'_{arc}, \Psi'_{arc}, \theta)$, where a unique start node x_s , an end node x_e and a permutation $(x_{i_1}, \dots, x_{i_n}), x_s = x_{i_1}, x_e = x_{i_n}$ exists with

$$|\Upsilon_{arc}| = |\mathcal{X}| - 1 \quad (\text{B.3})$$

$$\forall k (i_k, i_{k+1}) \in \Upsilon_{arc}, \Upsilon_{arc}(i_k, i_{k+1}) \neq \emptyset \quad (\text{B.4})$$

Let $[l_i, u_i] \in \Upsilon_{node}(i)$ be the single temporal constraint of variable x_i . We define a strict temporal ordering of the variables x_i by $x_{(i)} \leq x_{(j)} \Leftrightarrow l_i + u_i \leq l_j + u_j$, i.e. by comparing the center of closed intervals. The domain constraints of arc $((i-1), (i))$ contain all constraints of all arcs, which

overlap $((i-1), (i))$. The domain constraints of node (i) are identical. The temporal constraints of arc $((i-1), (i))$ consist of a single closed interval, where the left bound is the minimum of the left bounds of all temporal constraints, which restrict the transition to $((i))$, and the right bound is the maximum of all right bounds. The temporal constraints of node $((i))$ are defined based on the temporal constraints of the previous node. The bounds of the previous node will be extended using the temporal arc constraint connecting $((i-1))$ and $((i))$. The transformation, which will be called *linearization*, is defined [52] as

$$\Psi_{arc}'((i-1), (i)) = \bigcup_{\substack{j < (i) \\ k > (i-1)}} \Psi_{arc}(j, k) \quad (\text{B.5})$$

$$\Psi_{node}'((i)) = \Psi_{node}((i)) \quad (\text{B.6})$$

$$\Upsilon_{arc}'((i-1), (i)) = \left[\min_{\substack{[l,u] \in \Upsilon_{arc}(k,(i)) \\ \forall k}} l, \max_{\substack{[l,u] \in \Upsilon_{arc}(k,(i)) \\ \forall k}} u \right] \quad (\text{B.7})$$

$$\Upsilon_{node}'((0)) = \Upsilon_{node}((0)) \quad (\text{B.8})$$

$$\Upsilon_{node}'((i)) = [l + a, u + b], [l, u] \in \Upsilon_{node}'((i-1)), \quad (\text{B.9})$$

$$[a, b] \in \Upsilon_{arc}'((i-1), (i)), i > 0 \quad (\text{B.10})$$

In general, temporal information about the ordering of subgoals will be lost but the set of domain constraints is maintained, which is sufficient for simple manipulation tasks.

C. Software Implementation

Figure C.1 shows an overview of the developed software system. *MS-PbD* is short for Manipulation Strategy Programming by Demonstration. It refers to the learning system developed in Section 4.2. *MS-Generalization* is the implementation of the demonstration-based generalization algorithm in Section 4.4. *MS-Planner* holds the implementation of the planning algorithms in Section 3.2.2 and 3.4. The parallelized, evolutionary algorithm to generalize the learned planning model using robot-tests, see Section 4.5, is implemented in *MS-Distributed Generalization*. *MS-Search heuristics* refers to the implementation of the learning and planning of search heuristics in Section 4.6. The controllers as well as software interfaces to read sensors and write actuators on Adero and Albert II were developed by Xue et al. using the Modular Controller Architecture (MCA2) framework. In order to define and simulate robots, we integrated the simulation frameworks OpenRAVE [32] and Motion Planning Kit (MPK) [67]. Existing software at the Humanoids and Intelligence Systems Lab (HIS) to acquire sensor data in the sensory environment was interfaced using the Robot Operating System (ROS) [85]. The interface to Justin was provided by Florian Schmidt et al. (Deutsches Zentrum für Luft- und Raumfahrt) and is skipped for clarity.

