

BERLIN INSTITUTE OF TECHNOLOGY

A THESIS SUBMITTED FOR THE DEGREE OF  
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ELECTRICAL ENGINEERING

---

# Optimizing Motion Primitives to Integrate Symbolic and Motion Planning

---

*Author:*

Andreas ORTHEY

*Supervisor:*

Prof. Dr. Marc TOUSSAINT

*Second Supervisor:*

Prof. Dr. Oliver BROCK

*Thesis Advisor*

Dr. Tobias LANG

September 4, 2012



## Abstract

To advance current robot technology, we think it essential for a robot to autonomously execute a high-level task like “clean up the room”. One fundamental problem of this execution is how we can translate the high-level task into a motion trajectory of the robot. Usually, this is approached by using two separated planners: A symbolic planner and a motion planner. But this separation often fails to find a successful plan, due to the abstractness of the symbolic planner or uncertainty in the environment. We therefore describe an approach to increase the probability to find a successful motion trajectory, by using a closer integration of symbolic planning and motion planning. This integration is accomplished by defining a motion primitive for each symbolic action, which describes the constraints of the underlying motion planner. Each motion primitive is then optimized with respect to the success of a symbolic action, by using *covariance matrix adaptation* (CMA). Eventually, we conducted a series of simulated experiments, to show that this approach is able to increase the probability of a successful motion trajectory, and that we can use the optimized motion primitives to generalize to unseen scenarios.



## Zusammenfassung

Um heutige Robotik Technology voranzubringen, ist es essentiell, dass ein Roboter abstrakte Aufgaben, wie z.B. "Räum das Zimmer auf", autonom ausführen kann. Ein Kernproblem dieser Ausführung beinhaltet, dass wir die abstrakte Aufgabe in einen Bewegungsplan des Roboters übersetzen müssen. Um dieses Problem zu lösen werden normalerweise zwei getrennte Planner benutzt: Ein Symbolischer Planner und ein Bewegungsplaner. Diese Trennung führt jedoch zu weniger erfolgreichen Planungen, da erstens der Symbolische Planner zu abstrakt ist, und zweitens, da diverse Unsicherheiten in der Umwelt existieren. Daher ist das Ziel dieser Arbeit einen Ansatz zu beschreiben, wie wir die Wahrscheinlichkeit von erfolgreichen Planungen erhöhen können, indem wir Symbolisches Planen und Bewegungsplanen integrieren. Diese Integration wird erreicht, indem wir für jede symbolische Aktion ein Bewegungs Primitiv definieren, welches die Randbedingungen des Bewegungsplanners beschreibt. Unsere Hauptbeitrag ist dann, dass wir diese Bewegungs Primitive optimieren, hinsichtlich ihrer Wahrscheinlichkeit, dass sie zu erfolgreichen Planungen führen. Dazu benutzen wir die Optimierungsstrategie *covariance matrix adaptation* (CMA). In simulierten Experimenten zeigen wir anschließend, dass dieser Ansatz die Wahrscheinlichkeit von erfolgreichen Planung erhöht, und dass wir die optimierten Bewegungs Primitive benutzen können, um auf unbekannte Situationen zu verallgemeinern.



# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
1.1	Contributions	2
1.2	Definitions and Outline	3
<b>II</b>	<b>Background in Symbolic and Motion Planning</b>	<b>5</b>
2.1	Symbolic Planning with Relational Representations	5
2.1.1	Relational Representations	6
2.1.2	Uncertainty in Symbolic Representations	7
2.1.3	Probabilistic State Transitions	7
2.1.4	Symbolic Planning with PRADA	8
2.1.5	Open problems of symbolic planning	9
2.2	Motion Planning by Inference	10
2.2.1	From Motion Planning to Stochastic Optimal Control	10
2.2.2	Casting SOC as an Inference Problem	11
2.2.3	Task Variables and Task Targets	13
2.2.4	Precision of Task Variables	13
<b>III</b>	<b>Related Work</b>	<b>15</b>
3.1	Combination of Symbolic and Motion Planning	15
3.1.1	Hierarchical Top-Down Approach	16
3.1.2	Heuristics to predict Feasibility of Symbolic Actions	17
3.1.3	Bottom-Up: Symbolic State from C-Space	18
3.2	Motion Primitives	21
3.2.1	Attractor Based Models	21
3.2.2	Via Point Models	22
3.2.3	From Motion Primitives to Motion Planning	22
<b>IV</b>	<b>Integrating Symbolic and Motion Planning</b>	<b>25</b>
4.1	Formalization of a Motion Primitive $\mathcal{P}$	26
4.2	Objective Function from $\mathcal{P}$	28
4.3	Mappings to evaluate $\alpha(\mathcal{P})$	29
4.3.1	$\Psi_{q \rightarrow \beta}$	32
4.3.2	$\Psi_{\beta \rightarrow \gamma}$	35
4.4	Covariance Matrix Adaption (CMA) to Optimize $\mathcal{P}$	36
4.5	Final Remarks on Optimizing $\mathcal{P}$	37
4.5.1	General Assumptions	37
4.5.2	Time for Offline Optimization	38
<b>V</b>	<b>Experiments</b>	<b>39</b>
5.1	Optimizing Grasping Action	42

5.2	Optimizing Placing Action . . . . .	50
5.3	Optimizing Homing Action . . . . .	56
5.4	Discussion of Results . . . . .	61
<b>VI</b>	<b>Conclusion</b> . . . . .	<b>62</b>
6.1	Future Research Directions . . . . .	63



# Chapter I

## Introduction

A desirable scenario for an advanced robotic system is the ability to execute a high level task like “clean up the room”. Such a task can be divided into smaller subtask like “grasp coffee mug” and “place coffee mug in dishwasher”. Each of those subtasks has to be executed on the motion control level, which means that we need to find a trajectory, consisting of joint torques of the robot.

This possesses still a fundamental problem for robotics and artificial intelligence, mainly because it was assumed, that we can separate the problem by using symbolic planning and motion planning independent of each other. If we follow this separation, we can regard each subtask as a motion planning problem. But there are several difficulties with this view. First, the symbolic planner is ignorant of the physical constraints of the environment and can therefore easily lead to non-optimal plans on the motion level. Second, a symbolic planner ignores what happens during the transition from one state to another, which means that the correct next state can be reached, even if the robot is damaged during the transition. Third, the symbolic planner cannot predict or circumvent high risk actions, which can lead to irreversible situations. All three difficulties have a common cause, namely that the abstraction level of the symbolic planner is too high.

We therefore propose to integrate symbolic and motion planning, in order to increase the probability, that a symbolic action is successful. A successful symbolic action will be achieved, if we found a trajectory on the motion planning level, which reaches the desired symbolic state. We will call such a trajectory feasible, and the corresponding symbolic action likewise feasible. The feasibility of a symbolic action is demonstrated in Figure 1.1. A symbolic action  $GraspCylinder(X)$  has to be translated into a robot trajectory. If we face uncertainty, the desired effect  $inhand(X)$  is not deterministic, but will have a certain probability to be feasible. In this example, we have obtained a value of 0.6, which means, that the robot on the right is able to grasp the cylinder in 60 percent of the cases. The goal of this thesis is to increase this probability, in order to end up with symbolic actions, which are more robust against noise. To make this clear, we want to state the central question, which we try to answer in the following chapters:

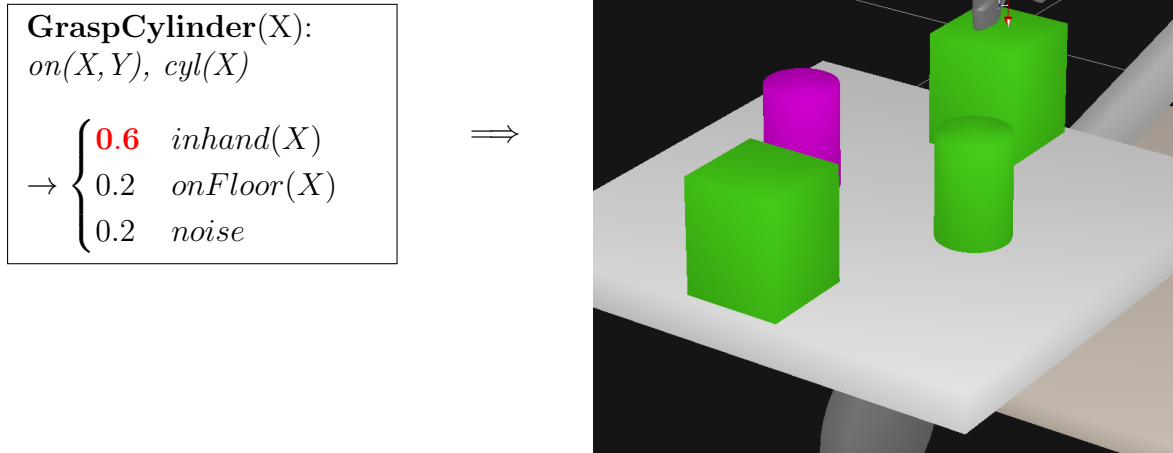


Figure 1.1: The symbolic action *GraspCylinder* has to be translated into a trajectory of the robot. In this example, the transition probability to reach the effect *inhand(X)* is 0.6, which means, that the robot can only grasp the cylinder in 60 percent of the cases.

**How can we increase the transition probability of a desired outcome of a symbolic action?**

## 1.1 Contributions

Our answer to the above stated question is a closer integration of symbolic and motion planning. Our starting point is the observation, that each motion planning algorithm has to be defined by a set of constraints. Those constraints define for example that collisions with objects and joint limits should be avoided. Different symbolic actions need different constraints to be satisfied. For example, if we want to grasp an object, we want the hand of the manipulator to open slowly during the execution. Each symbolic action will have a set of constraints, which we will henceforth call task variables. Given a set of task variables, we can already use a motion planner to find a feasible trajectory, which satisfies all constraints. But if we face uncertainty, it is important to find not only a feasible trajectory, but an optimal trajectory, which finds the best possible trajectory, given the constraints. This helps us to achieve more robust results.

To describe an optimal trajectory, we need to define a cost function, including all task variables. Thereby, we have to consider, that not every variable is equally important. For example, colliding with a wall should be avoided at all costs, while opening the finger before a grasp is less important. We will call these importance variables

*precision parameters*. To specify those parameters becomes difficult, if the number of variables increases. A manual specification is tedious and needs to be done by trial and error.

Our contribution in this thesis will be to automatically optimize the *precision parameters* for each symbolic action. Each set of parameters for one symbolic action constitutes a motion primitive, which defines how the motion planner is constrained. The goal will be to find the *precision parameters* for each symbolic action, which increase the desired transition probability. We will refer to this transition probability in this thesis as *success rate*. To evaluate this *success rate*, we need to compare the *precision parameters* to each other. To be able to do this, we created a fixed set of scenarios, where the robot has to execute the symbolic action. We then measured the *success rate* over all scenarios, together with important physical variables, which characterize the trajectory. We used them to create a scalar value  $\gamma$ , inversely proportional to the *success rate*. This value  $\gamma$  is the output of our objective function, which converts the input *precision parameters* into  $\gamma$ , by using a motion planner and an evaluation of the final trajectory. To optimize this objective function, we used an evolutionary strategy algorithm called *covariance matrix adaption* (CMA).

We were able to show empirically, that the *success rate* indeed increases if we optimize the *precision parameters*, for a given set of scenarios, and that it outperforms manually specified parameters by an expert. Eventually, we conducted an experiment to test, if the optimized parameters generalize to unseen scenarios. We could show, that in random scenarios, the optimized parameters improved the number of feasible plans, in comparison to manually specified ones.

## 1.2 Definitions and Outline

In artificial intelligence research, it is common to refer to any system, which “*can be viewed as perceiving its environment through sensors and acting upon that environment through actuators*”, as an agent (Russell and Norvig, 2010)[p. 34]. Because our work is specifically interested in planning in real world environments, we will name our agent “robot” throughout the thesis.

Tasks like “clean up the room” will be called either a symbolic task or a high level task. Both names are interchangeable and should distinguish the task from its counterpart, i.e. low level motion planning actions, which try to find a *trajectory* of the robot.

The remainder of this thesis will consist of a related work part, where we explain symbolic planning in section 2.1. Afterwards, in section 2.2, we discuss motion planning, by concentrating on optimal motion planning and how task variables are constructed. We proceed in Section 3.1, by discussing current ideas and algorithms, which integrate symbolic and motion planning. This leads to the topic of motion primitives,

which we will discuss in Section 3.2, where we will point out, how the general notion differs from our approach.

After providing the background and discussed the main ideas on integration, we focus on our approach in Chapter IV. Here we will explain in detail, how to evaluate the *precision parameters* by using different mappings and a motion planner. Each mapping is then explained in detail, together with the optimization strategy CMA in Section 4.4. In the end, we show the simulated evaluation of the methods in Chapter V and discuss applicability and future research in Chapter VI.

# Chapter II

## Background in Symbolic and Motion Planning

To provide a sufficient background for integration, we cover the topics of symbolic planning and motion planning in the next two sections. The main focus here is to provide important concepts from both areas, which on the one hand explain how a planning procedure works, and on the other hand provide an intuition, how we can integrate both areas.

We start by introducing symbolic planning. *Symbolic planning* refers to finding a sequence of abstract actions, called a plan, which enables the robot to reach a desired state of the world (Russell and Norvig, 2010). An important aspect for robustness of symbolic planning is the uncertainty of state transition. We discuss its origin in Section 2.1.2 and how it is handled in Section 2.1.3.

After introducing symbolic planning, we explain how a symbolic action can be executed by using a motion planning algorithm. *Motion planning* is concerned with finding a trajectory for a set of continuous variables. In robotics, we are usually interested in finding a trajectory in configuration space (C-space), which is the space of all joint torques. For example, given a 7 degrees-of-freedom robot, the C-space will become a seven dimensional space. The final trajectory is subsequently called a plan, but we will call it *trajectory* to distinguish it from the symbolic plan.

### 2.1 Symbolic Planning with Relational Representations

Giving a comprehensive overview about symbolic planning is beyond the scope of this thesis. Please refer to the basic text from Russell and Norvig (2010), but also to discrete planning from LaValle (2006). Our main focus here is to introduce symbolic planning by discussing a recent symbolic algorithm by Lang and Toussaint (2010a), and discussing probabilistic rules by Pasula et al. (2007) based on relational representations.

This symbolic framework is particularly suited for our motion primitive optimization task (see Chapter IV), because it possess two important properties.

First, it can handle uncertain state transitions, which is important, if we want to execute a symbolic action in a real environment. We will discuss the causes of uncertainty in Section 2.1.2.

Second, its state representations are relational, which is an efficient way to represent a state. We will proceed by discussing relational representations in Section 2.1.1.

### 2.1.1 Relational Representations

A recent planner by Lang (2011) has demonstrated, that symbolic planning can benefit from a relational state representation. Relational representations are based on the notion of objects, its properties and its relation to each other. Lang and Toussaint (2009, 2010b) showed that they are able to outperform enumerated or factored states in a variety of settings. Enumerated states are represented by a single entry for each action and each state, while factored states are represented by a list of state properties (Lang, 2011).

The main reason for the advantage of relational states is the abstractness of the representation, which constitutes another abstraction layer above symbolic representations. The symbolic representation is first created by obtaining symbols from the sensory input. For example, we could create a symbol for a coffee mug and a table, by observing the sensory input and fitting models to it. From the set of symbols, we can create the relational representation, by analyzing the relations between symbols. If we observe, that the coffee mug touches the table with its bottom side, we can conclude, that there is a relation  $on(coffee\_mug, table)$ , which expresses, that the coffee mug stands on the table.

This abstractness of relations comes with the disadvantage, that it is possible that important informations are neglected, which can lead to undesired outcomes. For example, if a coffee mug is located on the edge of a table, it would be risky to grasp it, because the environment is uncertain, and the mug can easily fall of the table.

So far, we discussed the abstractness of relational representations, which improves planning efficiency. But an important question remains: **How are relational representations grounded in the real world?**

To ground a relation in the real world, we need to compute a mapping from input to the relational state. How this is done is still an actively researched question. In their experiments however, Lang (2011) used models of already known objects types to recognize them in the sensory input. Relations are afterwards computed by using so called literals, which are either true or false, depending on the state. A literal  $on(X, Y)$  is for example computed by

$$on(A, B) = \begin{cases} \text{true} & , \text{if } A.X \approx B.X \text{ and } A.Y \approx B.Y \text{ and } A.Z \approx B.Z + B.H \\ \text{false} & , \text{otherwise} \end{cases} \quad (2.1)$$

where  $A$  and  $B$  are two recognized objects from the input stream,  $on(A, B)$  means, that object  $A$  stands on object  $B$ , and  $A.X$  depicts the  $X$  coordinate of the object  $A$  in a coordinate system, where the z-axis corresponds to the height in the real environment.  $B.H$  depicts the height of the object  $B$ , which has to be known to determine if object  $A$  really stands on top of object  $B$ . Given a set of those literals, one can determine a relational state.

This shows, how the symbolic state is acquired. While the symbolic conversion does not influence our devised methods, we will come back to it in section 3.1.3, where we will discuss how a symbolic system can be built up from the configuration space.

## 2.1.2 Uncertainty in Symbolic Representations

Classical algorithms like STRIPS (Fikes and Nilsson, 1971) assume, that the transition between two states is deterministic. But in a real environment, there are several circumstances, where we face non-deterministic transitions. We therefore discuss in this section the possible causes of uncertainty. A more detailed overview about uncertainty in robotics can be found in Thrun et al. (2005).

If a symbolic planner starts in a state  $s_0$  and applies action  $a_0$ , we can reach states, which are undesired. The major cause is that the position of objects can only be approximated. This is caused by several effects. First, we face uncertainty of our sensors, caused for example by a coarse camera resolution or changing brightness. Second, we have unpredictable external influences, like changing air pressure or air draft, which can change the position of the objects or influence the sensors.

Besides the noisy position of objects, we also have uncertainty in the actuation of robots. Due to mechanical limitations, we usually only reach an approximate position of the endeffector. This is further enhanced by noisy sensors, which measure the torque of the robot joints. Together with approximate motion planning solutions, this can lead to an undesired symbolic state transition.

It is therefore necessary to model this uncertainty in the transition between two states. We will discuss this in the following section, by introducing probabilistic state transitions.

## 2.1.3 Probabilistic State Transitions

We have seen in Section 2.1.2, that the environment is usually uncertain and that we have to capture this uncertainty in our state transitions. We therefore introduce a concept called *noisy indeterministic deictic* (NID) rules, which was developed by Pasula

**GraspCube(X,Y)**  
*Precondition:* on(X,Y), cube(X)  
*Effects:* inhand(X),  $\neg$  on(X,Y)

Figure 2.1: STRIPS Rule

**GraspCube(X):** on(X,Y), cube(X)  
 $\rightarrow$   $\begin{cases} 0.7 & \text{inhand}(X), \neg\text{on}(X,Y) \\ 0.2 & \text{onFloor}(X), \neg\text{on}(X,Y) \\ 0.1 & \text{noise} \end{cases}$

Figure 2.2: NID Rule

et al. (2004). *Noisy* refers to the incorporation of noise into the symbolic rule, *indeterministic* describes that one rule can lead to different results in the same situation and *deictic* means, that the rule is defined in relation to other objects, which leads to a compact argument list.