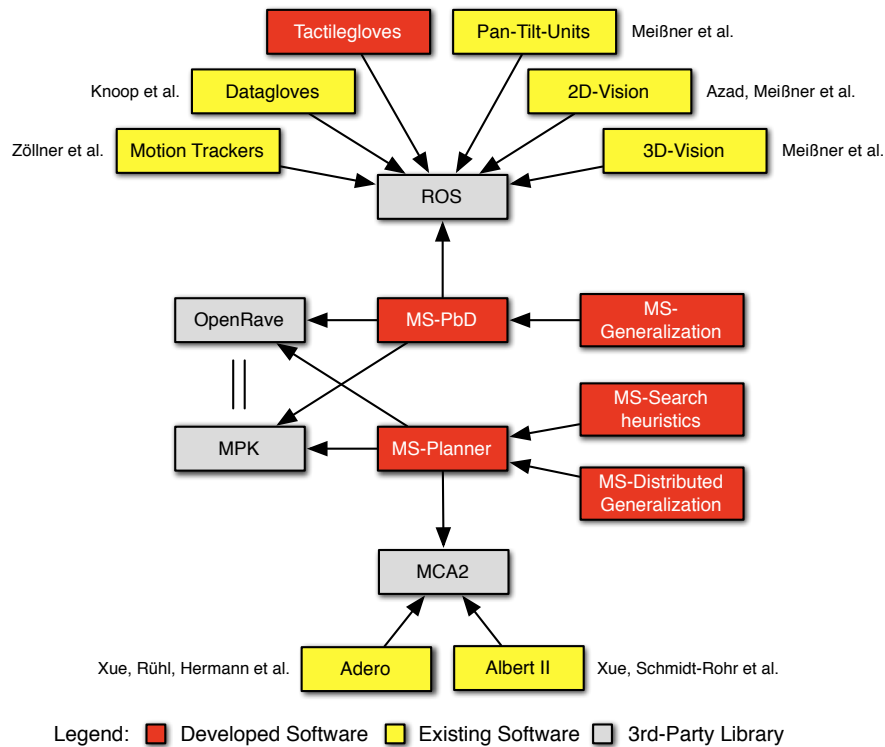


Figure C.1.: Overview of developed software system.

D. Prior Publications and Research Collaboration

The author of this thesis is the first author of the following publications, which were cited in the respective sections, tables and figures in Chapter 3, Chapter 4 and Chapter 5:

1. Rainer Jäkel, S. R. Schmidt-Rohr, M. Loesch, and R. Dillmann. Representation and constrained planning of manipulation strategies in the context of programming by demonstration. In IEEE International Conference on Robotics and Automation, May 2010.
2. Rainer Jäkel, S. R. Schmidt-Rohr, Z. Xue, M. Loesch, and R. Dillmann. Learning of probabilistic grasping strategies using programming by demonstration. In IEEE International Conference on Robotics and Automation, May 2010.
3. Rainer Jäkel, S. R. Schmidt-Rohr, M. Loesch, A. Kasper, and R. Dillmann. Learning of generalized manipulation strategies in the context of programming by demonstration. In IEEE-RAS 10th International Conference on Humanoid Robots, December 2010.
4. Rainer Jäkel, P. Meißner, S. R. Schmidt-Rohr, and R. Dillmann. Distributed generalization of learned planning models in robot programming by demonstration. In IEEE/RSJ International Conference on Intelligent Robots and Systems, September 2011.
5. Rainer Jäkel, S. R. Schmidt-Rohr, S. W. Rühl, A. Kasper, Z. Xue, and R. Dillmann. Learning of planning models for dexterous manipulation based on human demonstrations. International Journal of Social Robotics, 2011.
6. Rainer Jäkel, Y. Xie, P. Meißner, and R. Dillmann. Online-generation of task-dependent search heuristics to execute learned planning models in programming by demonstration. In IEEE-RAS 11th International Conference on Humanoid Robots, October 2012.

In all publications, the author carried out problem formulation, solution finding as well as visualization and participated in and crucially steered discussion and experimental evaluation.

The implementation of the learning algorithm in Section 4.6 as well as a part of the experimental evaluation in Section 5.4 was done in the following students thesis, in which the author was decisively included:

1. Yi Xie. Automatische Erzeugung Task-abhängiger Suchheuristiken zur Beschleunigung von Bahnplanungsalgorithmen. Students thesis, Betreuer: Rainer Jäkel, Gutachter: Rüdiger Dillmann, Karlsruher Institut für Technologie, 2012.

The author participated in and crucially steered problem formulation and solution finding, discussion, visualization and experimental evaluation in the students thesis.