A NID rule represents an action of the robot, together with the probabilities for its outcomes. We can define them formally as

$$\alpha_r(\mathcal{X}) : \phi_r(\mathcal{X}) \rightarrow \begin{cases} p_{r,1} & : \Omega_{r,1}(\mathcal{X}) \\ p_{r,m_r} & : \Omega_{r,m_r}(\mathcal{X}) \\ \vdots & \\ p_{r,0} & : \Omega_{r,0} \end{cases} \quad (2.2)$$

whereby  $\mathcal{X}$  represents the set of logical variables,  $r$  is the NID rule,  $\phi_r$  are the preconditions which have to be fulfilled,  $\alpha_r$  is the action applied on  $\mathcal{X}$ ,  $m_r + 1$  the number of outcomes,  $p_{r,i}$  the probability of each outcome and  $\Omega_{r,i}(\mathcal{X})$  a list of literals which describe which literals will change (Lang, 2011).

A symbolic action to grasp a coffee mug could for example lead to the changed literal *inhand(coffee\_mug)* in the next state, but could also lead to undesired outcomes like *onFloor(coffee\_mug)* or *broken(coffee\_mug)*, depending on the outcome of the action. Practically, these rules are an extended form of the STRIPS representation (Fikes and Nilsson, 1971) with added probabilistic outcomes. The main difference for a simple cube grasping action is depicted in Figure 2.1 and 2.2, respectively. NID rules are particularly useful because they enable a compact representation and can be learned efficiently (Pasula et al., 2007).

NID rules are not the only relational representations using uncertain outcomes. Another popular representation is the extension of the Planning Domain Definition Language (PDDL), called Probabilistic-PDDL (Younes and Littman, 2004). It attempts to define a common standard for planning domains, incorporating uncertain outcomes. We will not attempt to give a complete overview about probabilistic representations. For this thesis, it will be sufficient to assume that the symbolic planner can in fact handle uncertain outcomes.



## 2.1.4 Symbolic Planning with PRADA

If we declare our state and action representation, the final question is how we can come up with a plan from a start state to a desired goal state. This process can be seen as an optimal path search between two states. We here briefly introduce a planning algorithm based on random action sampling, called Probabilistic Relational Action-sampling in DBNs planning Algorithm (PRADA) (Lang and Toussaint, 2010a). Actions and their outcomes are hereby represented as a dynamic bayesian network (DBN), and random paths from the start to the goal state are sampled. The algorithm terminates, if one of the sampled paths has reached a certain goodness criterion.

This criterion is based on the probabilities of action outcomes. Each probability is obtained by executing a symbolic action through a motion planner, and observing the state transition. The goal is to find a plan, which is robust against noise, by using actions which have a high probability to reach a desired next state. We will show in Chapter IV, how one can increase the probability of a desired next state by optimizing the underlying parameters of the motion planner, for each symbolic action individually. This supports the symbolic planner in order to end up with more robust plans.

## 2.1.5 Open problems of symbolic planning

To motivate the approach in this thesis, we will discuss two of the open problems, which are still present in symbolic planning. This will be further discussed in Chapter IV.

### (1) Optimal actions can lead to irreversible outcomes

In a real environment, where we have to cope with noise, we can reach an undesired situation, which we cannot reverse. For example, if we want to grasp a coffee mug, but the grasp was not strong enough, the coffee mug could fall on the floor. Symbolic planners can already incorporate noise directly in their transition model, but it still bears a problem, because the probabilities of specific outcomes cannot be influenced. We will later show, that a closer integration between symbolic and motion planning can improve upon this problem.

### (2) Where does the symbolic representation come from?

Before anything can be planned, one has to prespecify a symbolic representation, which includes a mapping from sensor input to the symbolic state. This mapping is crucial, because we have to specify a symbolic representation, which is neither too abstract or too close to the physical motion level. Too abstract means that we do not incorporate important information, while too close to the physical level means that we suffer the complexity issue of the motion planner. This is still an open issue, and we will discuss this in section 3.1.3, where we will discuss how symbolic representation can emerge directly from the configuration space.

## 2.2 Motion Planning by Inference

To execute a symbolic action, we need to come up with a trajectory for the robot. A trajectory will be defined as a set of joint configurations  $q_{1:T}$ :

$$q_{1:T} : \mathcal{R}^{T \times N} \tag{2.3}$$

whereby  $N$  is the number of joints of the robot, and  $T$  the number of individual configurations, which correspond to different instances of time. The goal is to find a trajectory, which can reach a goal configuration  $q_g$ , which corresponds to the desired symbolic state we want to reach. Given a goal configuration, a mapping  $\Psi_{q \rightarrow s}$  is used to translate the configuration to a symbolic state. This section will focus on the motion planning procedure, and how we can create a trajectory, given a start configuration, a goal configuration and the constraints.

We proceed by introducing the relation between classical motion planning and stochastic optimal control, which motivates the introduction of task variables. Task variables are specific constraints for the trajectory, which we require to be satisfied. Also, we introduce *precision parameters*, which define the importance of each task variable. This will become an important aspect of our approach in Chapter IV. We will also discuss the general idea by introducing a recent planning algorithm, based on inference.

For a broader introduction and further reading on motion planning, we refer to LaValle (2011a,b). The relation between motion planning and stochastic optimal control is introduced in Hespanha (2009). We also make use of factor graphs, which are discussed in detail by Loeliger (2004).

### 2.2.1 From Motion Planning to Stochastic Optimal Control

Motion planning, in its classical definition<sup>1</sup>, is concerned with finding a constraint-free trajectory between two points in a continuous space. In robotics, this space is called the *configuration space* (C-space), where one point corresponds to a configuration of joint torques of the robot. The C-space is divided into two subspaces, first  $\mathcal{C}_{obs}$ , the obstacle space, which represents all invalid configurations of the robot, and second  $\mathcal{C}_{free}$ , representing all valid configurations. The goal is to find a trajectory from start to goal configuration, which lies in  $\mathcal{C}_{free}$ . Such a trajectory is called a feasible path.

To find a feasible path, we first have to define  $\mathcal{C}_{obs}$ .  $\mathcal{C}_{obs}$  is usually constructed by checking all constraints of a configuration. The constraints depend on the task and can include collisions with other objects, joint limits or forbidden areas in task space.

---

<sup>1</sup>The general problem is also called the Piano-Movers problem, see LaValle (2006)[p. 131] for a detailed definition of this problem

We discussed in Section 2.1.2, that the environment is uncertain. Because this also applies to motion planning, it is usually not sufficient to find a feasible path, but to find an optimal path, which is more robust against noise. This is called optimal motion planning, but more prominently treated as *stochastic optimal control* (SOC). In SOC each state transition is treated as uncertain. Given a robot configuration  $q_t$  at time  $t$ , and a control signal  $u_t$ , which describes an additional torque which we apply to the robot, the state equation of the SOC problem is then described by

$$q_{t+1} = f(q_t, u_t) + \xi \quad (2.4)$$

where  $q_t$  is the configuration of the system at time  $t$ ,  $u_t$  the control signal at time  $t$ , and  $\xi$  the noise, which is assumed to be present in the transition (Hespanha, 2009).

Equation (2.4) is the state transition model, i.e. our physical constraint. Our objective is to minimize a cost of all involved constraints. In SOC, this is called the *cost function*. A classical example of the cost function is the quadratic regulator, which defines a quadratic cost of the state and the control. It can be defined as

$$J_{LQR} = \int_0^T \underbrace{q^T Q q}_{\text{output energy}} + \underbrace{u^T R u}_{\text{control signal energy}} dt \quad (2.5)$$

(compare to Hespanha (2009)[p.192ff]), whereby  $J_{LQR}$  is called the cost-to-go,  $u$  is the control vector,  $q$  the configuration vector and we omitted that  $u = u_t$  and  $q = q_t$ .  $R$  and  $Q$  are weight matrices, which determine the costs of each individual dimension.

While the quadratic regulator can be analytically solved by using the ricatti equations (see Hespanha (2009) and Toussaint (2009)), this is not true for the general case. Furthermore, the quadratic regulator does not explicitly include important task objectives, like “do not fall over”, but only cares about minimizing the quadratic values. But what one is really interested in, is an optimal trajectory for a specific task. It is therefore necessary to include different task constraints directly into the cost-function.

## 2.2.2 Casting SOC as an Inference Problem

Instead of using the energy of the system, Toussaint (2009) use different task variables like “reaching goal”, “do not fall over” or “avoid obstacles”. This will cause the cost-function to become non-quadratic, and will therefore be more difficult, i.e. in general not analytically, to solve. In order to solve arbitrary costs-to-go functions, Toussaint (2009) introduced a bayesian view on planning, by acquiring a probabilistic model of trajectories, which are conditioned on all desired task variables.

As described by Rawlik et al. (2010), a belief distribution  $b_t(q_t)$  is calculated for each time step. It represents the belief about the goodness of a certain posture at a specific timestep, including all constraint task variables. Given this distribution, the maximum a posteriori (MAP) configuration is chosen as

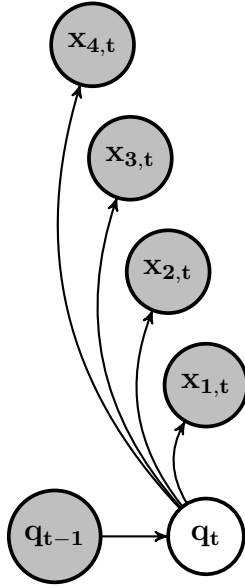


Figure 2.3: Conditional dependencies for the belief calculation.  $q_t$  represents the posture of the robot at time  $t$ ,  $x_{i,t}$  is a task variable at time  $t$ . Taken from Toussaint and Goerick (2010)

$$q_{MAP} = \arg \min_q b_t(q_t) \quad (2.6)$$

The belief is calculated by inference, the dependencies in each timestep are depicted in figure 2.3. Instead of calculating the belief directly, Rawlik et al. (2010) generalize the algorithm by using a concept called factor graphs (Loeliger, 2004). It calculates the belief by using a message passing algorithm in the following form:

$$b_t(q_t) = \mu_{q_{t-1} \rightarrow q_t} \mu_{q_{t+1} \rightarrow q_t} \prod_{i=1}^V \mu_{x_{t,i} \rightarrow q_t} \quad (2.7)$$

This is depicted in Figure 2.4, where the current configuration  $q_t$  is conditioned on the previous configuration  $q_{t-1}$ , the next one  $q_{t+1}$ , and the task variables  $x_{t,1}, \dots, x_{t,V}$ . The individual messages are defined as

$$\mu_{q_{t-1} \rightarrow q_t} = \mathcal{N}[q_t | S_t^{-1} s_t, S_t^{-1}] \quad (2.8)$$

$$\mu_{q_{t+1} \rightarrow q_t} = \mathcal{N}[q_t | V_t^{-1} v_t, V_t^{-1}] \quad (2.9)$$

$$\prod_{i=1}^V \mu_{x_{t,i} \rightarrow q_t} = \mathcal{N}[q_t | r_t, R_t] \quad (2.10)$$

$$(2.11)$$

whereby  $S_t, s_t, V_t, v_t$  parameterize the individual state transitions. See Toussaint and Goerick (2010) for a more detailed overview. The mean  $r_t$  and the covariance  $R_t$

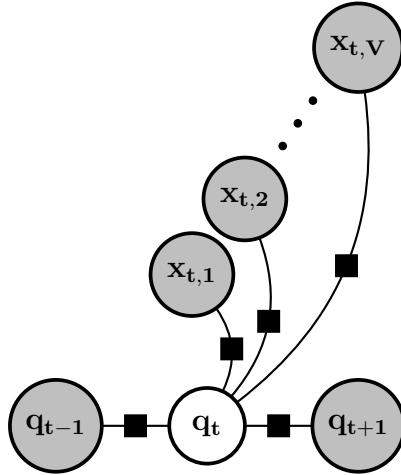


Figure 2.4: Factor graph representation to calculate the belief  $b_t(q_t)$ .  $q$  represents the configuration of the robot,  $x$  is a task variable, for example the position of the endeffector. From Toussaint and Goerick (2010)

of the task messages are defined as

$$r_t = R_t \hat{q} + \sum J_i^T C_{i,t}^{-1} (y_{i,t}^* - \phi_i(\hat{q})) \quad (2.12)$$

$$R_t = \sum J_i^T C_{i,t}^{-1} J_i \quad (2.13)$$

with  $J_i$  the jacobian of a configuration,  $\hat{q}$  the approximated configuration,  $x_{i,t}$  the desired goal of task  $i$  at time  $t$ ,  $\phi_i(\cdot)$  the task mapping for task  $i$  and  $C_{i,t}$  depicts the covariance of task  $i$  at time  $t$ . Here we face the tunable parameters of the algorithm: The task mappings  $\phi_i$ , the desired goal of the task  $y_{i,t}^*$  and the inverse of the covariance matrix  $C_{i,t}^{-1}$ , which defines with which precision a task has to be fulfilled. We will explain each one of them in more detail in the following section, because they will constitute the parameters of our optimization task.

### 2.2.3 Task Variables and Task Targets

A task is a specific constraint, which the motion planner has to satisfy in order to find a feasible trajectory. We represent a task by a task mapping  $\phi_i : q \rightarrow y_i$  (Toussaint, 2009), where  $q$  depicts the robot configuration and  $y_i$  is the task variable. For each mapping, we can define a task target  $y_{i,t}^*$ , which represents our desired goal. An example is a desired trajectory of the robot's endeffector, whereby our task mapping would become the forward kinematics as described in Craig (1989) and the task target would become the individual points on the trajectory.

Equation (2.12) sums up all differences between task variables and task targets and weighs them by the inverse of the covariance matrix. This inverse covariance matrix is called the precision of the task, because it defines how close we want to reach the target. We discuss this precision matrix in detail in the following section.

## 2.2.4 Precision of Task Variables

In Toussaint and Goerick (2010) the covariance matrix is defined as

$$C_{i,t}^{-1} = v_{i,t} \mathbf{I} \quad (2.14)$$

whereby  $i$  is the task,  $t$  the time,  $C_{i,t}^{-1}$  the precision matrix,  $\mathbf{I}$  is the unit matrix and  $v_{i,t}$  the scalar precisions for each task  $i$ . Through the choice of  $v_{i,t}$  we can determine the importance of each task. Note also, that the precision for one specific task can vary over time. For example, if we define a task variable to reach a specific goal, the position of the robot manipulator on its way is not as important as the final goal position. Therefore, we could reduce the precision on its way and define a large precision at the end.

As the number of task variables increases, it gets more complicated to define the precision in the right way. For the task variables of a grasping action, it is important to reach the goal, but also to avoid obstacles, avoid joint limits, obtain a good grasp and align the finger onto the object. The precisions for each task have to be specified manually, by trial and error.

As already discussed in Section 1.1, our main contribution in this thesis will be to develop an algorithm, which can automatically find the optimal precision parameters, which lead to a successful execution on the symbolic level. This extension will be discussed in Chapter IV.

# Chapter III

## Related Work

After we discussed the necessary background for the remainder of this thesis, the following two sections will discuss the related work in the area of combination of symbolic and motion planning, and the area of motion primitives. To begin, we will first discuss, why it is necessary to attempt a combination of both planning levels. We then proceed to discuss different approaches for combination, first the hierarchical top-down approach, which starts with a symbolic planner and breaks its tasks down into smaller subtasks, which eventually can be executed by a motion planner. Second, we discuss, how one can predict if a symbolic action is feasible on the motion planning level, and how this can be used to speed up planning in generally. Third, we discuss a bottom-up approach, where the symbolic world is build up from the underlying configuration space.

After discussing combination, we will proceed by a giving an overview about motion primitives. Our main focus is first to introduce the two prevalent models of primitives, and second to discuss how motion primitives can be incorporated into the motion planning procedure. This incorporation leads directly to the approach used in this thesis, which we will delimit against related approaches.

### 3.1 Combination of Symbolic and Motion Planning

To justify this thesis, it is essential to discuss, why we need to combine symbolic and motion planning. We therefore will point out three arguments for the combination of both planning levels.

**(1) Optimality criterion on symbolic level is different from physical optimality**

On the symbolic level, our definition of optimality is different from the one on the motion planning level. Motion planning accesses directly the physical constraints of the world, like the position of the objects, collisions of the links of the robot, joint limits or the position of the fingers. Contrary, symbolic planning operates on the symbolic states only, and defines optimality by the state transition probabilities, or by

the shortest path between the start state and the goal state. An optimal path on the symbolic level is therefore not necessarily optimal on the motion level.

## **(2) Unable to handle irreversible actions**

The different meanings of optimality can also inadvertently lead to high risk motion plans and therefore to an irreversible outcome. For example, if a coffee mug stands at the edge of the other end of a table, it could be risky to try grasping it directly. But if one goes around the table, the action can become more robust. This cannot be modelled directly by relying on the symbolic planner alone.

## **(3) Symbolic planner analyzes only the start and end configuration**

After the execution of a trajectory, the symbolic planner obtains the next symbolic state by using a mapping from the goal configuration to a symbolic state. If this configuration belongs to the desired symbolic state, the transition is successful. Thereby, any events during the execution of the trajectory are ignored. If the robot collides with an object during the execution, or joint limits are reached, the robot or the objects can be damaged, even if the final symbolic state is reached.

All three arguments have a common cause: The abstraction of the symbolic planner. While the abstraction is necessary to obtain plans in a reasonable amount of time, a too high abstraction can lead to the aforementioned problems. It is therefore necessary to discuss, how we can approach the abstraction by an integration of symbolic planning and motion planning. We therefore proceed by introducing three different ideas of integration. The first section is concerned with algorithms using a top-down approach, which starts on the symbolic domain and uses a motion planner to solve for a corresponding motion plan. One could improve upon symbolic plans, if the feasibility of a motion plan can be guessed. Section 3.1.2 will therefore discuss ideas to predict if a motion plan is feasible. Eventually, in Section 3.1.3, we will discuss a bottom-up approach. The ideas in this section start in C-space and try to find relations to a symbolic description. This can lead to complete and feasible trajectories, but comes at the cost of a high complexity.

### **3.1.1 Hierarchical Top-Down Approach**

The underlying idea of a top-down approach is to start planning in a symbolic representation by using a high level task. This high level task is then divided into smaller subtasks, until we end up with a symbolic action, which can be executed by a motion planning algorithm. Tasks like “clean up the room” can be divided into a sequence of smaller tasks like “clean floor” and “wash coffee mug”. “wash coffee mug” can in turn be divided into smaller tasks like “grasp coffee mug” and “place coffee mug in dishwasher”. If we cannot divide the task into smaller subtasks, we invoke a motion planner to execute a motion plan. Kaelbling and Lozano-Pérez (2011) use this principle to find a feasible trajectory, given a high level task. Instead of planning the complete



trajectory for all subtasks, they execute each subtask directly and use the goal state for the next subtask. If a subtask leads to a wrong state, they try to reverse the action. The main assumption is, that each low level task is feasible and can be executed by a motion planner.

A similar hierarchical approach is proposed by Wolfe et al. (2010). It differs from Kaelbling and Lozano-Pérez (2011) in two points. First, it keeps a record of motion plans for a specific symbolic action, which is similar to the dynamic programming idea. Second, it assumes that a subtask is able to declare, if its trajectory will be feasible. Dornhege et al. (2010) point out, that all approaches still have a fundamental problem: There is no guarantee, that a motion plan for a subtask is feasible.

### 3.1.2 Heuristics to predict Feasibility of Symbolic Actions

Let us assume, that the symbolic planner devised an optimal sequence of symbolic actions. One of them is called “grab coffee mug on the table”. Given this symbolic action, we can ask:

(1) **Can we predict, if the motion plan will be feasible?**

Given only the high level plan, we cannot know the answer. But it is possible to devise heuristics which can indicate if an action is more likely feasible than another. The most prominent approach is to run the motion planning algorithm for a short length of time to collect information about its proceedings. Sucan and Kavraki (2011a) use this to devise a heuristic, involving the dimensionality of an action, the distance to the goal state and the time spent on the plan. This idea was further developed in Sucan and Kavraki (2011b), where many motion planners share information about its progress, to guide the allocation of computational resources.

A similar idea is used by Plaku and Hager (2010). They circumvent unfeasible actions by monitoring the progress of the motion planner. If an action does not make any progress, they stop it at a cut-off time. Thereby, motion plans, which make fast progress towards the goal are preferred.

Bretl et al. (2004) provide a prediction, based on a mathematical prove. If no feasible motion can be found in a short time, they try to prove disconnection between the involved discrete spaces, thereby using insights from computational real algebraic geometry.

Nevertheless, there is no general solution to predict if a motion plan is feasible. As Hauser and Latombe (2009) discussed, only sampling-based methods are really applicable in practical settings. To stop those methods, a cut-off time is usually introduced, which does not guarantee a complete algorithm, i.e. that it can decide if the plan is feasible or unfeasible. Because the motion planning problem is P-SPACE hard

(LaValle, 2006), complete algorithms perform in practical settings far worse than real-time. Therefore, it could be advisable to restrict to approximate complete algorithms, which discretize the space or give probabilistic bounds like Hauser and Latombe (2009).

## (2) Can we guarantee optimality of the motion plan?

Motion planner and symbolic planner have different ideas of optimality. In a symbolic planner, we usually want to find the shortest path in terms of actions or a manually specified heuristic. A motion planner, however, will try to find an optimal way in terms of dynamical, limit or time constraints. This is strongly related to optimal control theory (Hespanha, 2009), where we want to minimize a certain cost function, constrained by our dynamical system. To determine the optimal action involves acquiring information about the whole trajectory from start state to goal state. But using motion planning alone for a whole trajectory is clearly too complex. It seems, that both methods together are necessary to find an approximate optimal action. Nevertheless, it is essential to have a symbolic system, which is near to the configuration space, and still abstract enough too allow for near real-time solutions. The next section will discuss, how such a system can emerge directly from the configuration space.

### 3.1.3 Bottom-Up: Symbolic State from C-Space

Both aforementioned approaches rely on a predefined symbolic planner. But we can also go the other way round, if we devise a mapping from C-space to the discrete, symbolic space (lets call it D-space). Because this D-space is then built from the C-space up, we can be sure that each action is feasible. If it was not feasible, we could not reach it from our current start state.

This is visualized in figure 3.1, where the space is divided into configurations belonging to a symbolic state  $s_0$  and to  $s_1$ , respectively. By sampling the space, we can directly obtain all the symbolic states, which are reachable from the current configuration. If we know about all reachable symbolic states, we can forward this information to the symbolic planner, which can now ignore all non-reachable symbolic states during its planning procedure. It seems that this is a more natural approach, because we only need to define a mapping  $f : C \rightarrow D$ , which maps a configuration to a corresponding symbolic or discrete space. The symbolic action, which emerges, is then guaranteed to be feasible.

Following this idea, Choi and Amir (2009) build a planner, which first samples the C-space to obtain applicable symbolic actions. Each action is afterwards used in a symbolic planner to obtain an optimal symbolic plan. The underlying motion planner will then be used to find a feasible trajectory, which is now guaranteed to be feasible, because we sampled the set of applicable symbolic actions. The disadvantage of this approach is the additional calculation to acquire the nearby symbolic states and to compute the applicable actions.

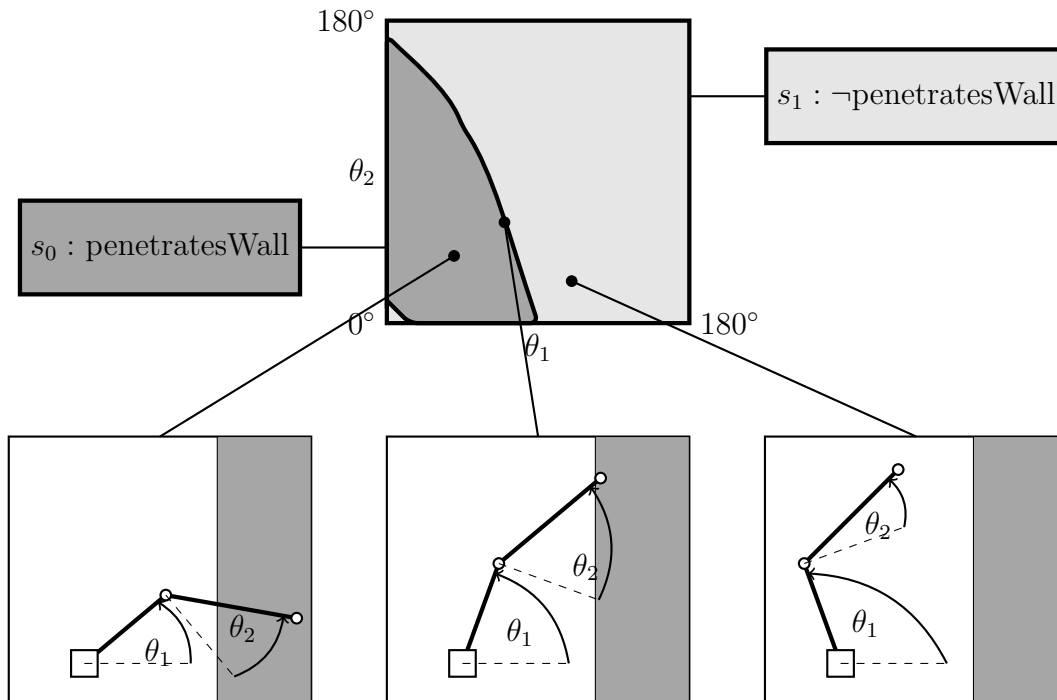


Figure 3.1: The configuration (C-)space (top) and three different corresponding robot configurations. By observing the symbolic state of each configuration, one could sample the C-space and devise a map of symbolic states. This is depicted in the C-space, where one set of configurations belongs to the symbolic state  $s_0$ , while the other belongs to  $s_1$ .

Cambon et al. (2009) extend all symbolic actions by incorporating its associated configuration spaces. This C-space is sampled during each motion planning call. Because each action has an associated C-space, it is now possible to check if two states have a common C-subspace, which can be used as a state transition. Nevertheless, they thereby assume that the configuration space of an action does not change over time.

The type of symbolic representation is of course a crucial decision: Fainekos et al. (2009) use an advanced temporal logic, which combines logic operators and C-subspaces. A reach operation could be described as  $\neg(c_1, \dots, c_n) \text{ UNTIL } c_{n+1}$ , which means: avoid all subspaces  $c_1, \dots, c_n$  until subspace  $c_{n+1}$  is reached. Using this logic is a compromise between the more abstract relational representations and pure motion planning and could be useful for an intermediate representation. On the downside, this approach works currently only in a 2D environment, and the complexity to find the C-subspaces is too high to be useful for real-time planning.

Belta et al. (2007) sample the C-space to obtain a set of motion primitives. To acquire a plan, they use a finite state machine, called a maneuver automaton, which finds a sequence of primitives to accomplish a symbolic task. Nevertheless, there is no

guarantee that they can find a feasible plan by a sequence of primitives.

The introduced ideas in this section all sample the C-space, in order to build up a symbolic space, or to find motion primitives. The remaining question is how we can come up with a valid feasible plan, given a set of motion primitives. The next section will therefore explain motion primitives in detail, and subsequently discuss, how we can incorporate them into a motion planning algorithm to yield a feasible trajectory.

## 3.2 Motion Primitives

The primary idea behind a motor primitive is the wish for abstractness. We want to obtain a set of motions, like “throwing a ball”, “open a door”, “sit on a chair”, which can be used in a higher level plan. Therefore, we need a formalism, which captures the essential properties of a motion. Such a formalism should be “*compact, robust against noise, easy to reuse and it should allow us to recognize the motion*” (Ijspeert et al., 2002b).

Research in motion primitives mainly focuses on two different paradigms for describing a motion, which we will briefly discuss. The first are *attractor based models*, which describe a controller function, which attracts the robot towards its goal. The controller is especially designed to capture the desired motion. Second, a motion can be represented by a set of intermediate milestones. Those milestones are called *via points*.

Motion primitives are generally learned from a set of observed trajectories, either acquired from previously executed motions, or from humans by using imitation learning. It is also possible to learn from previously planned trajectories.

### 3.2.1 Attractor Based Models

Attractor based models follow the principle of designing a controller for a specific motion, which can be used to attract the manipulator towards a desired goal configuration. Attraction was first defined by Ijspeert et al. (2002b), which used nonlinear oscillators as controlling functions. Each controller is shaped by adding a gaussian kernel to the output, and learning its weights from previous trajectories (Ijspeert et al., 2002a). This model can incorporate also perceptual variables (Ijspeert et al., 2002b) and can be used to define rhythmic goal movements, like a waving hand.

Building on oscillators, Kober et al. (2010) simplified the original controller and showed its applicability towards moving targets, like batting and hitting. Learning of the controller can be done by reinforcement learning techniques (Kober and Peters, 2009). This approach leads to two interesting ideas, which we want to discuss here:

#### (1) Supervisor chooses set of primitives

Peters et al. (2009) describe a complete framework for learning and executing a task. They propose a supervisor module, which basically chooses a set of primitives for a specific action. Our framework, discussed in section IV will use a symbolic planning system instead of a supervisor module to choose the primitives. It is interesting to note, that both constitute a different layer of abstractness. While symbolic states plan in a more abstract level, the supervisor module plans in a lower abstractness level, where it tries to combine different primitives by manipulating their duration, their amplitude or their goal. As we will discuss in section VI, the right level of abstractness is crucial to trade-off complexity and efficient algorithms.

## (2) Optimizing on different abstraction levels

Kober and Peters (2011) introduced an optimizing process on different abstraction levels. While optimizing a high level task, their algorithm tries at the same time to optimize lower level motor task, which do not directly influence the high level task. We think it constitutes another example, that robots can benefit greatly from a closer integration of high level tasks and their lower motor tasks.

### 3.2.2 Via Point Models

Instead of designing a controller, one could represent a motion by a set of points. This approach is also called *via points*, because the robot has to pass each point on its way towards the goal. To generate a trajectory, spline-based interpolation could be used to connect the points. Toussaint et al. (2007) use this approach, but instead of using spline methods, they design a controller for each *via point*, which constitutes an intermediate model between attractor based approaches and via points.

Hauser et al. (2006) use via points to better incorporate them into a motion planning algorithm. Contrary to attractor based model, via points can directly be checked for feasibility and therefore provide an easier integration into the planning process.

*Via points* and *attractor based* models should show, how one can define a motion explicitly. Contrary, our approach will define a motion primitive implicitly, by defining constraints of the symbolic action. Still, we need to come up with a feasible trajectory, independent of the model. The following section will discuss how this can be accomplished.

### 3.2.3 From Motion Primitives to Motion Planning

A major question is how motion primitives can be used to accelerate or replace motion planning. A mayor problem is that motion primitives are generally seen as special kind of controller, like a PID one. Like any other controller, they suffer from local optima, which can appear for example, if objects are blocking the motion path. Besides local optima, we also have to think about, which motion primitive we should use at which time. Our intention is to devise an algorithm, which can decide to use the right motion primitive, while also avoiding local optima. We will therefore review existing literature for their contribution to the following questions:

(A) **How can we avoid local optima?**

(B) **Which motor primitive should be used at which time?**

Koenig (2010) approaches (A), by starting a planner, once a local optimum is reached. They call this a Planning-on-demand system, where the robot uses a potential function to reach the goal, but starts a motion planning algorithm, if it is stuck in

a local optima. The same could in principle be done with a motion primitive controller.

Park et al. (2008) embed the perturbances generated by objects directly into the controller by using dynamical potential fields. Local optima are assumed to appear seldom and only when many objects appear to block the path. Because they concentrated on learning primitives from imitations, they conducted experiments with one primitive and did not approach our question (B).

Cohen et al. (2010) first acquire a set of basic motion primitives. To solve (B), they conduct a search for a series of primitives, which lead towards the goal. This was extended by incorporating lower-dimensional primitives and by acquiring new primitives during the planning process by Cohen et al. (2011). Note that this seems to be a more local version of the symbolic vs. motion planning integration. Instead of planning a series of symbolic actions, which are assumed to be feasible on the motor level, they plan a series of motion primitives, which are also assumed to be feasible. But we already discussed this problem in Section 3.1. Nevertheless, this approach constitutes an abstraction of the original problem, and could be advantageous, if it is closer integrated into the motion planner.

A quite similar idea to solve (B) was used by Meier et al. (2011). They build a motion primitive library from newly observed motions and try to fit them to other motions. They do not directly plan with this library, but use motions to fit newly observed motions to acquire a classification skill.

Both Cohen et al. (2010) and Meier et al. (2011) use an unchanged version of their primitives, which constrains the reachable spaces. In contrast, Hauser et al. (2006) transform previously generated primitives, so that they fit the start and goal state. Afterwards, equally spaced points on the path are checked on feasibility. Non feasible points are expanded by a sampling based motion planner until a nearby feasible point is found. The final path will have close resemblance to the original motion, while avoiding non feasible regions. This integration solves (A), but still gives no answer to (B).

In general, we think, that the incorporation of motion primitives into the motion planning algorithm is more promising, because replanning from local optima could be contra productive to the intended motion. For example, imagine that we learned a motion to hit a ball with a racket. Now we want to hit the ball, but a table is in our way. If we just execute our controller, we could end up under the table, before we realize that we need to replan. The ball would already be gone. If we invest time to plan at the start, we could improve the controller in a way, that we would hit the ball above the table. Of course, the time to plan is crucial, but in a changing environment, it seems to be necessary to consider this option.

Our own approach follows therefore the incorporation of motion primitives into the planning algorithm, but is contrary to Hauser et al. (2006), because we consider a

symbolic planner as part of the framework to approach (B), and because we do not directly include a motion trajectory into the motion planner. Instead, we shape the parameters of the motion planner directly, depending on the symbolic action. We think that this constitutes a more integrated approach, because we change the search space according to the motion primitive, in contrast to fitting a motion trajectory directly into the C-space. This concept will be part of the discussion about our approach in the following section.



# Chapter IV

## Integrating Symbolic and Motion Planning

We have seen in Section 2.1 how a symbolic planner can calculate a sequence of actions, in order to solve a high level task. To execute one of the symbolic actions, we introduced motion planning in Section 2.2, and discussed its relation to stochastic optimal control, together with the constraints which constitute the cost-to-go function. After executing a symbolic action, we will end up in another symbolic state. Because of uncertainty, this state transition will be stochastic, as depicted in Figure 4.1. The figure shows that if we apply an action  $a_0$  in the start state  $s_0$ , we will reach the next state  $s_1$  with a probability of  $p_{0 \rightarrow 1}$ . Usually, we have a preference, which next state should be reached. For example, if we execute a symbolic action *grasp*  $X$ , our desired effects on the next state would be *inhand*( $X$ ). This desired state transition has a probability to succeed. Our goal is to increase this probability, in order to achieve more robust and feasible actions. The general principle is demonstrated in Figure 4.2. On the left, a NID rule for the symbolic action *GraspCube*( $X$ ) is shown, together with three different outcomes. The desired goal is to reach a next state, where the effect *inhand*( $X$ ) holds true. Our approach will optimize the underlying motion planner for each symbolic action, so that this transition probability increases. After our optimization process, we want to obtain a changed state transition model, as shown on the right of Figure 4.2, where the desired probability significantly increases.

Before formalizing our approach, we will briefly discuss the optimization procedure from a conceptual level. We discussed in Section 2.2, that a motion planning task has to be constructed by a set of constraints, which were called task variables. We also specified the importance of each variable by so called *precision parameters*, which define how close the task variable has to be satisfied. In our approach, we will use a static set of task variables, which are predefined and remain unchanged, together with a dynamic set of *precision parameters*, which we want to optimize. By optimizing the *precision parameters*, we can identify which task variables are important for a specific symbolic action, in order to increase the desired state transition probability.

This probability can be computed, by executing a symbolic action repeatedly from

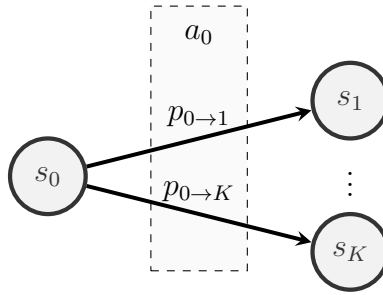


Figure 4.1: Stochastic state transition model: If we apply an action  $a_0$  in state  $s_0$ , we will reach the next states  $s_0, \dots, s_K$  with a certain probability  $p_{0 \rightarrow 1}, \dots, p_{0 \rightarrow K}$ , because of uncertainty in the state transition.



Figure 4.2: Left: A symbolic NID-rule to grasp a cube, which has preconditions  $on(X, Y)$  and  $cube(X)$ . Without optimization, the success rate for the desired next state is 0.6, which means that in 60 percent of the action execution, we will get into state  $inhand(X)$ . After applying our optimization process, our goal is to raise the success rate to a higher value (This is an example with imaginary values, which should demonstrate the principle behind our optimization).

a start state and invoking a mapping from configuration to symbolic space, which translates the goal configuration of the robot to a symbolic state. If we reach a symbolic state, which is equal to our desired state, we increase a counter. The probability is then obtained by repeating the execution in different scenarios, and divide the counter by the total number of executions. We will call this probability *success rate*, to emphasize our objective to increase its value.