The hand model in Section 4.1.1 was created in collaboration with Alexander Kasper, who scanned the plaster cast to create the initial 3D model. The tactile gloves in Section 4.1.1 were developed in collaboration with the student assistant Alexander Bachmeier.

The thesis itself represents a novel contribution by presenting and evaluating the complete, refined concept as well as the developed process in a coherent way.

E. List of Figures

1.1	Examples of service robots	2
1.2	Manipulation task representations	3
1.3	Sensor environment to observe manipulation tasks	6
1.4	Overview of developed PbD process	8
1.5	Examples of the execution of learned manipulation tasks	11
2.1	Probabilistic roadmap	15
2.2	Unidirectional RRT	19
2.3	Connection of Voronoi regions and RRT tree extension	20
2.4	Projection techniques for constraint-based motion planning	22
2.5	Workspace decomposition	25
2.6	Workspace Biasing	26
2.7	Watershed transformation	27
2.8	Task Frame example: screwing	28
2.9	Task Frame example: grasping	29
2.10	Imitation Learning with GMMs	31
2.11	Imitation Learning with GMMs and DBNs	32
2.12	Imitation Learning with DMPs	34
2.13	Formalization of PbD process	35
2.14	PbD system using POMDPs	37
2.15	Features and feature selection in PbD.	38
2.16	Principal Component Analysis: reduction of dimensionality	39
3.1	Modeling aspects in developed PbD process	46
3.2	Constraint example: pressing a button with a robot finger.	47
3.3	Input-Output-System	48
3.4	Different constraint representations of a single task	49
3.5	Example of position constraint	52
3.6	Example of direction constraint	53
3.7	Constraint-Sampling	55
3.8	Example of contact constraint	57
3.9	Geometric representations of constraint regions	59
3.10	Strategy graph: place a bottle in the fridge	63

3.11	Planning non-linear strategy graphs: constraint ambiguities	68
3.12	Planning non-linear strategy graphs: refinement of temporal domains	69
3.13	Example of a linear planning model	71
3.14	Blended intrinsic maps: examples	72
3.15	Constraint Projection: Randomized Gradient Descent	75
3.16	Contact projection: different start situations and projection result.	76
3.17	Rejection and Constraint-Sampling	78
3.18	Visualization of goal configuration generation during motion planning	79
3.19	Path smoothing	81
4.1	Overview of developed PbD process.	87
4.2	Sensor environment with two robot heads, magnetic field trackers, datagloves and tactile sensors to observe human demonstrations of a manipulation task.	89
4.3	Generation of user-dependent hand model	90
4.4	Tactile sensors to measure fingertip contact forces.	91
4.5	Visualization of sensor data	92
4.6	Human demonstrations: pour-in task	94
4.7	Learning the structure of a strategy graph: pour-in	96
4.8	Constraint mapping: from small to large bottle cap	97
4.9	Contact coordinate frames and mapping using Blended Intrinsic Maps	97
4.10	Overview of constraint 1 and 2	99
4.11	Constraint set 1: examples of manipulation tasks.	100
4.12	Constraint set 2: examples of manipulation tasks.	101
4.13	Learned constraints: pour-in	102
4.14	Visual comparison of human and robot hand	103
4.15	Workspace of robot and human fingers for a subset of finger motions	105
4.16	Workspace comparison	106
4.17	Demonstration-based generalization: human demonstrations	109
4.18	Demonstration-based generalization: result	109
4.19	Robot-test-based generalization: overview	111
4.20	Strategy graph with irrelevant constraints: place bottle in fridge door	112
4.21	Constraint inconsistencies: example	113
4.22	Robot-tests: example	113
4.23	Robot-test-based generalization: state	114
4.24	Robot-test-based generalization: statistics	115
4.25	Robot-test-based-generalization: mutation operator	118
4.26	Robot-test-based-generalization: process	120
4.27	Robot-test-based-generalization: result	120
4.28	Planning model specialization: place a bottle in the crate	122