## 4.1 Formalization of a Motion Primitive $\mathcal{P}$

We visualized in Figure 4.3, how several tasks for a symbolic action *grasping* are used as the input to a motion planner. Each task is defined by a mapping  $\phi_i$  from joint configuration  $q \in \mathcal{R}^N$  to a task variable  $y_i \in \mathcal{R}^{D_i}$ :

$$\phi_i : \mathcal{R}^N \rightarrow \mathcal{R}^{D_i} \quad (4.1)$$

whereby  $N$  is the number of joints of the robot, and  $D_i$  the space of the task variable. For example, the mapping  $\phi_i$  can be the forward kinematics (Craig, 1989), if we

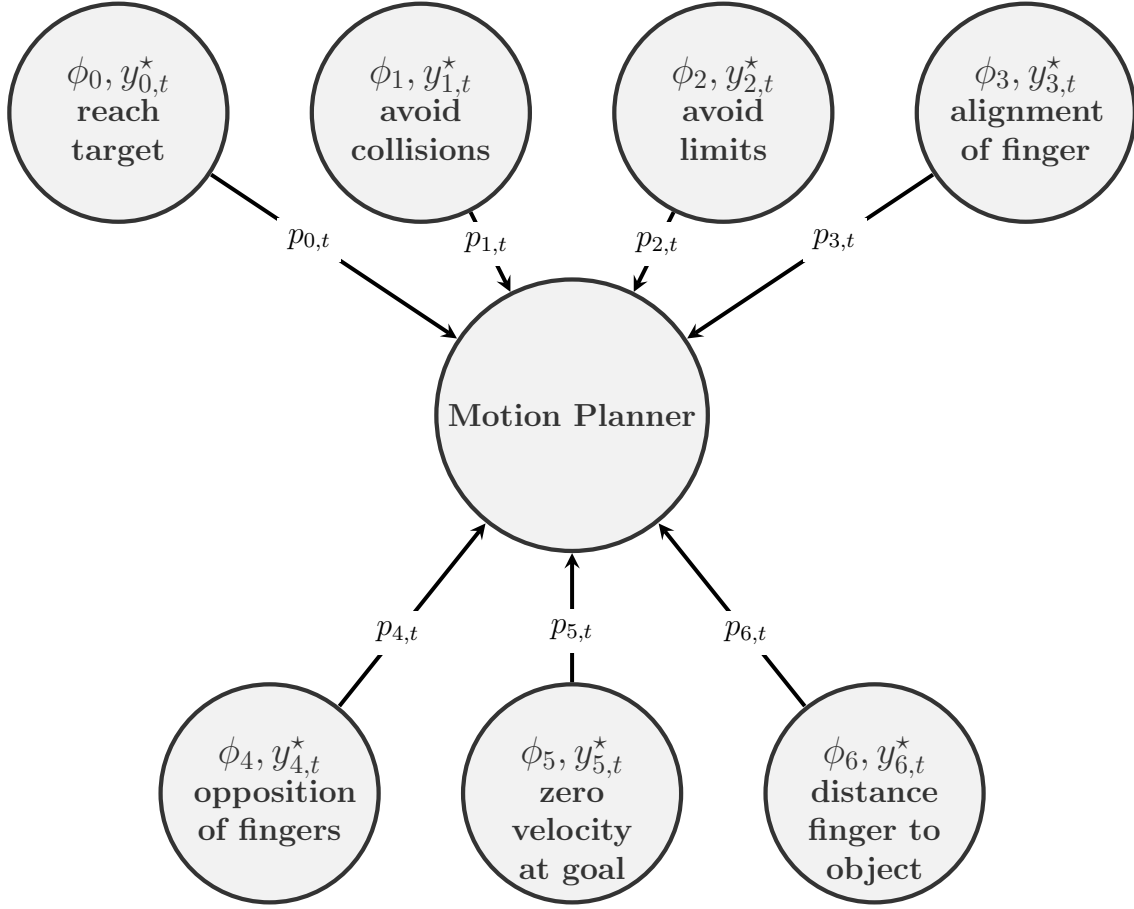


Figure 4.3: Symbolic *grasping* action: A set of tasks define the constraints of the motion planner. Each task is represented by a task mapping  $\phi$  and a task target  $y^*$ , together with the precision parameters  $p$ . See text for clarifications.

want to control the endeffector position. The space of the task variable would become  $D_i = 3$  (position) or  $D_i = 6$  (position and rotation) for the endeffector. Moreover, each task is accompanied by a task target  $y_{i,t}^*$ , which describes the goal of the task at each instance of time  $t$ . In the case of the task *reach target*, the task target is the goal position of the endeffector. Eventually, every variable is weighted by a parameter  $p_{i,t}$ , which defines its precision at each instant of time  $t$ . For the *reach target* task, we would define a low precision during the trajectory, but a large one at the end of it.

Given these tasks, our goal is obtain a formalization which allows us to optimize the *precision parameters* for one symbolic action. We start by joining all task mappings, including the task targets, into a new variable  $\Phi$ :

$$\Phi = \{\phi_1, y_{1,t}^*, \dots, \phi_V, y_{V,t}^*\} \quad (4.2)$$

whereby  $V$  is the number of tasks. The *precision parameters* are joined together by a new variable  $\alpha$ :

$$\alpha : \mathcal{R}^{P \times T} \quad (4.3)$$

$$\alpha = \{p_{1,1}, \dots, p_{P,T}\} \quad (4.4)$$

whereby  $P$  is the number of task variables and  $T$  the number of time steps.

The variables  $\Phi$  and  $\alpha$  define the constraints and the importance of a symbolic action. We will define the concatenation of  $\Phi$  and  $\alpha$  as a motion primitive  $\mathcal{P}$

$$\mathcal{P} = \{\alpha, \Phi\} \quad (4.5)$$

The motion primitive  $\mathcal{P}$ , together with the start configuration and the scenario of the environment, defines the input parameters for the underlying motion planner. Here we want to point out, how the optimization of motion primitives differs fundamentally from classical approaches. We depicted a sequence of symbolic actions in Figure 4.4. Each symbolic action will be executed by a set of constraints on the motion planning level, which we called a motion primitives  $\mathcal{P}$ . A classical separation of planning algorithms would assume, that a motion planner can find a feasible path, independent of the symbolic action. Each action belongs therefore to the same  $\mathcal{P}$ . Using a motion planner, as discussed in Section 2.2, we would define a different  $\mathcal{P}$  for each symbolic action. But these  $\mathcal{P}$  are specified manually by an expert, and are not optimal. The main focus therefore will be to find a set of  $\mathcal{P}^*$ , which are approximate optimal.

## 4.2 Objective Function from $\mathcal{P}$

To find an approximate optimal  $\mathcal{P}^*$ , we assume that  $\Phi$  is static, while  $\alpha$  is dynamic and can be used to optimize  $\mathcal{P}$ . We therefore will use only  $\alpha$  to further define our optimization. Our first goal is to define an objective function, which maps the  $\alpha$  vector to a scalar value. We represent this by a mapping

$$\Psi_{\alpha \rightarrow \gamma} : \mathcal{R}^{P \times T} \quad (4.6)$$

This mapping constitutes the evaluation of one set of  $\alpha$  parameters into one scalar value  $\gamma$ .  $\gamma$  will represent how successful the symbolic action could be executed. Its value is proportional to the inverse of the *success rate*, but also includes other physical variables, to guide the optimization process. We will proceed by explaining the mapping first on a conceptual level, and in the following sections in detail.

The main idea is summarized in Figure 4.5. Given a vector  $\alpha$  of precision parameters, we evaluate them on  $K = e_0, \dots, e_K$  different scenarios, and obtain a set of  $K$  different trajectories  $q_{e_i}$ , by using a mapping  $\Psi_{\alpha \rightarrow q}$ . This mapping constitutes the motion planner, whereby the start configuration, the goal configuration, the task

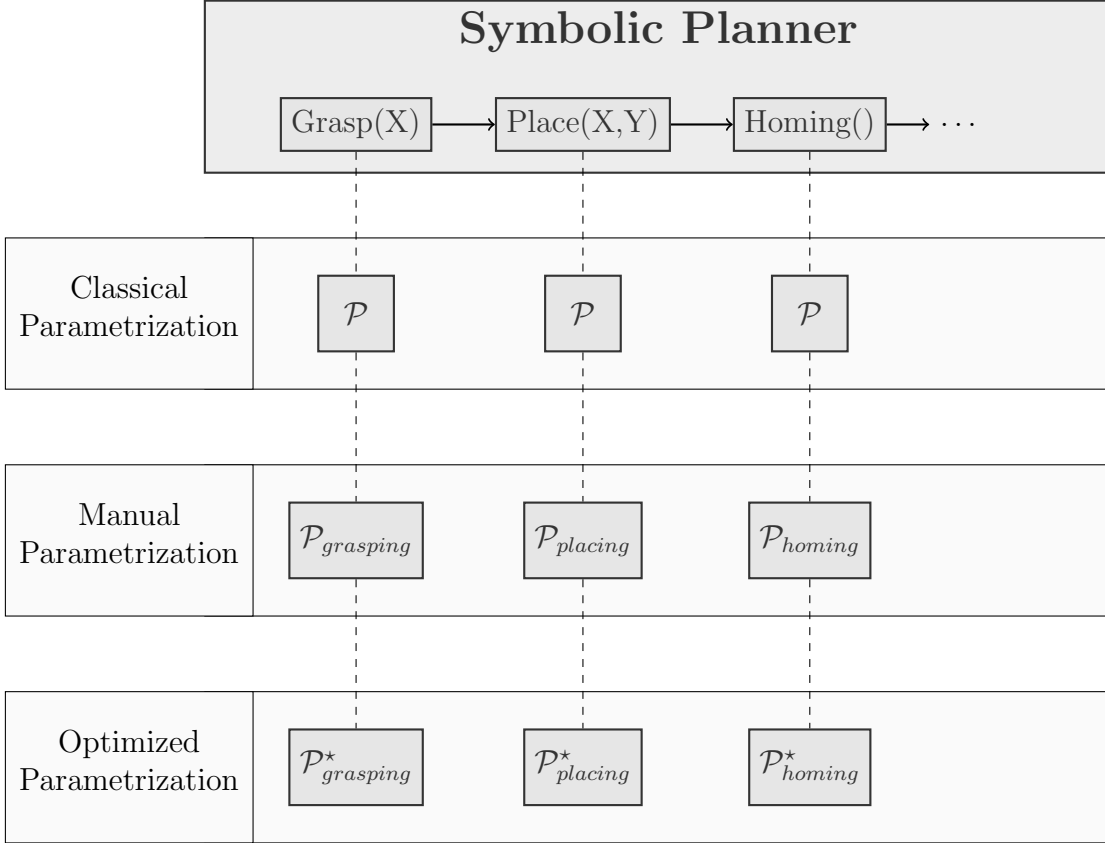


Figure 4.4: Symbolic Planner

target, the task mapping and the scenario remain unchanged. Each trajectory is afterwards characterized by a set of variables  $\beta_{e_i}$ , by invoking a mapping  $\Psi_{q \rightarrow \beta}$ .  $\beta_{e_i}$  includes the *success rate*, but also other important physical variables like maximum collision between robot and objects in the scene. Given a set of  $K$  times  $\beta_{e_i}$  variables, obtained from  $K$  scenarios, we invoke a mapping  $\Psi_{\beta^K \rightarrow \beta}$ , which maps all  $\beta_{e_i}$  variables onto a single  $\beta$ . This single  $\beta$  variable is then evaluated by a mapping  $\Psi_{\beta \rightarrow \gamma}$ , which returns a scalar value  $\gamma$ . We will explain each mapping in detail in the following section.

### 4.3 Mappings to evaluate $\alpha(\mathcal{P})$

After giving a conceptual overview about the evaluation of a vector  $\alpha$  onto one scalar value  $\gamma$ , we will proceed by investigating each mapping individually. We start by defining the first mapping from  $\alpha$  to a trajectory  $q$  as

$$\Psi_{\alpha \rightarrow q} : \mathcal{R}^P \rightarrow \mathcal{R}^{N \times T} \quad (4.7)$$

whereby  $N$  is the number of joints of the robot and  $T$  the time steps of the planned trajectory  $q$ . This mapping represents the motion planner, where the scenario of the

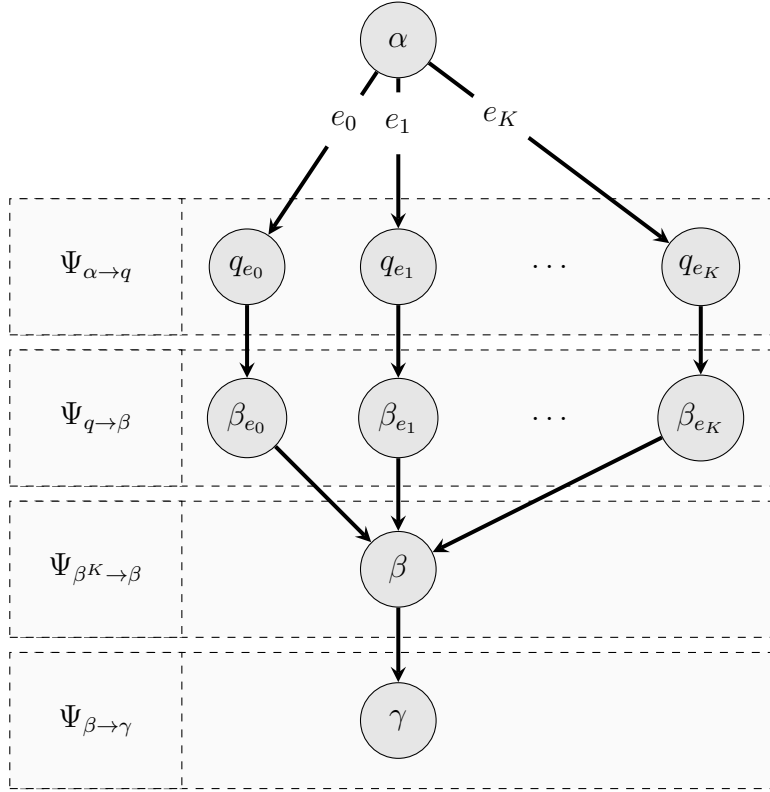


Figure 4.5: Mapping from parameters  $\alpha$  to the scalar value  $\gamma$ , by using a set of  $K$  scenarios.  $e_0, e_1, \dots, e_K$  refer to the scenario, which will be evaluated.

environment, the task target, the task mapping, and the start configuration are previously defined and remain constant.

After obtaining a trajectory from the motion planner mapping, we are interested if we have reached our desired symbolic state. While a separation of symbolic and motion planning would consider the trajectory of the state transition as a motion planning problem and use only the final configuration to compute the next state, we also want to investigate the performance of the trajectory, in order to obtain the next symbolic state. Our main motivation is, that movements along the trajectory could damage the robot or change the environment, even if we reach the desired goal configuration. Therefore, it is essential to obtain a set of variables, which let us describe the trajectory and allows us to compute the next state. We call them  $\beta$  and include the following variables, which can be obtained from a planned trajectory:

- d** - Final distance to the target goal
- c** - Maximum collision distance from all links of the robot to objects
- m** - Maximum distance of each joint of the robot to its limit
- l** - Length of the final trajectory

- i** - Time to plan the trajectory
- g** - Grasp evaluation (only symbolic grasping action)
- s** - Success of symbolic state transition

whereby the success of a symbolic state is directly dependent on the distance to the goal and the grasp evaluation, but also on the collision and limits along the trajectory. The computation of the variables from the trajectory will be done by a mapping:

$$\Psi_{q \rightarrow \beta} : \mathcal{R}^{N \times T} \rightarrow \mathcal{R}^B \quad (4.8)$$

whereby  $N$  the number of robot joints,  $T$  the time steps of the trajectory and  $B$  the number of variables of  $\beta$ , which can vary, depending on the symbolic action. How we compute the mapping  $\Psi_{q \rightarrow \beta}$  for a specific symbolic action will be shown in Section 4.3.1.

Our primary objective, the success of a symbolic state transition, will be a binary variable, which is either true if we reach our desired symbolic state, without violating constraints, or false otherwise. But usually there are several trajectories, which will lead to a success. Because we want to compare trajectories with each other, we will introduce another mapping from  $\beta$  to a scalar  $\gamma$ , which depicts the costs of a trajectory

$$\Psi_{\beta \rightarrow \gamma} : \mathcal{R}^B \rightarrow \mathcal{R}^1 \quad (4.9)$$

whereby  $B$  is the number of variables in  $\beta$  and  $\gamma$  a scalar value. The calculation of  $\gamma$  also depends on the symbolic action and will be explained in detail in Section 4.3.2. Moreover, the values of  $\beta$  are obtained from one trajectory only and in one scenario. But to generalize to unknown scenarios, it is essential to evaluate the parameters on different scenarios. We therefore use a mapping from a multitude of  $\beta^K$  vectors to a single  $\beta$  vector.

$$\Psi_{\beta^K \rightarrow \beta} : \mathcal{R}^{B \times K} \rightarrow \mathcal{R}^B \quad (4.10)$$

whereby  $K$  is the number of scenarios we use, and  $B$  the number of variables of  $\beta$ . We used in this mapping the average over all individual  $\beta_i$  vectors:

$$\Psi_{\beta^K \rightarrow \beta} = \frac{1}{K} \sum_{i=1}^K \beta_i \quad (4.11)$$

The mappings from Equation 4.10, Equation 4.7, Equation 4.8 and Equation 4.9 make up the evaluation of the parameter vector  $\alpha$  to the scalar value  $\gamma$ . To summarize, we end up with the final mapping as

$$\Psi_{\alpha \rightarrow \gamma}(\alpha) = \Psi_{\beta \rightarrow \gamma}(\Psi_{\beta^K \rightarrow \beta}(\{\Psi_{q \rightarrow \beta}(\Psi_{\alpha \rightarrow q}(\alpha))\}^K)) \quad (4.12)$$

We proceed by explaining the two mappings  $\Psi_{q \rightarrow \beta}$  and  $\Psi_{\beta \rightarrow \gamma}$  in detail.

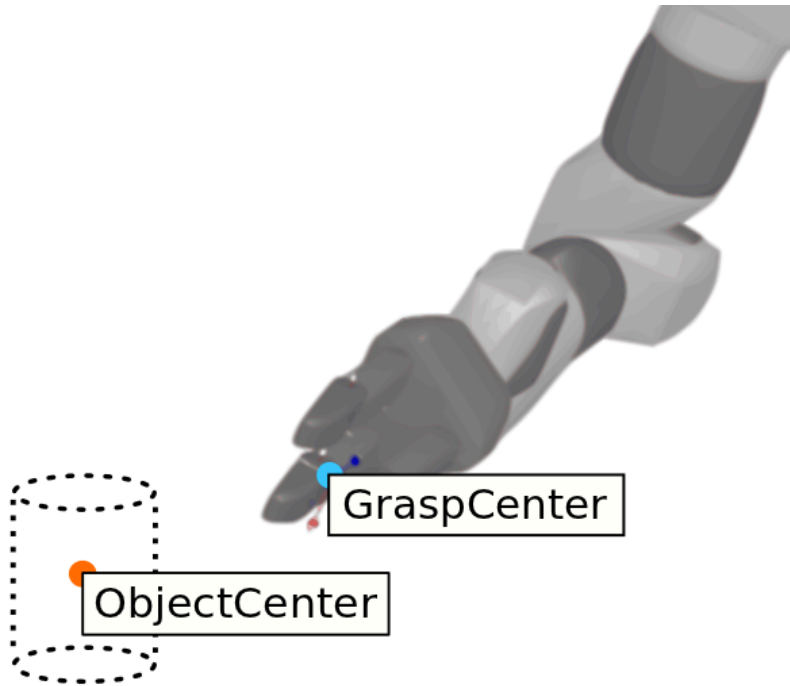


Figure 4.6: graspcenter is a imaginary point in the middle of the robot hand. objectCenter is defined as the center of the object which we want to grasp.

### 4.3.1 $\Psi_{q \rightarrow \beta}$

The mapping from trajectory  $q$  to a set of  $\beta$  variables is used to describe the trajectory in terms of its physical appearance. We already discussed in the previous section, which variables constitute  $\beta$ . Here we will explain each variable in detail, starting with the distance variable.

- **Distance to object**

The distance  $d$  is defined as the l2-norm of the distance between the grasp center and the object we want to grasp. Figure 4.6 shows where the two points are located in space. We end up with

$$d = \|\text{graspCenter} - \text{objectCenter}\|_2 \quad (4.13)$$

We also define this variable as being successful, if a threshold  $\theta_d$  is reached.

- **Collisions**

To acquire a measurement of collisions, we introduce a variable  $c_{i,t}$ , which is a function of the distance between a pair  $i$  of a link on the robot and an object in the



scene. If the link touches or penetrates the objects, we require that  $c_{i,t} \geq 1$ , which leads to

$$c_{i,t} = \begin{cases} [0, 1[ & , \text{if no collision} \\ [1, \infty[ & , \text{if in collision} \end{cases} \quad (4.14)$$

Given a trajectory with  $T$  points, we compute a maximum collision  $c_i$  by using the maximum  $c_{i,t}$  of all points:

$$c_i = \max_{c_{i,1:T}} \quad c_i \in 1, \dots, M \quad (4.15)$$

whereby  $M$  depicts the number of pairs of links and objects. Given the maximum collision  $c_i$  for each pair, we define an overall collision variable  $c$  as

$$c = \max_{c_1, \dots, c_N} \quad (4.16)$$

Given  $c$ , we define a threshold  $\theta_c$ , which depicts, that an action is successful, if we encounter no collided pair along the trajectory, together with a small margin to avoid very close encounters, which can lead to collisions, if we regard the uncertainty of the environment.