4.29	Constraint specialization: place a bottle in the crate	123
4.30	Search heuristics: selection	125
4.31	Search heuristics: training examples	126
4.32	Search heuristics: learning	127
4.33	Search heuristics: planning process	129
5.1	Robots used in key experiments.	136
5.2	3D-models in KIT Object Models database	137
5.3	Grasp strategies: human grasps	139
5.4	Grasp strategies: planned grasps	140
5.5	Grasp strategies: execution of grasps on Adero	142
5.6	Planning: pour-in with variable cup, bottle and obstacle poses.	143
5.7	Planning: visualization of results in first setting	144
5.8	Planning: visualization of results in second setting	145
5.9	Planning: visualization of results in third setting	145
5.10	Human demonstrations of chess knight move	147
5.11	Visualization of constraints in chess knight task	147
5.12	Execution of chess knight task after first additional demonstration	148
5.13	Execution of chess knight task on a non-empty chessboard after generalization	149
5.14	Human demonstrations of the bottle closing task	150
5.15	Planning result of the bottle closing task with identical bottle	150
5.16	Planning result of the bottle closing task with a smaller bottle	151
5.17	Pour-in: constraints before and after generalization	153
5.18	Bottle in crate: examples of robot tests.	154
5.19	Execution of generalized planning model to press a key	154
5.20	Experiment: bottle opening	156
5.21	Execution of generalized planning model to lift a chair	156
5.22	Task A: visualization of task and used goal constraint	158
5.23	Task A3: obstacle arrangements	158
5.24	Task A1: execution of task using a single search heuristic	159
5.25	Task A1: training data and search heuristics	160
5.26	Task A1 and A2: comparison of training data	161
5.27	Task A2: execution on Adero	162
5.28	Task A3: incremental learning of search heuristics with different parameters	163
5.29	Task A3: incrementally learned search heuristics	163
5.30	Task B: grasp a bottle and place it in the fridge door	164
5.31	Task B1: learned search heuristics with different parameters	164
5.32	Task B2: learned search heuristics	166
5.33	Task B3: incremental learning	166

5.34	Task C: grasp a chips can and place it in a box on a rack	167
5.35	Task C1: learned search heuristics with different parameters	168
5.36	Task C1: execution on Albert II	169
5.37	Task C3: incremental learning	170
5.38	Search heuristics: comparison with control algorithm	171
5.39	Opening a bottle: visualization of constraints and search heuristics	173
5.40	Opening a bottle: overview of planning results	174
5.41	Opening a bottle: execution in similar setting on Adero	175
5.42	Opening a bottle: close-up of execution in similar setting on Adero	175
5.43	Lifting a spoon: human demonstrations	176
5.44	Lifting a spoon: mapping of contact coordinate frames	176
5.45	Lifting a spoon: constraints	177
5.46	Lifting a spoon: validation of planning result in physics simulation	177
5.47	Lifting a spoon: visualization of search heuristic 1.	178
5.48	Lifting a spoon: execution on Adero	178
5.49	Constraint specialization: grasping a bottle cap	180
5.50	Constraint specialization: pressing a key	181
5.51	Constraint specialization: placing a bottle in the fridge	182
5.52	Execution of pour-in task on Adero (version 1)	184
5.53	Execution of pour-in task on Adero (version 2)	184
5.54	Execution of pour-in task on Adero (version 3)	185
5.55	Execution of pour-in task on Albert II	185
5.56	Execution of pour-in task on Justin (DLR)	186
5.57	Execution of pour-in task on Armar IIIb	186
5.58	Execution of pour-in task on PR2	187
B.1	Superimposition, simplification and concatenation	204
B.2	Decomposition	205
C.1	Overview of developed software system.	207

F. List of Tables

2.1	Rating of selected related work	41
5.1	Grasp strategies: result of planned grasps with fixed object poses	138
5.2	Grasp strategies: result of precomputed grasps with fixed object poses	141
5.3	Grasp strategies: result of planned grasps with random object poses	141
5.4	Grasp strategies: result of precomputed grasps with random object poses	141
5.5	Planning: pour-in success rates in three settings	144
5.6	Planning: pour-in failures in three settings	146
5.7	Robot-test-based generalization: summary of results	152
5.8	Search heuristics: summary of results	157
5.9	Task A1: planning results	159
5.10	Task A1: speed-ups	161
5.11	Task A2: planning results	161
5.12	Task A2: speed-ups	162
5.13	Task B1: planning results	165
5.14	Task B2: planning results	165
5.15	Task B3: speed-ups	167
5.16	Task C1: planning results	168
5.17	Task C2: planning results	170
5.18	Search heuristics: overview of average speed-ups	172
5.19	Overview of constraint specialization results	180