- **Limits**

The limits variable  $m_{i,t}$  is defined for each link  $i$  of the robot at each instance of time  $t$  along the trajectory. It depicts if the link  $i$  has reached a joint limit, which is defined by internal hardware specifications. We define it as zero, if no limit specification was violated and greater as zero, otherwise. Similiar to the collision, we use the maximum value over all links and all time instances. This leads to the following formalization

$$m_{i,t} = \begin{cases} 0 & , \text{if no joint limits reached} \\ ]0, \infty[ & , \text{if joint limit reached} \end{cases} \quad (4.17)$$

$$m_i = \max_{m_{i,1:T}} \quad i \in 1, \dots, N \quad (4.18)$$

$$m = \max m_i \quad (4.19)$$

whereby  $T$  is the number of time instances along the trajectory,  $N$  the number of links of the robot and  $i$  is a specific link. We define a successful action, if the limit variable is below a threshold of  $\theta_m$ .

- **Grasp Evaluation**

To evaluate the goodness of the grasp, we copied the hand and the object from the simulator to a free space, where we stopped the number of time steps, after the object falls from the hand. One timestep is defined as one step of our physical simulator, and equals roughly  $10ms$ . After 100 time steps, we start a jiggling motion, that means a

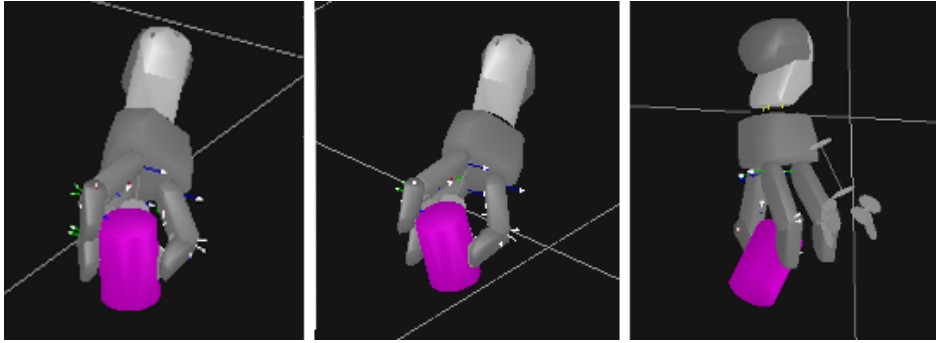


Figure 4.7: Grasp evaluation, after 100 time steps, the robot hand moves on a prespecified trajectory, until the object falls from the hand, or 1000 time steps are reached.

predefined rotation of the hand. We stop, if the object increases its distance from its original relative position to the hand, or if 1000 time steps have elapsed. This motion is indicated in Figure 4.7, where we have shown the start position (left), an intermediate position during the jiggling motion, and the final time step, where the cylinder slides from its original position downwards. The more time steps it takes, until the cylinder leaves the hand, the more we can be sure that the grasp is robust.

- **Length**

To calculate the length of the trajectory, we simply take all the planned points on the trajectory, and sum up the l2-norms of the configurations between them. The point is, that we want to prefer shorter trajectories. Because the length varies between scenarios, we cannot define a successful length, but we will incorporate it as an additional variable.

- **Iterations**

If a plan is unfeasible, the motion planner needs a longer time, because all possible configurations have to be investigated. The standard approach in motion planning is to use a cut-off time, at which we want to stop planning and declare the motion planning task as unfeasible. Similiar to the cut-off time, we define a maximum number of iterations, at which we stop planning, if the planner has not converged yet. The number of iterations  $i$  will be proportional to the planning time. We define that an action was successfully planned, if  $i$  is below a threshold of  $\theta_i$  maximum iterations.

- **Success rate**

The most important variable is the success rate, which is defined for each evaluation individually, and becomes one if certain requirements are fulfilled, and zero if not. We discussed in Section 3.1, that in symbolic planning, only the start and the end state are considered, but not what happens in between. If the robot bumps against an object, this will be hidden from the symbolic level, in the case that the bumping does not change a symbolic literal in the goal configuration. But the robot itself or the object can be damaged, and we will therefore refrain from calling an action successful, if any

constraints are violated on the way.

Therefore, we define a success, if no constraints were violated on the final trajectory:

$$s = \begin{cases} 1 & \text{if } c < \theta_c \text{ and } m < \theta_m \text{ and } d < \theta_d \text{ and } i < \theta_i \text{ (and } g < \theta_g) \\ 0 & \text{otherwise} \end{cases} \quad (4.20)$$

$$(4.21)$$

whereby  $\theta_c, \theta_m, \theta_d, \theta_i$  and  $\theta_g$  are the thresholds for collision, limit, distance, iterations and grasp evaluation, respectively, as defined above. The grasp evaluation is set in parenthesis, because it is only considered, if we evaluate the success of a symbolic grasping action. The final  $\beta$  parameters are then defined by concatenating all variables together

$$\beta = \{d, c, m, l, i, g, s\} \quad (4.22)$$

### 4.3.2 $\Psi_{\beta \rightarrow \gamma}$

The previously defined variables from the mapping  $\Psi_{q \rightarrow \beta}$  measures the physical outcome of the trajectory. After obtaining  $\beta$ , the next step is to compute a scalar value  $\gamma$ , which represents the goodness of the  $\beta$  values. While to maximize the success rate is our main objective, we use the remaining variables to guide the optimization process. This motivates the factorization of the mapping into:

$$\Psi_{\beta \rightarrow \gamma}^{grasping} = f_1(s) * (f_2(d) + f_3(c) + f_4(m) + f_5(l) + f_6(i) + f_7(g)) \quad (4.23)$$

$$\Psi_{\beta \rightarrow \gamma}^{homing,placing} = f_1(s) * (f_2(d) + f_3(c) + f_4(m) + f_5(l) + f_6(i)) \quad (4.24)$$

where we included the grasp evaluation in the symbolic grasping mapping, and have chosen the corresponding functions  $f_{1:7}$  in a way, that we acquire a normalization of the involved variables and that all of them are becoming minimization objectives. We start by defining  $f_1$  as

$$f_1(s) = \epsilon_s * (1 - s + \epsilon_{offset}) \quad s \in [0, 1] \quad (4.25)$$

whereby  $s$  is the success rate,  $\epsilon_s$  a normalization constant, and  $\epsilon_{offset}$  is an additional constant offset, because we do not want to stop optimizing our objective function, even if our primary goal  $s = 1$  is reached. We further define  $f_{2:4}$  as

$$f_2(d) = \left(\frac{d}{\theta_d}\right)^{n_d} \quad d \in [0, \infty[ \quad (4.26)$$

$$f_3(c) = \left(\frac{c}{\theta_c}\right)^{n_c} \quad c \in [0, \infty[ \quad (4.27)$$

$$f_4(m) = \left(\frac{m}{\theta_m}\right)^{n_m} \quad m \in [0, \infty[ \quad (4.28)$$

whereby  $\theta_d$ ,  $\theta_c$  and  $\theta_m$  are the thresholds of the distance, collision and limits, respectively, and  $n_d, n_c$  and  $n_m$  are normalization constants. Similar, we define  $f_{5:7}$  as

$$f_5(l) = \left(\frac{l}{\eta_l}\right)^{n_l} \quad l \in [0, \infty[ \quad (4.29)$$

$$f_6(i) = \left(\frac{i}{\eta_i}\right)^{n_i} \quad i \in [0, \infty[ \quad (4.30)$$

$$f_7(g) = \left(\frac{g - g_m}{\eta_g - g_m}\right)^{n_g} \quad g \in [0, g_m] \quad (4.31)$$

$$(4.32)$$

whereby  $n_l, n_i, n_g$ ,  $\eta_l$ ,  $\eta_i$  and  $\eta_g$  are normalization constants.  $g_m$  represents the maximum number of time steps, until we stop the grasp evaluation, if the object did not fall from the hand.

The normalization constants were chosen to be  $n_d = 5$ ,  $n_c = 3$ ,  $n_m = 2$ ,  $n_l = 1.1$ ,  $n_i = 1.1$ ,  $n_g = 2$ ,  $g_m = 1000$ ,  $\epsilon_s = 10$ ,  $\epsilon_{offset} = 0.2$ ,  $\eta_l = 50$ ,  $\eta_i = 50$ , and  $\eta_g = 200$ .

## 4.4 Covariance Matrix Adaption (CMA) to Optimize $\mathcal{P}$

Given the objective function for  $\alpha$  from our motion primitive  $\mathcal{P}$ , our goal is to search for  $\mathcal{P}^*$  by finding the optimal  $\alpha^*$ . Because the search space is highly irregular, methods such as gradient descent or simulated annealing were not successful in our initial experiments. Therefore, we used an evolution strategy (ES) algorithm, called *covariance matrix adaption* (CMA) (Hansen and Ostermeier, 1996; Hansen and Kern, 2004). ES algorithms are classified by their number of parents  $\mu$ , their children  $\lambda$ , the reproduction type, and their selection type. The standard procedure involves sampling  $\mu$  parents from an initial distribution, and using a reproduction technique to spawn  $\lambda$  children. Then, by using the best parameters of the last generations, a new set of children is selected. A set of children is also called a generation or a population. CMA uses a normal distribution to select the new population. The normal distribution is defined by two parameters. First, its mean vector, which is calculated by using a weighted average over the best parents. Second, the covariance matrix, which is calculated *to fit the search distribution to the contour lines of the objective function* (Hansen, 2006). An additional step size parameter varies the size of the covariance matrix.

To generate the first parent population, we randomly sample from the domain of  $\alpha$ . For the optimization task, we restricted it to  $\alpha \in [10^{-3}, 10^9]$ . We then evaluate the objective function for each parent and calculate the  $\gamma$  value. To generate a new set of  $\alpha$  values, CMA uses the equation (Hansen, 2006):

$$\alpha_{\mathbf{k}}^{(\mathbf{g}+1)} \sim \mathcal{N}(\mathbf{m}^{(\mathbf{g})}, (\sigma^{(\mathbf{g})})^2 \mathcal{C}^{(\mathbf{g})}) \quad (4.33)$$

whereby  $g$  is the current generation,  $\alpha_{\mathbf{k}}^{(g+1)}$  is the  $k$ -th offspring of the new generation  $g+1$ ,  $\mathbf{m}^{(g)}$  the mean value of the distribution of generation  $g$ ,  $\sigma$  is the step width and  $\mathcal{C}^{(g)}$  the covariance matrix of the generation.

The mean value of the distribution is calculated by (Hansen, 2006):

$$\mathbf{m}^{(g+1)} \leftarrow \sum_{i=1}^{\mu} w_i \cdot \alpha_{i:\lambda}^{(g+1)} \quad (4.34)$$

whereby  $w_i$  is a positive weight coefficient for recombination,  $x_{i:\lambda}$  is the  $i$ -th best individual from the  $\lambda$  children, and we constrain it to be  $\sum_{i=1}^{\mu} w_i = 1$  and  $w_i > 0$  for  $i = 1, \dots, \mu$ . The weights  $w_i$  are chosen proportional to the  $\gamma$  evaluation of each  $\alpha$ , which effectively means, that the  $\mu$  best  $\alpha$  parameters are weighted and used for the mean calculation. For  $w_i = 1/\mu$ , we would directly use the mean value of the  $\mu$  best  $\alpha$  parameters.

Please note, that the selection is usually divided into two categories: First, we could select a new population by using all parents and children. This is called an elitist algorithm and is depicted by the notation  $(\mu + \lambda)$ . Second, we could use only the children to reproduce, which is called a non-elitist selection and depicted by  $(\mu, \lambda)$ , as seen in Equation 4.34. In our experiments, we used an  $(1+40)$ -CMA elitist algorithm. As discussed by Auger and Hansen (2005), a larger  $\lambda$  can help us finding a better solution, if the search space is highly nonlinear. Our  $\lambda = 40$  approach was a trade-off between better solutions and the approximate calculation time.

## 4.5 Final Remarks on Optimizing $\mathcal{P}$

We have shown in this Chapter, how we can in principle create a motion primitive  $\mathcal{P}$  from tasks  $\Phi$  and their *precision parameters*  $\alpha$ . In order to find the approximate optimal motion primitive  $\mathcal{P}^*$ , we casted  $\alpha$  to an optimization problem, and we introduced CMA, in order to find a solution. Before we demonstrate that we are indeed able to find an approximate optimal  $\mathcal{P}^*$  for different symbolic actions, we will briefly examine two practical considerations. First, we will summarize the general assumption, on which our approach is based. Second, we discuss the time consumption of the optimization procedure.

### 4.5.1 General Assumptions

Optimizing  $\mathcal{P}$  is mainly based on the assumption that a symbolic planner and a motion planner are already predefined. But we also require that specific properties of the framework are existent.

- The symbolic planner is able to handle uncertain state transitions

Our introduced symbolic framework was based on NID rules, which incorporated non-deterministic state transitions. We generally require, that uncertain state transitions can be handled. Otherwise, there would be only deterministic actions, which are unrealistic, because of the uncertainty in a real environment.

- Each symbolic action has a desired outcome

Also, we need to know, which action outcome is desired. For example, if we use the *grasp(X)* action, we desire to get the effect *inhand(X)*. If the desired outcome is unknown, it would be possible to use the outcome with the highest probability as the desired one, but this is currently not supported.

- Motion Planner is able to incorporate  $\mathcal{P}$  into the planning procedure

Of course, the motion planner has to be able to use the optimized  $\mathcal{P}$ . While this restricts the number of motion planner we can use, it is essential in order to define a measurement of optimality. Otherwise, the uncertainty in state transitions would rise, because we could only calculate feasible trajectories instead of optimal ones.

#### 4.5.2 Time for Offline Optimization

The computation of  $\mathcal{P}^*$  requires us to obtain a trajectory for a fixed set of scenarios  $K$ , in order to calculate the value  $\gamma$  for one  $\alpha$ . For each scenario, we have to invoke the underlying motion planner. Furthermore, the CMA algorithm needs to compute up to 2000 evaluations of  $\gamma$ , to converge to a solution for one symbolic action. In our experiments, this took up to  $> 12$  hours on a 2500 Mhz, 8GB Ram Laptop. But this computation will be done offline, which means that we need to compute  $\mathcal{P}^*$  only once, and are able to use it for future plans.

# Chapter V

## Experiments

In the previous section, we discussed the theoretical foundations, about how we can optimize the desired transition probabilities of symbolic actions. In this section, we will demonstrate, that this method can indeed increase the probabilities for a set of symbolic actions. We therefore used three symbolic actions, *GraspCylinder*, *PlaceCylinder* and *Homing*. The corresponding NID rules for each action are summarized in Figure 5.1. Each action has a transition probability, called the *success rate*, which we will refer to as  $s$ . We simplified each rule by assuming that all undesired outcomes are noise. Our goal will be to demonstrate that we are able to increase  $s$  for each action, by optimizing the *precision parameters*  $\alpha$  of its motion primitive  $\mathcal{P}$ .

During the discussion about the experiments, we will make use of several definitions, which we want to clarify here:

1. Scenario: A static environment, where the robot starts from a specific configuration and has to fulfil a task, like grasping a purple cylinder. If we refer to a scenario, this means the setting, not a planning procedure.
2. Scenario evaluation: Using a set of  $\alpha$  *precision parameters*, a scenario evaluation will apply the mappings  $\Psi_{\alpha \rightarrow q}$  and  $\Psi_{q \rightarrow \beta}$  to obtain a vector  $\beta$ .
3. Evaluation: The results from evaluating a set of scenarios. The obtained  $\beta$  values are combined by the mapping  $\Psi_{\beta K \rightarrow \beta}$  into one  $\beta$  value, and afterwards combined to the scalar value  $\gamma$  by using  $\Psi_{\beta \rightarrow \gamma}$ .
4. Trial: A set of evaluations. For example, if we optimize the parameters over 1000 evaluations by using CMA, we will call this one trial.
5. Experiment: The general term for describing what we intend to do. Usually an experiment will involve the average over several trials to obtain a statistically relevant outcome.

We conduct for each symbolic action two experiments. In the first experiment, we want to show, how CMA can be used to optimize the  $\alpha$  parameters of the motion primitive  $\mathcal{P}$  on a fixed set of  $K = 6$  scenarios. Each  $\alpha$  vector is constrained on the

Grasping	<b>GraspCylinder</b> (X): on(X,Y), cylinder(X), table(Y) $\rightarrow \begin{cases} s_{grasp} & inhand(X), \neg on(X,Y) \\ 1 - s_{grasp} & noise \end{cases}$
Placing	<b>PlaceCylinder</b> (X,Y): inhand(X), on(Y,Z), table(Z), cylinder(X) $\rightarrow \begin{cases} s_{place} & on(X,Y), \neg inhand(X) \\ 1 - s_{place} & noise \end{cases}$
Homing	<b>Homing</b> : $\neg home(R)$ , robot(R) $\rightarrow \begin{cases} s_{homing} & home(R) \\ 1 - s_{homing} & noise \end{cases}$

Table 5.1: The three symbolic actions, which we optimized in our experiments. Each one has a *success rate* called  $s$ , which describes the probability, that the outcome will be reached.

domain  $\alpha \in [10^{-3}, 10^9]$ . During the optimization, we transformed  $\alpha$  to the logarithmic space, because we observed during initial tests, that changing a small parameter has a greater effect on the outcome. By applying the logarithmic transformation, the optimization algorithm prefers to search more frequently in the space of lower numbers. Finally, we will show the average results of CMA in comparison to a monte carlo sampling technique. Monte Carlo (MC) sampling means, that we sample a parameter vector  $\alpha$  at random from the domain, calculate its  $\gamma$  value, and compare it to the best  $\gamma$  value obtained so far. If it is better, we choose the new  $\alpha$  parameter as our best result.

In the second experiment we will show, how the obtained optimized  $\alpha$  parameters generalize to unseen scenarios, and how they perform, in comparison to manually specified parameters by an expert. We therefore use the original scenarios and add a uniform noise to the position of each object. Each object stands on a table with a width and length of 0.4m. If we add a uniform noise of for example 0.2, this means, that we add a random value from the uniform distribution  $\mathcal{U}(-0.2m, +0.2m)$  to the  $x$  and  $y$  position of the objects. To circumvent that objects touch each other, or that objects fall from the table, we repeat the noise adding procedure, until all constraints are fulfilled. If the noise value increases, the scenario will become more and more unknown to the robot. We will then show, how much noise can be added, until the *success rate*  $s$  of each symbolic action converges.

Each experiment assumes, that a mapping from configuration space to relational state space exists, which can be used to compare the desired goal state to the obtained state from the goal configuration of the robot. We implement this by using a shortcut over the *success rate*, which we defined in Section IV. For example, for



the *GraspCylinder* action, we investigate, if any limits or collisions are reached, if the motion planner has converged, if the center of the hand coincides with the center of the cylinder which we want to grasp, and if the grasp was successful, which means, that indeed  $inhand(X)$  is fulfilled. Please note, that this is a more strict mapping to relational space, for two reasons. First, instead of using the final configuration, we additionally check if there was any collision or limit violation. If this is true, we regard the next state as not successful. Second, we instantiate a grasp evaluation procedure at the final position. If the cylinder is not correctly grasped, it will fall from the hand, and we conclude that  $inhand(X)$  is not fulfilled. Both are additional restrictions on the relational state, which were not used in the original mapping. This is why the manually specified parameters are much worse in our experiments. Nevertheless, we think that both restriction are necessary, in order to correctly identify a successful state transition, and obtain parameters which are more robust against noise.