G. Bibliography

- [1] A. Alissandrakis, C. L. Nehaniv, and K. Dautenhahn. Correspondence mapping induced state and action metrics for robotic imitation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):299–307, 2007.
- [2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, pages 155–168, 1998.
- [3] Ascension Technology Corporation, www.ascension-tech.com. *Flock of Birds. Installation and Operation Guide. Revision 910141 Rev B*, January 2002.
- [4] P. Azad, T. Gockel, and R. Dillmann. *Computer Vision – das Praxisbuch*. Elektor-Verlag, 2007.
- [5] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, Dec. 1996.
- [6] A. Baroncelli. IFR Vice President, AUTOMATICA 2012, Munich. May 2012.
- [7] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009.
- [8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.
- [9] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation*, May 2009.
- [10] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, 2009.
- [11] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Handbook of Robotics*, pages 1371–1394. Springer, Secaucus, NJ, USA, 2008.
- [12] A. Billard, Y. Epars, S. Calinon, S. Schaal, and G. Cheng. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77, 2004.

- [13] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.
- [14] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1018–1023, 1999.
- [15] C. Breazeal, M. Berlin, A. G. Brooks, J. Gray, and A. L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems*, 54(5):385–393, 2006.
- [16] O. Brock and L. E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *Proceedings of the International Conference on Robotics and Automation*, pages 1469–1474, 2001.
- [17] H. Bruyninckx and J. De Schutter. Specification of force-controlled actions in the task frame formalism - a synthesis. *IEEE Transactions on Robotics and Automation*, 12(4):581–589, aug 1996.
- [18] B. Bustos, D. A. Keim, D. Saupe, T. Schreck, and D. V. Vranić. Feature-based similarity search in 3d object databases. *ACM Computing Surveys*, 37, 2005.
- [19] S. Calinon and A. Billard. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proceedings of the 22nd international conference on Machine learning, ICML '05*, pages 105–112, New York, NY, USA, 2005. ACM.
- [20] S. Calinon and A. Billard. Teaching a humanoid robot to recognize and reproduce social cues. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 346–351, September 2006.
- [21] S. Calinon and A. Billard. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 367–372, September 2008.
- [22] S. Calinon, F. D’halluin, D. Caldwell, and A. Billard. Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework. In *Proceedings IEEE-RAS International Conference on Humanoid Robots*, pages 582–588, December 2009.
- [23] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):286–298, 2007.
- [24] U. G. Center. QHull, UIUC Geometry Center, QHull Computational Geometry Package. <http://www.qhull.org>, 2004.

-
- [25] J. Chestnutt, M. Lau, K. M. Cheung, J. Kuffner, J. K. Hodgins, and T. Kanade. Footstep planning for the honda asimo humanoid. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2005.
- [26] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.
- [27] M. R. Cutkosky. On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation*, 5(3):269–279, 1989.
- [28] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *International Journal of Robotics Research*, 26(5):433–455, May 2007.
- [29] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
- [30] W. Decré, R. Smits, H. Bruyninckx, and J. De Schutter. Extending itasc to support inequality constraints and non-instantaneous task specification. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation, ICRA'09*, pages 1875–1882, Piscataway, NJ, USA, 2009. IEEE Press.
- [31] G. C. Devol. U.S. Patent 2,988,237, 1954.
- [32] R. Diankov and J. Kuffner. OpenRAVE: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, July 2008.
- [33] R. Dillmann, M. Kaiser, and A. Ude. Acquisition of elementary robot skills from human demonstration. In *International Symposium on Intelligent Robotics Systems*, pages 185–192, 1995.
- [34] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zollner, and M. Bordegoni. Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm. In *Robotics Research-International Symposium*, volume 9, pages 229–238, 2000.
- [35] D. Eberly. Distance from a point to an ellipse, an ellipsoid, or a hyperellipsoid. *Geometric Tools, LLC, www.geometrictools.com*, 2011.
- [36] A. Eiben and J. Smith. *Introduction to evolutionary computing*. Springer, 2003.
- [37] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 358–363, sep. 2006.