Before showing the experimental results, we will briefly introduce our computational tools used here. We conducted each experiment on a simulated robotics platform, using the libraries *Open Dynamics Engine (ODE)*<sup>1</sup> for physical simulations and *SWIFT++*<sup>2</sup> for proximity measurements. Furthermore, we used the *SHARK*<sup>3</sup> library for the calculation of the covariance matrix adaption (CMA) algorithm.

---

<sup>1</sup><http://www.ode.org/>

<sup>2</sup><http://gamma.cs.unc.edu/SWIFT++/>

<sup>3</sup><http://sourceforge.net/projects/shark-project/>

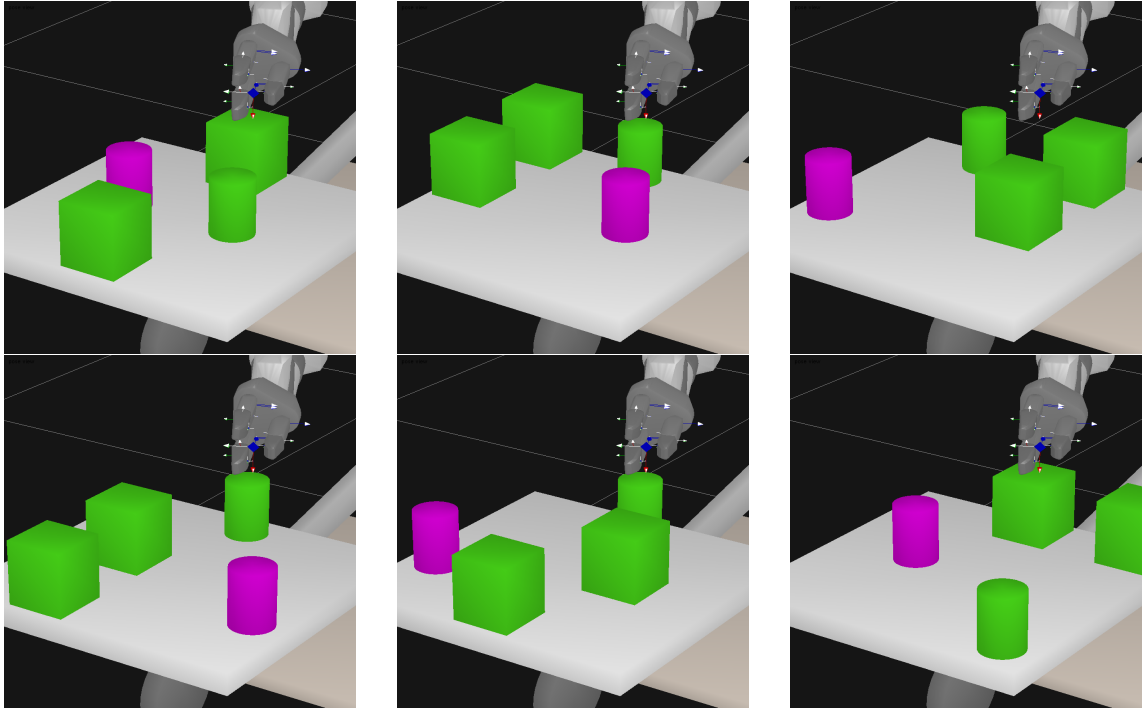


Figure 5.1: A set of scenarios for evaluating the *GraspCylinder* action. In each experiment, the robot has to grasp the purple cylinder, while avoiding to touch green objects or the table.

## 5.1 Optimizing Grasping Action

The symbolic grasping action will be optimized by using 6 different scenarios. In each scenario, the robot has to grasp a purple cylinder, as depicted in Figure 5.1. The choice of those scenarios was motivated by situations, which were difficult to perform for the motion planner, by using manually specified parameters by an expert. We compute the  $\gamma$  value for one set of  $\alpha$  parameters by calculating the average over one evaluation, as discussed in Chapter IV. Our optimization algorithm CMA(1+40) will optimize the parameters, until a number of 2000 evaluations is reached. Because we do not have any guarantee on convergence to the global optimum, we restart the algorithm 10 times. Our main interest will be to show how many evaluations are on average necessary to reach the maximum *success rate*  $s_{grasp}$ .

This optimization process can be seen in Figure 5.2. It shows in the upper graph the *success rate*  $s_{grasp}$ , for CMA, MC and the manually specified parameters by an expert. One can see, that we obtained a  $s_{grasp}$  of zero for the manually specified parameters. This is due to the more restricted choice of our success calculation, as discussed above, and the choice of challenging scenarios, where we observed in pre-experiments, that it was difficult to obtain a successful grasp. Between CMA and MC, we can observe that CMA slightly outperforms MC during the optimization. The lower graph addi-

tionally shows the scalar value  $\gamma$  for each evaluation. While the manually specified parameters lead to a worse  $\gamma$ , we can observe that CMA and MC are both able to optimize the value. Please note that we inverted the  $y$ -axis, so that both desired values of  $s_{grasp} = 1.0$  and  $\gamma = 0.0$  are at the top of the graph.

The results for the final evaluation step is depicted in Table 5.2. It shows the average *success rate*  $s_{grasp}$  and the average  $\gamma$  for both CMA and MC, together with the value for one evaluation of the manual specified parameters. Please note, that the evaluation of one  $\alpha$  is deterministic, because we removed uncertainty by using a constant start configuration in each scenario. Otherwise, we would have to repeat the evaluation of each scenario multiple times. Table 5.2 finally shows that, on average, the CMA algorithm outperforms MC sampling and can obtain better values for  $\gamma$  and  $s_{grasp}$ .

So far we demonstrated, that we can obtain  $\alpha$  parameters, which achieve a higher *success rate*  $s_{grasp}$  on the initial set of scenarios. The next step is to compare the final trajectories, which were planned by using manually specified parameters and the optimized ones. We showed the final trajectories for each scenario from three different points of view in Figure 5.3 and 5.4, respectively. In each image, three lines can be seen, which correspond to the position of the center of the tip of the robot fingers at different instances of time during the execution. The obtained trajectory by using manually specified parameters are shown in blue with a dashed line, and the one by using optimized parameters with a solid red line. While both trajectories seem to be similar, we could observe, that the manual parameters had a significantly worse grasp evaluation. Moreover, in 5 of 6 cases, we observed, that the outer fingers collided during the grasp. This is not visible in the images, but is the reason, why the manual parameters cannot obtain a successful state transition in the shown scenarios.

After showing, that we are able to optimize the parameters on a fixed set of scenarios, we want to show, that the new obtained parameters can significantly improve planning, even in unseen scenarios. Therefore, we added uniform noise to each object position, as explained in the previous section. For each noise value, we performed 200 evaluations on each scenario. For a noise of 0.0, the scenarios are the same as in our optimization experiment. If the noise increases, the scenarios become random arrangements. We end up with a total of 1200 scenarios for each noise value between 0.0 to 1.0. Altogether we evaluated a total of 40 different noise values by using a step size of 0.025. The results can be seen in Figure 5.5. If we start with zero noise, the best  $\alpha$  parameter from our optimization process can reach a *success rate* of  $s_{grasp} = 1.0$ , while the manual specified parameters have  $s_{grasp} = 0.0$ , as we discussed above. If we increase the noise value, the scenarios become more and more random. We can observe, that the manual parameters get better, because there will occur easier scenarios more frequently, and also those scenarios, which the expert used to optimize. The optimized parameters can be seen to decrease, because the learned scenarios will occur more seldom. Nevertheless, even in random scenarios, the optimized parameters reach a higher *success rate* as the manual ones. Eventually, we compared the average *success rate* at the noise values of 0.0, 0.5 and 0.9 in Table ??, where we show the NID rules,

obtained by evaluating 1200 scenarios. In the left column, we can see the results of the *success rate* , by using a manual specified  $\mathcal{P}$ . The right column shows the results after we used our optimized  $\mathcal{P}^*$  parameters. The best *success rate* for each noise value is highlighted in red. The symbolic NID rules in this formalization can directly be used by a symbolic planner.

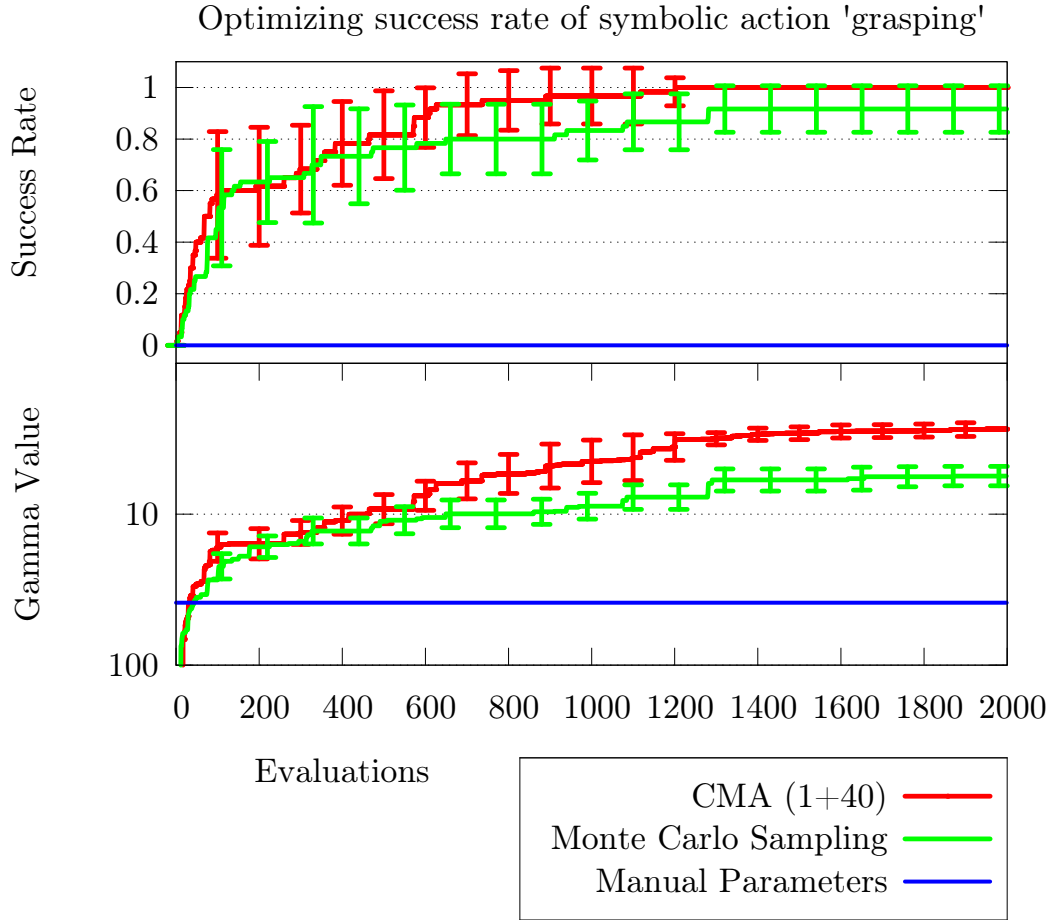


Figure 5.2: The upper graph shows the average number of evaluation steps for CMA(1+40), which are necessary to reach a specific success rate. In the lower graph, the average fitness value is shown. It can be seen, that the CMA(1+40) converges to a success rate of 1 for this experiments after approximately 1300 evaluations.

Algorithm	$\gamma^{avg}$	$s_{grasp}^{avg}$
Optimized parameters (10 trials)	<b>2.73</b>	<b>1.00</b>
Monte Carlo Sampling (10 trials)	5.59	0.92
Manual specified parameters	38.58	0.00

Table 5.2: Comparison between optimized, manual and random parameters for the *GraspCylinder* experiment

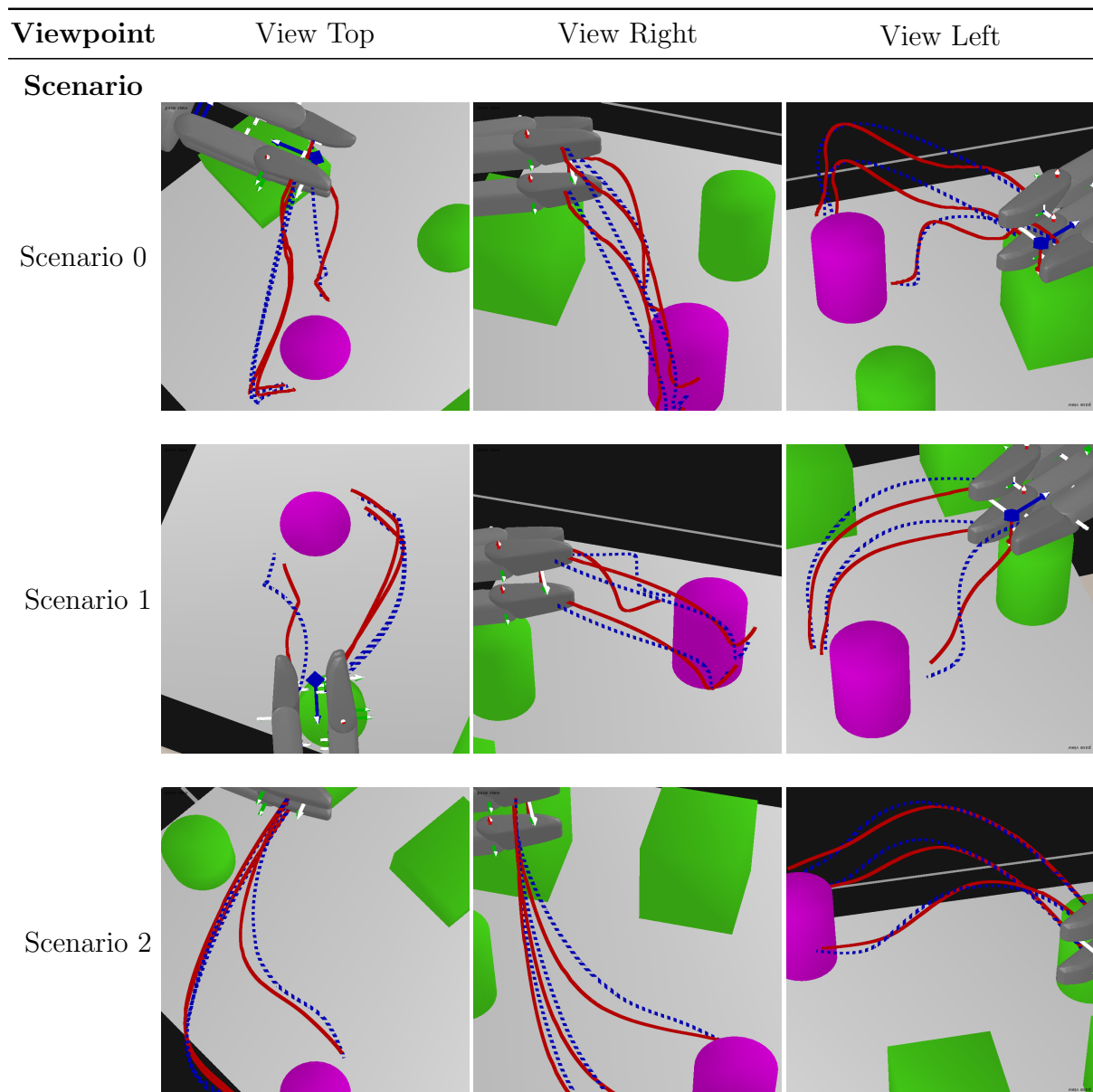


Figure 5.3: The final grasping trajectories in the first three scenarios and from three different perspectives, each with the CMA-optimized parameters (red,solid) compared to the manual parameters (blue, dashed), specified by an expert. See text for clarifications.

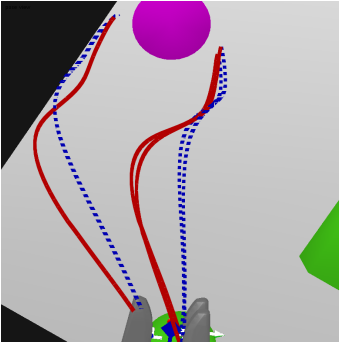
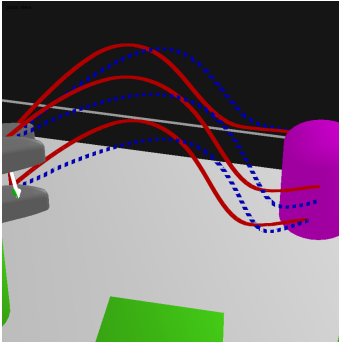
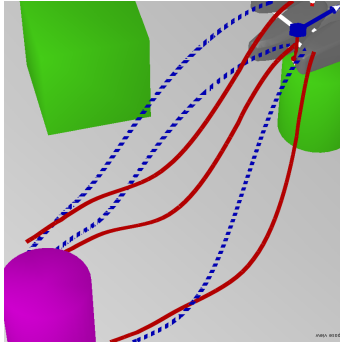
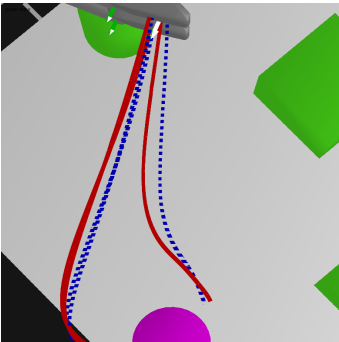
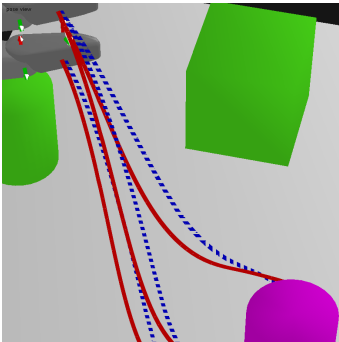
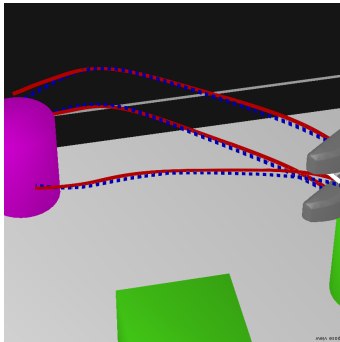
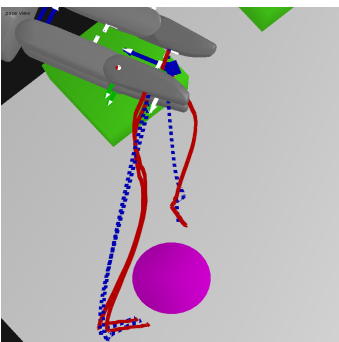
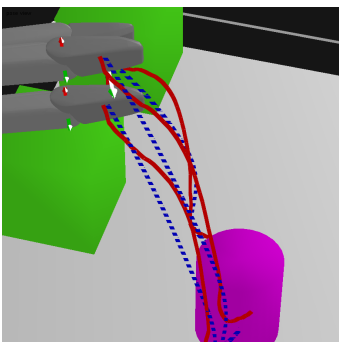
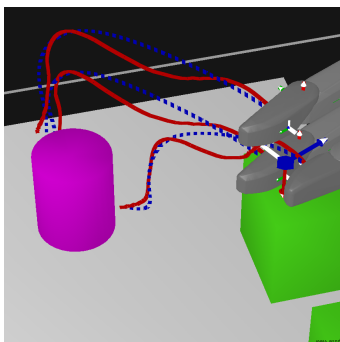
Viewpoint	View Top	View Right	View Left
Scenario			
Scenario 3			
Scenario 4			
Scenario 5			

Figure 5.4: Final trajectories on the last three scenarios. Compare with Figure 5.3.