- [38] C. Eppner, J. Sturm, M. Bennewitz, C. Stachniss, and W. Burgard. Imitation learning with generalized task descriptions. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 1804–1810, Piscataway, NJ, USA, 2009. IEEE Press.
- [39] C. Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*. Morgan Kaufmann, Jan. 2005.
- [40] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving. *Artificial Intelligence Journal*, 2:189–208, 1971.
- [41] M. Grebenstein, A. Albu-Schäffer, T. Bahls, M. Chalon, O. Eiberger, W. Friedl, R. Gruber, S. Haddadin, U. Hagn, R. Haslinger, H. Hoppner, S. Jörg, M. Nickl, A. Nothhelfer, F. Petit, J. Reill, N. Seitz, T. Wimböck, S. Wolf, T. Wüsthoff, and G. Hirzinger. The DLR hand arm system. In *IEEE International Conference on Robotics and Automation*, pages 3175–3182, 2011.
- [42] J. Himmelstein. *Geometric operators for motion planning*. PhD thesis, Institut National des Sciences Appliquées de Toulouse, 2008.
- [43] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1408–1413, 2000.
- [44] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings IEEE International Conference on Robotics and Automation*, 2003.
- [45] D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the hausdorff distance. *Computer Vision and Pattern Recognition*, (TR92-1321):705–706., 1993.
- [46] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *IEEE International Conference on Robotics and Automation*, pages 1398–1403, 2002.
- [47] N. Iyer, S. Jayanti, K. Lou, Y. Kalyanaraman, and K. Ramani. Three-dimensional shape searching: state-of-the-art review and future trends. *Computer-Aided Design*, 37(5):509–530, Apr. 2005.
- [48] L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon. Adaptive tuning of the sampling domain for dynamic-domain rrts. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2851–2856, 2005.

-
- [49] R. Jäkel, P. Meißner, S. R. Schmidt-Rohr, and R. Dillmann. Distributed generalization of learned planning models in robot programming by demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2011.
- [50] R. Jäkel, S. W. Rühl, S. R. Schmidt-Rohr, M. Lösch, Z. Xue, and R. Dillmann. Layered programming by demonstration and planning for autonomous robot manipulation. In B. Siciliano, editor, *Advanced Bimanual Manipulation*, volume 80 of *Springer Tracts in Advanced Robotics*, pages 1–57. Springer Berlin Heidelberg, 2012.
- [51] R. Jäkel, S. R. Schmidt-Rohr, M. Loesch, and R. Dillmann. Representation and constrained planning of manipulation strategies in the context of programming by demonstration. In *IEEE International Conference on Robotics and Automation (ICRA '10)*, May 2010.
- [52] R. Jäkel, S. R. Schmidt-Rohr, M. Loesch, A. Kasper, and R. Dillmann. Learning of generalized manipulation strategies in the context of programming by demonstration. In *IEEE-RAS 10th International Conference on Humanoid Robots (Humanoids 2010)*, December 2010.
- [53] R. Jäkel, S. R. Schmidt-Rohr, S. W. Rühl, A. Kasper, Z. Xue, and R. Dillmann. Learning of planning models for dexterous manipulation based on human demonstrations. *International Journal of Social Robotics*, 2011.
- [54] R. Jäkel, S. R. Schmidt-Rohr, Z. Xue, M. Loesch, and R. Dillmann. Learning of probabilistic grasping strategies using programming by demonstration. In *IEEE International Conference on Robotics and Automation (ICRA '10)*, May 2010.
- [55] R. Jäkel, Y. Xie, P. Meißner, and R. Dillmann. Online-generation of task-dependent search heuristics to execute learned planning models in programming by demonstration. In *IEEE-RAS 11th International Conference on Humanoid Robots (Humanoids 2012)*, October 2012.
- [56] C. Je, M. Tang, Y. Lee, M. Lee, and Y. J. Kim. Polydepth: Real-time penetration depth computation using iterative contact-space projection. *ACM Transactions on Graphics*, 31(1):5:1–5:14, Feb. 2012.
- [57] J. K. Jr, J. Koichi, N. S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *Proceedings of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 500–505, 2001.
- [58] A. Kasper. KIT ObjectModels Web Database, <http://www.iain.ira.uka.de/ObjectModels>.
- [59] A. Kasper, Z. Xue, and R. Dillmann. A object model database for object recognition, localisation and manipulation in service robotics. *International Journal on Robotics Research (IJRR)*, 2012.

- [60] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, June 1996.
- [61] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: A general purpose evolutionary computation library. *Artificial Evolution*, 2310:829–888, 2002.
- [62] V. Kim, Y. Lipman, and T. Funkhouser. Blended intrinsic maps. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 30(4), July 2011.
- [63] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84(1-2):171–203, 2011.
- [64] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3232–3237, 2010.
- [65] J. Kuffner, J.J. and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995 –1001 vol.2, 2000.
- [66] J. J. Kuffner and S. M. LaValle. An efficient approach to path planning using balanced bidirectional RRT search. Technical Report CMU-RI-TR-05-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Aug. 2005.
- [67] J. Latombe, F. Schwarzer, and M. Saha. Motion planning kit. Technical report, <http://robotics.stanford.edu/mitul/mpk/>, 2006.
- [68] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, Oct. 1998.
- [69] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at <http://planning.cs.uiuc.edu/>.
- [70] D. Leidner. Pfadplanung für Aufgaben in der realen Welt ausgeführt durch reale Robotersysteme. Master’s thesis, Deutsches Zentrum für Luft- und Raumfahrt, Hochschule Mannheim, 2011.
- [71] J. Leigh and I. of Electrical Engineers. *Control Theory, pp. 61*. IEE Control Engineering Series. Institution of Electrical Engineers, 2004.
- [72] J.-M. Lien, S. L. Thomas, and N. M. Amato. A general framework for sampling on the medial axis of the free space. In *Proceedings IEEE International Conference on Robotics and Automation*, 2003.

-
- [73] M. T. Mason. Compliance and force control for computer-controlled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):418–432, 1981.
- [74] P. Meißner, S. R. Schmidt-Rohr, M. Lösch, R. Jäkel, and R. Dillmann. Localization of furniture parts by integrating range and intensity data robust against depths with low signal-to-noise ratio. *Robotics and Autonomous Systems*, 61, 2012.
- [75] M. Mühlig, M. Gienger, and J. Steil. Human-robot interaction for learning and adaptation of object movements. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4901–4907, 2010.
- [76] M. Mühlig, M. Gienger, J. Steil, and C. Goerick. Automatic selection of task spaces for imitation learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4996–5002, oct. 2009.
- [77] K. Mülling, J. Kober, and J. Peters. Simulating human table tennis with a biomimetic robot setup. In *From Animals to Animats 11: Eleventh International Conference on the Simulation of Adaptive Behavior*, pages 273–282, Berlin, Germany, 08 2010. Springer.
- [78] M. N. Nicolescu and M. J. Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 241–248, 2003.
- [79] S. Nof. *Handbook of Industrial Robotics*. Number Bd. 1 in Electrical and electronic engineering. John Wiley, 1999.
- [80] M. Pardowitz. *Inkrementelles und interaktives Lernen von Handlungswissen für Haushalt-roboter*. PhD thesis, Universität Karlsruhe, 2007.
- [81] M. Pardowitz, R. Zollner, S. Knoop, and R. Dillmann. Using physical demonstrations, background knowledge and vocal comments for task learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 322–327, oct. 2006.
- [82] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE International Conference on Robotics and Automation*, 2009.
- [83] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. Skill learning and task outcome prediction for manipulation. In *IEEE international conference on Robotics and Automation*, 2011.
- [84] M. Prats, P. J. Sanz, and A. P. D. Pobil. A framework for compliant physical interaction. *Autonomous Robots*, 28(1):89–111, 2010.