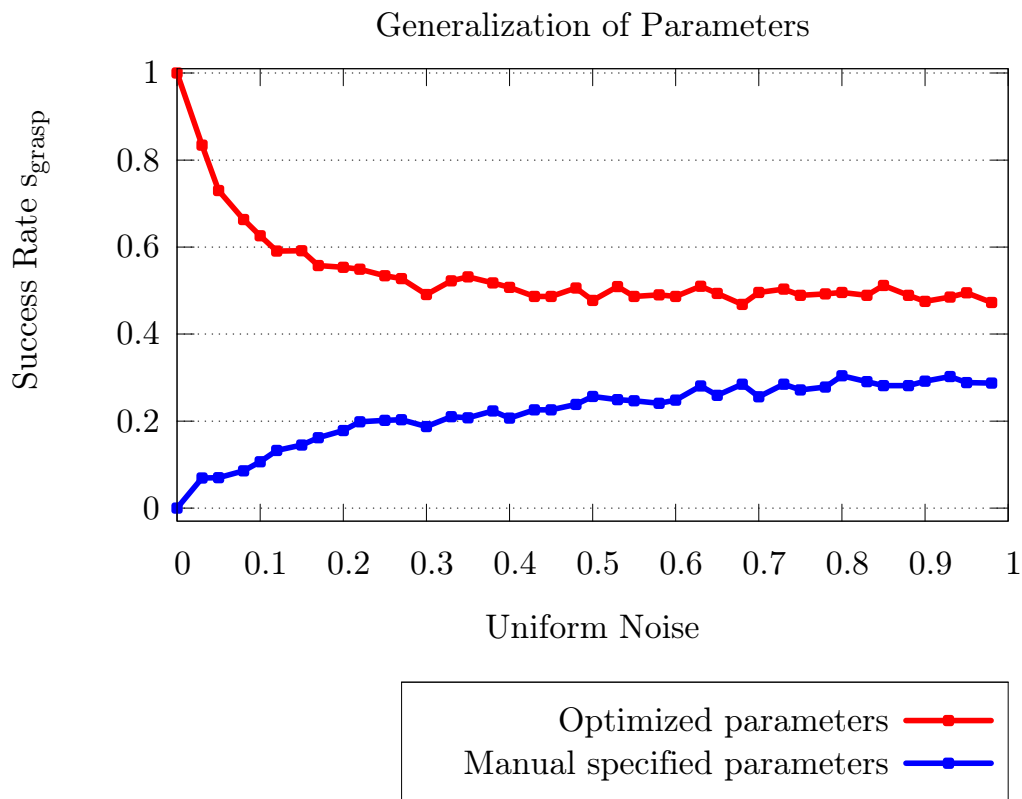


Figure 5.5: Generalization to unseen scenarios. Starting from the initial set of scenarios, on which the parameters were optimized, we increase the noise, until the scenarios are completely random. It can be seen that the optimized parameters obtain a better generalization, in comparison to the manually specified ones. Each datapoint is obtained by calculating the average success rate over 1200 scenarios. See text for clarifications.



Uniform Noise	Symbolic NID Rules	
	With Manual $\mathcal{P}$	With Optimized $\mathcal{P}^*$
0.0	<b>GraspCylinder</b> (X): $on(X, Y)$ , <i>cylinder</i> (X), <i>table</i> (Y) $\rightarrow \begin{cases} 0.0 & inhand(X), \neg on(X, Y) \\ 1.0 & noise \end{cases}$	<b>GraspCylinder</b> (X): $on(X, Y)$ , <i>cylinder</i> (X), <i>table</i> (Y) $\rightarrow \begin{cases} \mathbf{1.0} & inhand(X), \neg on(X, Y) \\ 0.0 & noise \end{cases}$
0.5	<b>GraspCylinder</b> (X): $on(X, Y)$ , <i>cylinder</i> (X), <i>table</i> (Y) $\rightarrow \begin{cases} 0.26 & inhand(X), \neg on(X, Y) \\ 0.74 & noise \end{cases}$	<b>GraspCylinder</b> (X): $on(X, Y)$ , <i>cylinder</i> (X), <i>table</i> (Y) $\rightarrow \begin{cases} \mathbf{0.48} & inhand(X), \neg on(X, Y) \\ 0.52 & noise \end{cases}$
0.9	<b>GraspCylinder</b> (X): $on(X, Y)$ , <i>cylinder</i> (X), <i>table</i> (Y) $\rightarrow \begin{cases} 0.29 & inhand(X), \neg on(X, Y) \\ 0.71 & noise \end{cases}$	<b>GraspCylinder</b> (X): $on(X, Y)$ , <i>cylinder</i> (X), <i>table</i> (Y) $\rightarrow \begin{cases} \mathbf{0.47} & inhand(X), \neg on(X, Y) \\ 0.53 & noise \end{cases}$

## 5.2 Optimizing Placing Action

The second symbolic action which we optimized is the *PlaceCylinder* action. Likewise to the *GraspCylinder* action, we used  $K = 6$  scenarios for one evaluation of a parameter vector  $\alpha$ . The start configurations for the 6 different scenarios can be seen in Figure 5.6. They are similar to the *GraspCylinder* action, but with a different starting configuration, where the hand of the robot is grasping the purple cylinder. Before the motion planner is started, the hand closes, until all fingers are touching the surface of the cylinder. The goal of each scenario is then to place the purple cylinder onto the green cylinder. Similar to the *GraspCylinder* action, we optimized the parameters by using CMA and MC sampling. Figure 5.7 shows the optimization process for both methods. Each function represents the average over 10 trials. The results for the final evaluation of CMA can be found in Table 5.3, together with the comparison to MC and manually specified parameters. The final trajectories, obtained by using the best optimized  $\alpha$  parameters from CMA and the manually specified ones, are shown in Figure 5.8. We can see that there is a big discrepancy between manual and CMA, which is mainly due to the difficult set of scenarios, which we intentionally used to be able to generalize better to future scenarios. Each scenario, which is depicted in Figure 5.8 and Figure 5.9, respectively, was successfully finished by the optimized parameters (red, solid line). The trajectories, obtained by manual parameters (blue, dashed line), either lead to a collision between the two cylinders (Scenarios 1 – 4), or do not reach the goal position (Scenarios 0, 5). This is also reflected in the  $\gamma$  value, as depicted in Table 5.3.

Eventually, similar to the *GraspCylinder* action, we performed an experiment, which demonstrates, how the optimized parameters generalize to unseen scenarios. Figure 5.10 shows the performance of the *success rate*, if we add different uniform noise distributions to the position of the objects. It can be seen, that the manual specified parameters lead to a *success rate* of zero, which is caused by our very restrictive computation of the *success rate*. In detail, this is caused by a lower precision to avoid collisions. In all experiments, we could observe, that this lower precision leads to a collision between the purple cylinder and the green one during the end of each trajectory. While this collision often does not cause damage, our goal is to avoid any collision during the execution. Therefore, we declare the plan as non successful, even if there is only a minimal collision. The final graph in Figure 5.10 consists of datapoints, which show the average *success rate* of 420 scenario evaluations. Additionally, the resulting symbolic NID rules for different noise levels are summarized in Table 5.4.

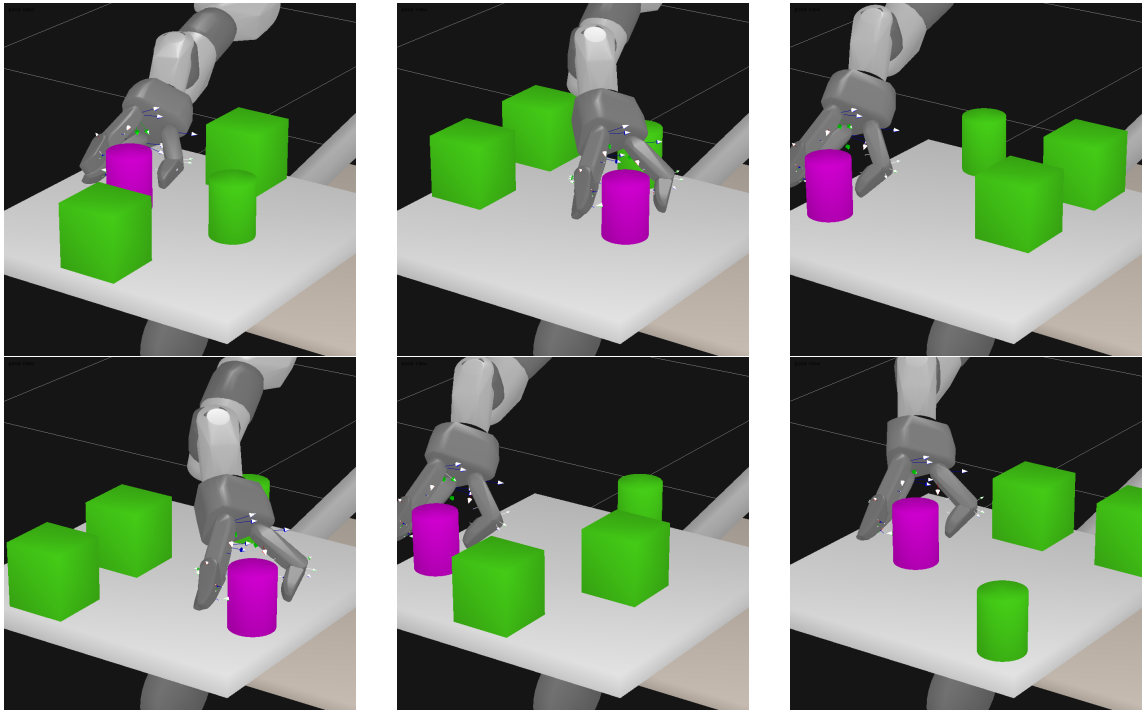


Figure 5.6: A set of scenarios for evaluating the *PlaceCylinder* action. In each experiment, the robot has to place the purple cylinder on the green cylinder, while avoiding to touch green objects or the table.

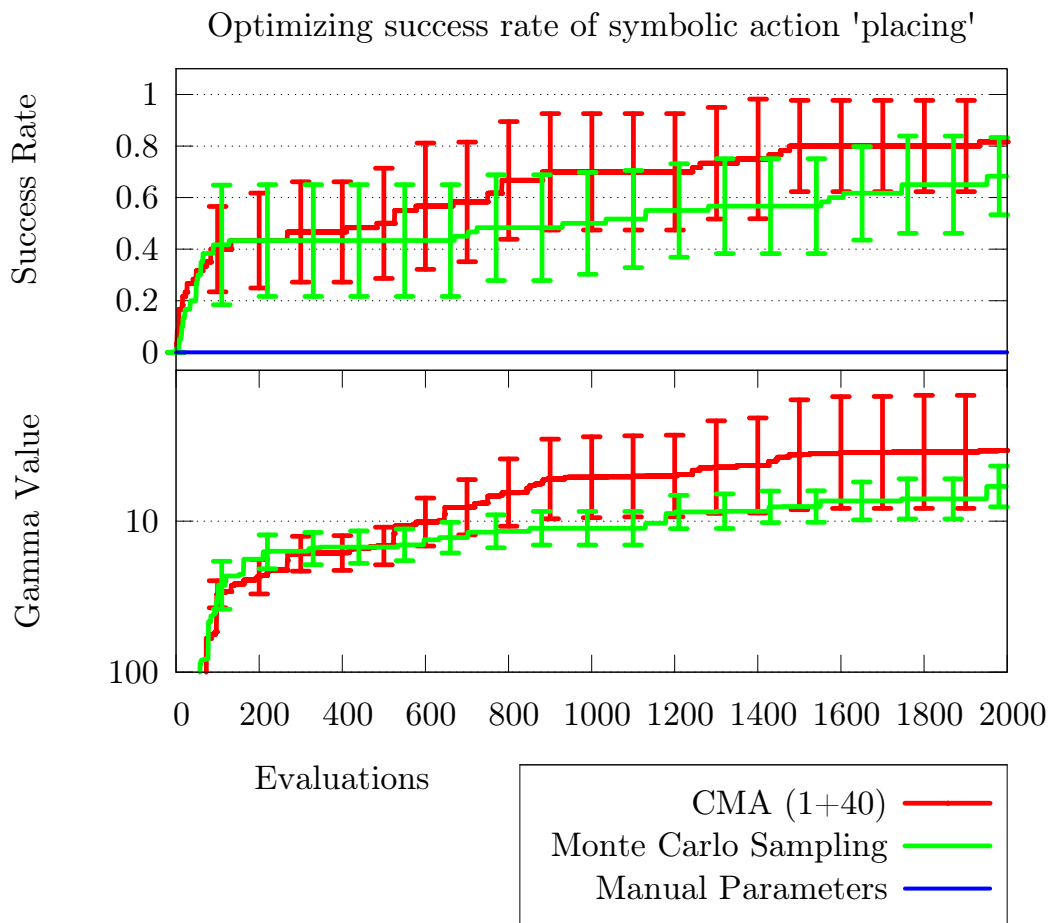


Figure 5.7: CMA (1+40) with random initializations, averaged over 10 different restart. This optimization routine uses the set of experiments from Figure 5.6.

Algorithm	$\gamma^{avg}$	$s_{grasp}^{avg}$
Optimized parameters (10 trials)	<b>3.41</b>	<b>0.82</b>
Monte Carlo Sampling (10 trials)	5.88	0.68
Manual specified parameters	1784226.95	0.00

Table 5.3: Comparison between optimized, manual and random parameters for the *PlaceCylinder* experiment

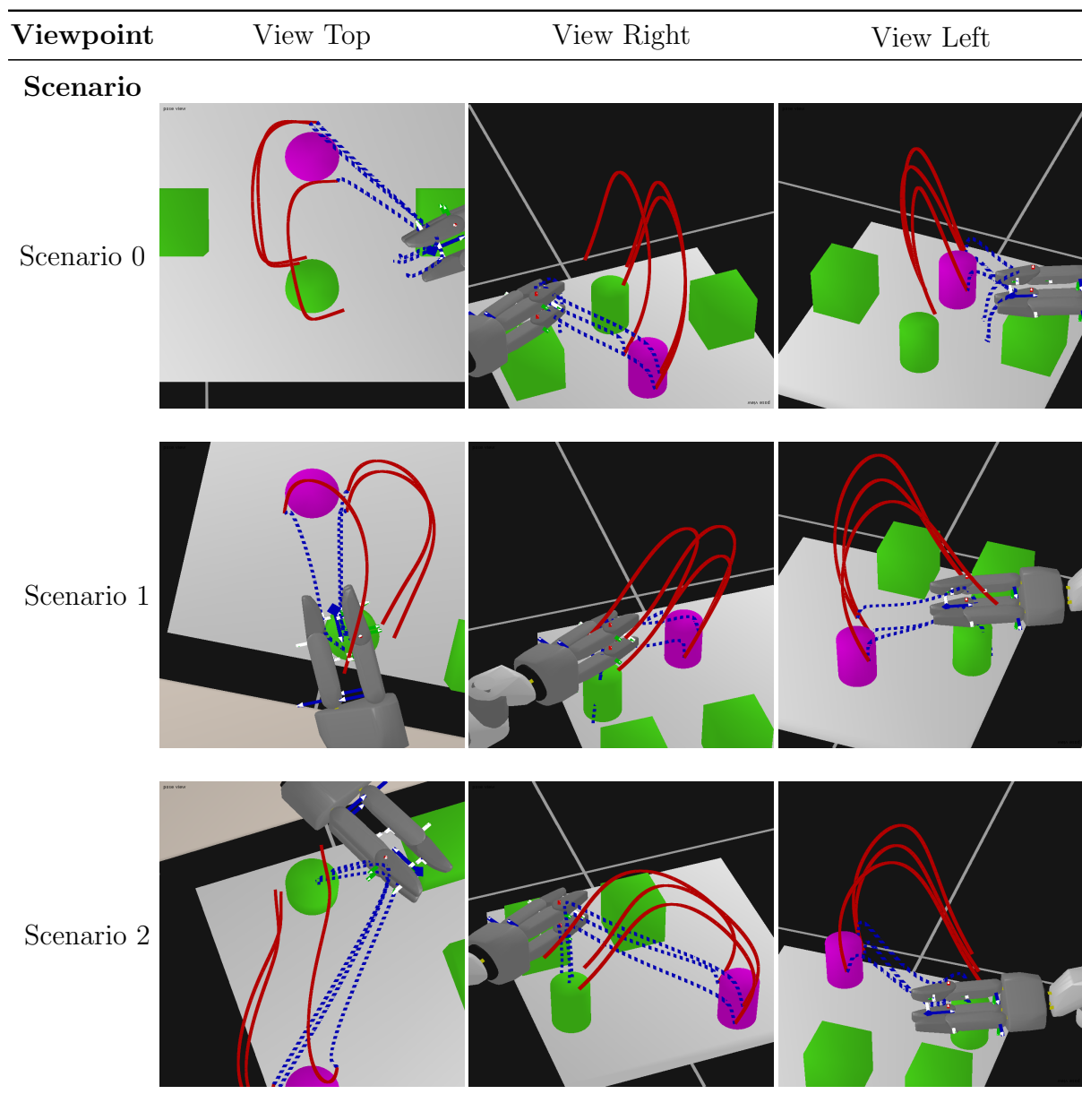


Figure 5.8: The final placing trajectories from three different perspectives. In each perspective, we can see the plan computed by using CMA-optimized parameters (red, solid) compared to plans based on manual specified parameters (blue, dashed). Each row shows the results from one experiment of Figure 5.6. The columns represent different points of view. In the first column, we see the scenario from the top, in the second from the left and in the third from the right with respect to the robot. The trajectories, based on the optimized parameters, were successful in all experiments, while all plans with manual parameters were not able to meet our success criteria. This can be seen in the last two experiment rows, where the plans with manual parameters lead to a collision between both cylinders. The first experiment shows that the manual specification of parameters can also lead to plans, where the final goal is not even reached.