- [85] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, 2009.
- [86] H. H. Rosenbrock. An automatic method of finding the greatest or least value of a function. In *Computer Journal* 3, 1960.
- [87] L. Rozo, P. Jiménez, and C. Torras. Robot learning from demonstration of force-based tasks with multiple solution trajectories. In *IEEE International Conference on Advanced Robotics*, June 2011.
- [88] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd Edition*. Prentice-Hall, Englewood Cliffs, NJ, 2003.
- [89] P. Rybski, K. Yoon, J. Stolarz, and M. Veloso. Interactive robot task training through dialog and demonstration. In *2nd ACM/IEEE International Conference on Human-Robot Interaction*, March 2007.
- [90] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.
- [91] S. Schaal, J. Peters, J. Nakanishi, and A. J. Ijspeert. Learning movement primitives. In *International Symposium on Robotics Research*, pages 561–572, 2003.
- [92] S. Schaal, S. Vijayakumar, A. Ijspeert, and J. Nakanishi. Real-time statistical learning for robotics and human augmentation. In *Syntax and semantics 26: Semantics and the*, pages 117–124. Academic Press, 2001.
- [93] S. R. Schmidt-Rohr. *Interactive Learning of Probabilistic Decision Making by Service Robots Considering Multiple Skill Domains*. PhD thesis, Karlsruher Institut für Technologie, Fakultät für Informatik, 2012.
- [94] S. R. Schmidt-Rohr, M. Lösch, R. Jäkel, and R. Dillmann. Programming by demonstration of probabilistic decision making on a multi-modal service robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipeh, Taiwan, 2010.
- [95] E. Schwalb and R. Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93:29–61, 1995.
- [96] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6), 2000.

-
- [97] A. Skoglund, J. Tegin, B. Iliev, and R. Palm. Programming-by-demonstration of reaching motions for robot grasping. In *Proceedings of the 2009 14th International Conference on Advanced Robotics*, June 24-26 2009.
- [98] D. Song, K. Huebner, V. Kyrki, and D. Kragic. Learning task constraints for robot grasping using graphical models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1579–1585. IEEE, Oct. 2010.
- [99] S. Stančin and S. Tomažič. Angle estimation of simultaneous orthogonal rotations from 3d gyroscope measurements. *Sensors*, 11(9):8536–8549, 2011.
- [100] J. Steffen, C. Elbrechter, R. Haschke, and H. Ritter. Bio-inspired motion strategies for a bimanual manipulation task. In *10th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 625–630, dec. 2010.
- [101] M. Stilman. Task constrained motion planning in robot joint space. *IEEE/RAS International Conference on Robots and Systems*, 2007.
- [102] M. Strandberg. Augmenting RRT-planners with local trees. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3258–3262, 2004.
- [103] T. Sugaiwa, G. Fujii, H. Iwata, and S. Sugano. A methodology for setting grasping force for picking up an object with unknown weight, friction, and stiffness. In *IEEE-RAS 10th International Conference on Humanoid Robots (Humanoids)*, pages 288–293, 2010.
- [104] M. Tang, Y. Kim, and D. Manocha. C2A: Controlled conservative advancement for continuous collision detection of polygonal models. In *IEEE International Conference on Robotics and Automation*, pages 849–854. IEEE, 2009.
- [105] M. Tang, Y. J. Kim, and D. Manocha. C2a: Controlled conservative advancement for continuous collision detection of polygonal models. *Proceedings of International Conference on Robotics and Automation*, 2009.
- [106] University of North Carolina. PQP: A proximity query package. GAMMA Research Group, Available from <http://www.cs.unc.edu/~geom/SSV/>, 2005.
- [107] J. van den Berg and M. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 1, pages 453–460, 2004.
- [108] V. N. Vapnik and A. Y. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, USSR, 1974.

- [109] H. Veeraraghavan and M. M. Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604, 2008.
- [110] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.
- [111] C. Wischnewski. Arbeitsraumbezogene Analyse der Wahrscheinlichkeiten von Aktionseffekten für probabilistisches Entscheiden eines autonomen Roboters. Master’s thesis, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, Karlsruher Institut für Technologie, 2012.
- [112] Y. Xie. Automatische Erzeugung Task-abhängiger Suchheuristiken zur Beschleunigung von Bahnplanungsalgorithmen. Students thesis, Betreuer: Rainer Jäkel, Gutachter: Rüdiger Dillmann, Karlsruher Institut für Technologie, 2012.
- [113] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings IEEE International Conference on Robotics and Automation*, 2005.
- [114] L. Zhang and D. Manocha. An efficient retraction-based rrt planner. In *IEEE International Conference on Robotics and Automation*, may 2008.
- [115] R. Zöllner, T. Asfour, and R. Dillmann. Programming by demonstration: Dual-arm manipulation tasks for humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.
- [116] M. Zucker, J. Kuffner, and J. Bagnell. Adaptive workspace biasing for sampling-based planners. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3757–3762, May 2008.