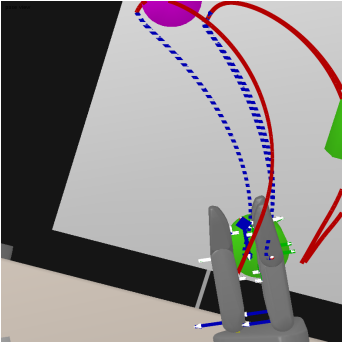
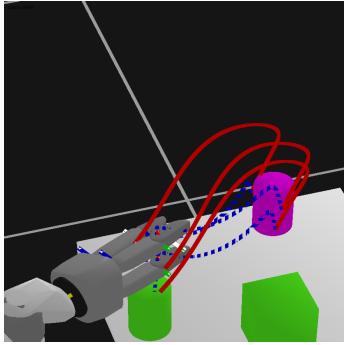
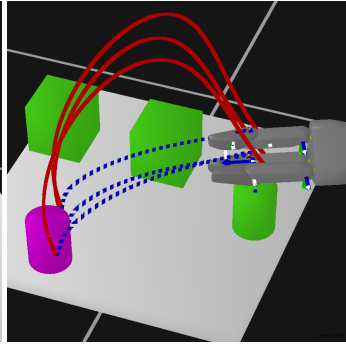
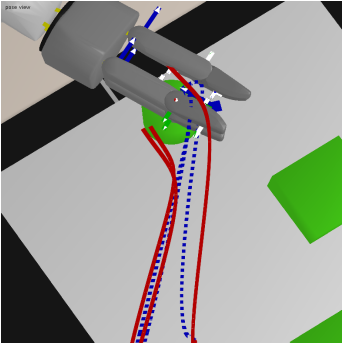
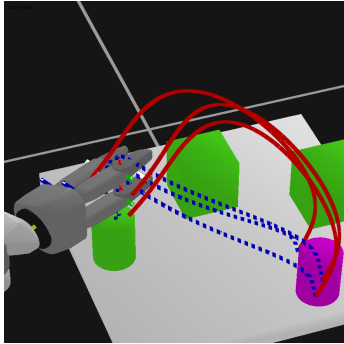
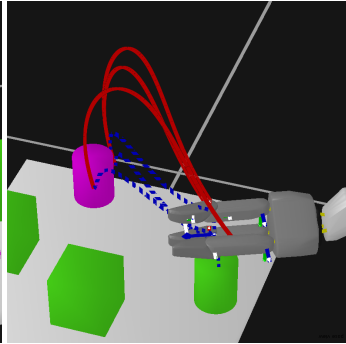
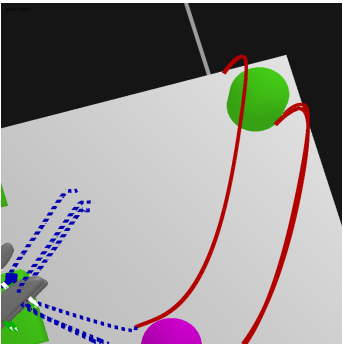
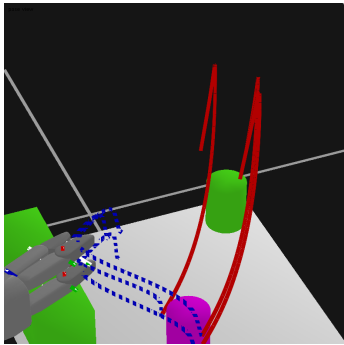
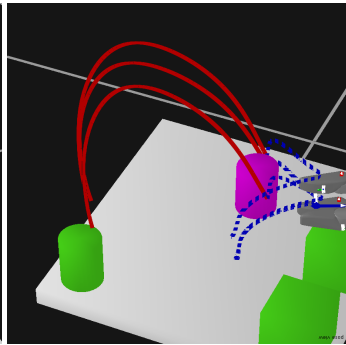
Viewpoint	View Top	View Right	View Left
Scenario			
Scenario 3			
Scenario 4			
Scenario 5			

Figure 5.9: Each row represents the last three experiments from Figure 5.6. It can be seen, that the plans computed with manual specified parameters were not able to compute a collision free path in the first two rows of experiments. In the last experiments, we again see that the final goal was not reached. Plans computed with CMA-optimized parameters (red,solid) were able to find successful plans in all cases.

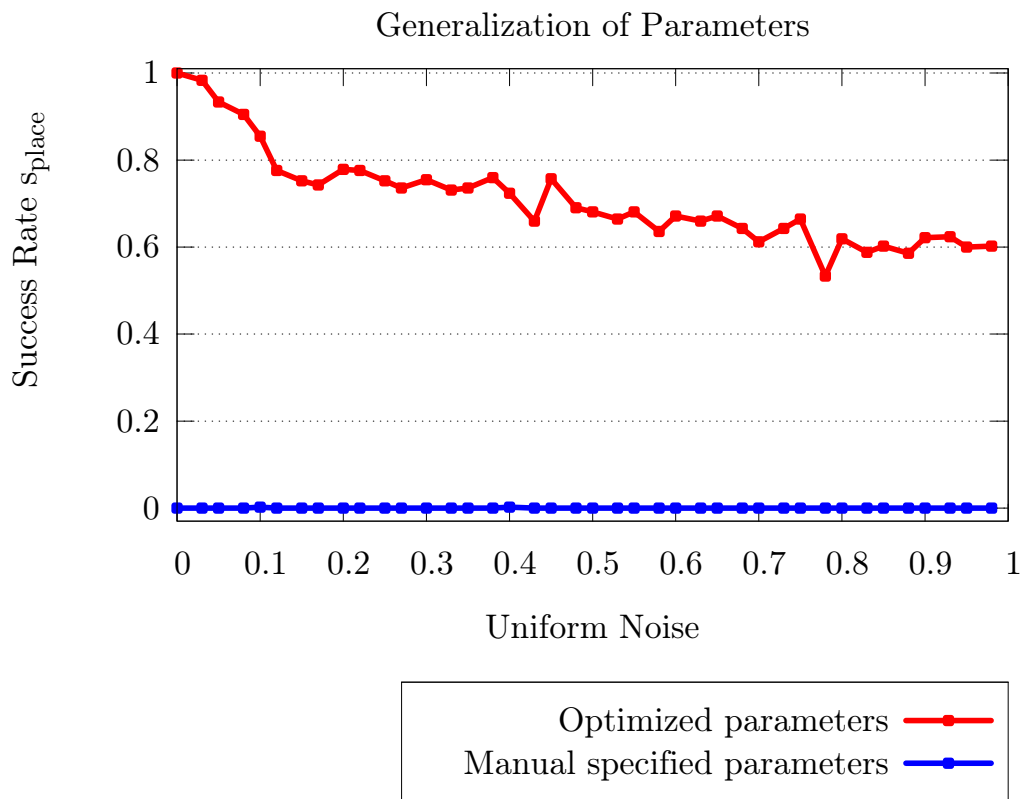


Figure 5.10: Generalization to unseen scenarios. Similar to the *GraspCylinder* evaluation, we added a uniform noise to the position of the objects. As can be seen, the optimized parameters maintain a high success rate, even for completely random actions. Because of our very restrictive success rate, the manual parameters cannot obtain a single success. Each datapoint is obtained by the average over 420 scenarios. See text for clarifications.

Uniform Noise	Symbolic NID Rules	
	With Manual $\mathcal{P}$	With Optimized $\mathcal{P}^*$
0.0	<b>PlaceCylinder</b> (X,Y): <i>inhand</i> (X), <i>on</i> (Y, Z), <i>table</i> (Z), <i>cylinder</i> (X) $\rightarrow \begin{cases} 0.0 & \textit{on}(X, Y), \neg\textit{inhand}(X) \\ 1.0 & \textit{noise} \end{cases}$	<b>PlaceCylinder</b> (X,Y): <i>inhand</i> (X), <i>on</i> (Y, Z), <i>table</i> (Z), <i>cylinder</i> (X) $\rightarrow \begin{cases} \mathbf{1.0} & \textit{on}(X, Y), \neg\textit{inhand}(X) \\ 0.0 & \textit{noise} \end{cases}$
0.5	<b>PlaceCylinder</b> (X,Y): <i>inhand</i> (X), <i>on</i> (Y, Z), <i>table</i> (Z), <i>cylinder</i> (X) $\rightarrow \begin{cases} 0.0 & \textit{on}(X, Y), \neg\textit{inhand}(X) \\ 1.0 & \textit{noise} \end{cases}$	<b>PlaceCylinder</b> (X,Y): <i>inhand</i> (X), <i>on</i> (Y, Z), <i>table</i> (Z), <i>cylinder</i> (X) $\rightarrow \begin{cases} \mathbf{0.68} & \textit{on}(X, Y), \neg\textit{inhand}(X) \\ 0.32 & \textit{noise} \end{cases}$
0.9	<b>PlaceCylinder</b> (X,Y): <i>inhand</i> (X), <i>on</i> (Y, Z), <i>table</i> (Z), <i>cylinder</i> (X) $\rightarrow \begin{cases} 0.0 & \textit{on}(X, Y), \neg\textit{inhand}(X) \\ 1.0 & \textit{noise} \end{cases}$	<b>PlaceCylinder</b> (X,Y): <i>inhand</i> (X), <i>on</i> (Y, Z), <i>table</i> (Z), <i>cylinder</i> (X) $\rightarrow \begin{cases} \mathbf{0.62} & \textit{on}(X, Y), \neg\textit{inhand}(X) \\ 0.38 & \textit{noise} \end{cases}$

Table 5.4: Comparison of the average state transition probability of the *PlaceCylinder* action. In each row, a different level of uniform noise is added to the starting scenarios, on which we optimized  $\mathcal{P}$ .

### 5.3 Optimizing Homing Action

Our last symbolic action, which we optimized, is the *Homing* action. The goal of this action is to return the robot to a predefined home position. We used the same start configurations as for the *PlaceCylinder* action and started the motion planner to find a plan towards the home position. This action has a smaller parameter space, compared to *GraspCylinder* and *PlaceCylinder*. Additionally, the manual parameters already showed successful results in each scenario. Figure 5.11 shows the results of the optimization procedure. The upper graph shows the *success rate*, which does not change, because we obtained a success in each scenario. The lower graph shows, that by using CMA, we could achieve a better gamma value in comparison to the manual set of parameters. Table 5.5 shows, that CMA also leads to a smaller  $\gamma$  value, but which is not statistically significant. Similar to the other two symbolic actions, we visualized the final trajectories of each scenario. Figure 5.12 and Figure 5.13, respectively, show that the difference for manual specified parameters (blue,dashed) is not visible in comparison to the optimized parameters (red,solid). Nevertheless we could decrease the number



of iterations to converge, which leads to the better  $\gamma$  value. To show, that we can also use *Homing* to generalize, we conducted the same experiment as for the *GraspCylinder* and *PlaceCylinder* action. For each noise level, we used the average over 540 scenarios to obtain the *success rate*. The optimized parameters have a slightly increased *success rate*, compared to manual parameters, as depicted in Figure 5.14 and in the *success rate* of the NID rules in Table 5.15.

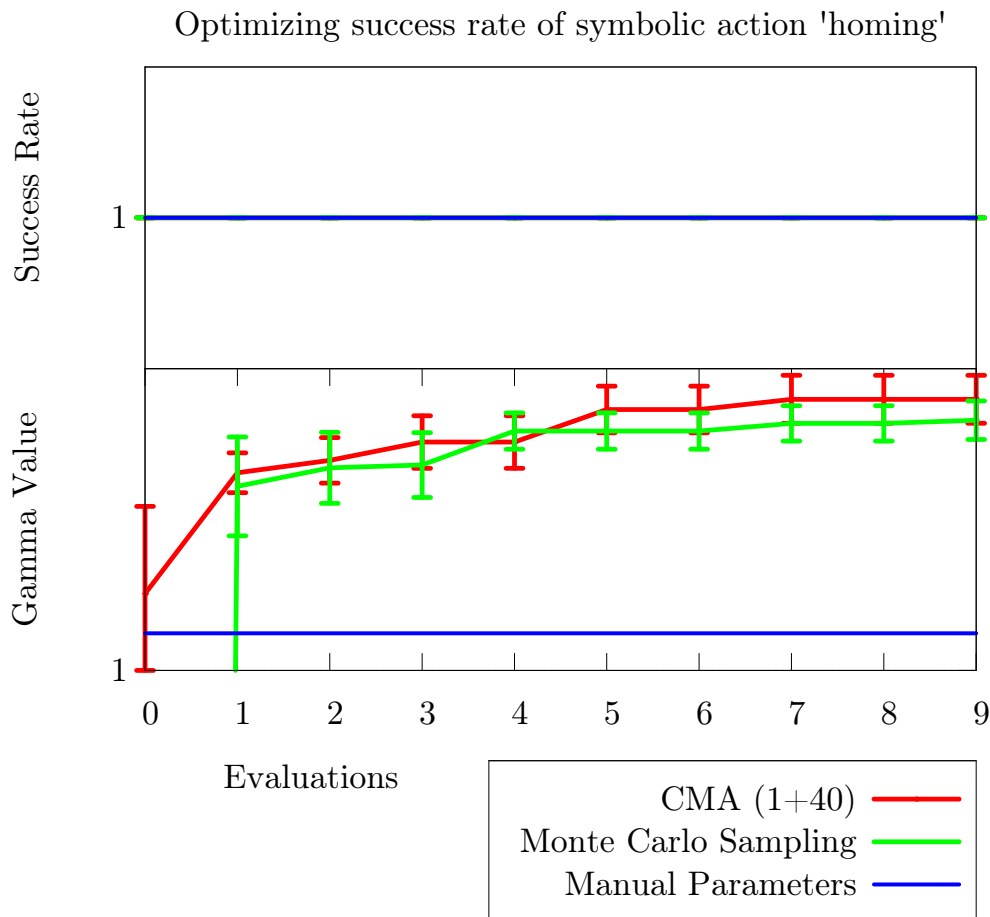


Figure 5.11: Homing experiment

Algorithm	$\gamma^{avg}$	$S_{grasp}^{avg}$
Optimized parameters (10 trials)	<b>0.58</b>	<b>1.00</b>
Monte Carlo Sampling (10 trials)	0.61	1.00
Manual specified parameters	0.93	1.00

Table 5.5: Comparison between optimized, manual and random parameters for the *Homing* experiment

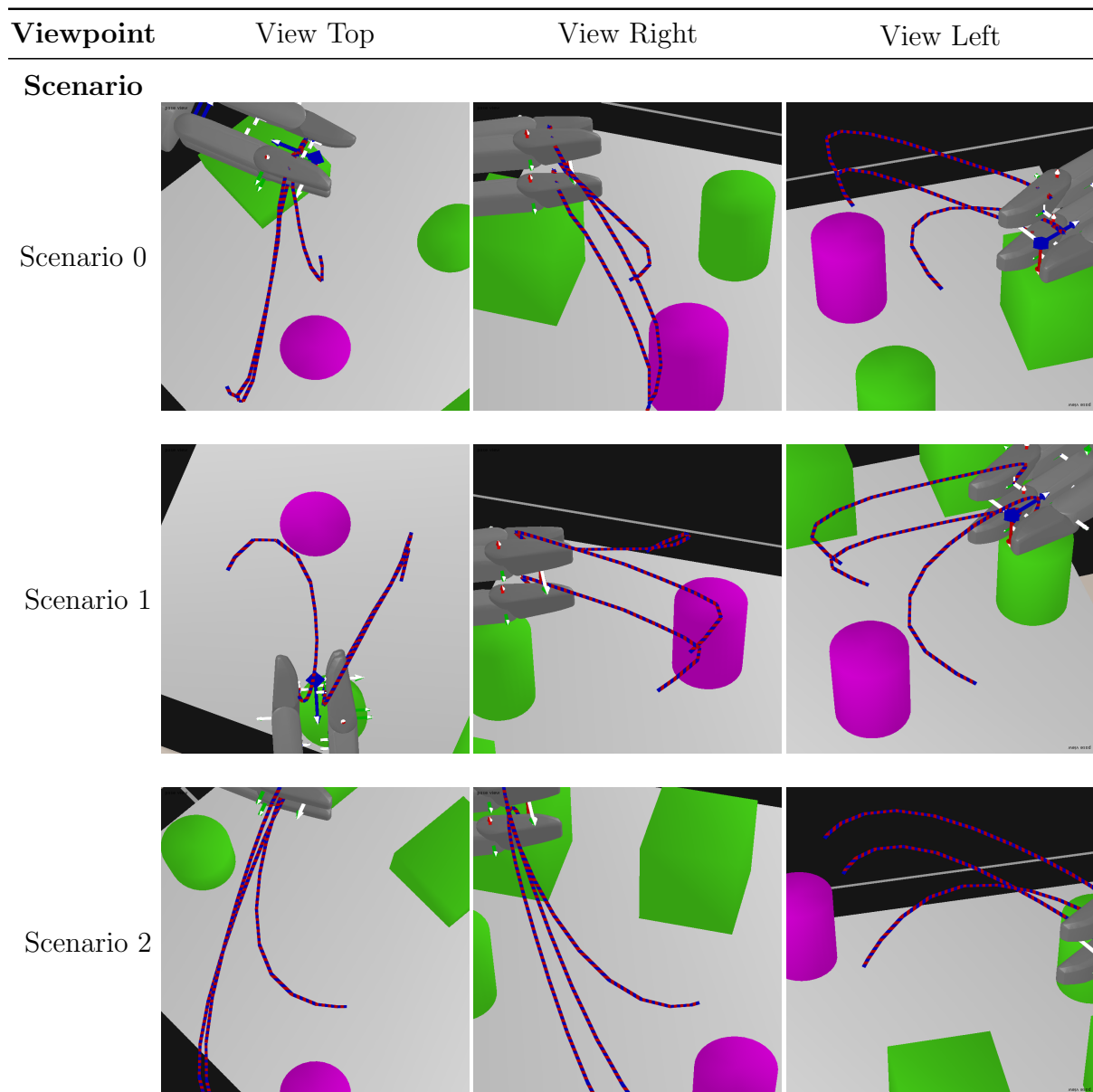


Figure 5.12: Comparison between the final trajectories, computed by using the CMA optimized parameters for the symbolic homing action (red, solid), and the manually specified parameters (blue, dashed). Both sets of parameters lead to a successful state transition, while the difference is not visible from the trajectories.

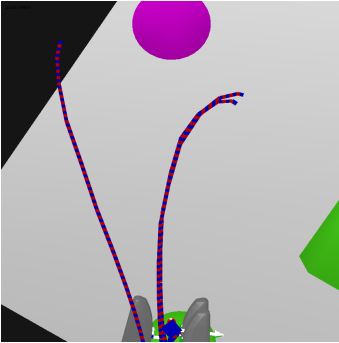
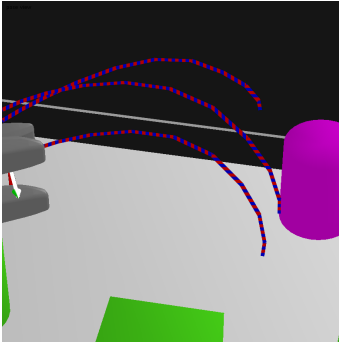
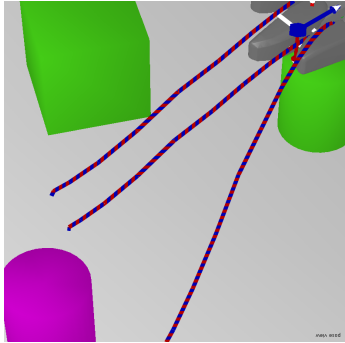
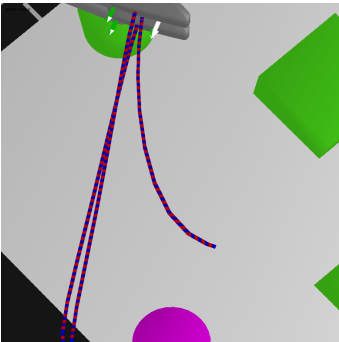
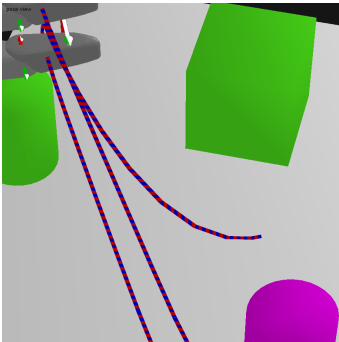
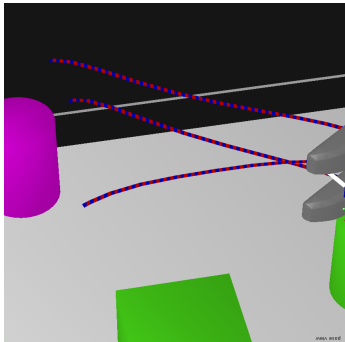
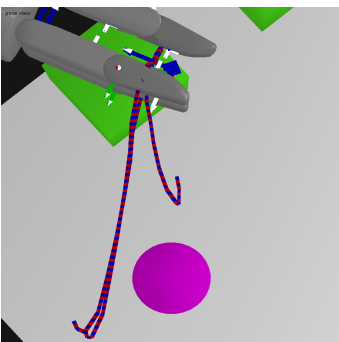
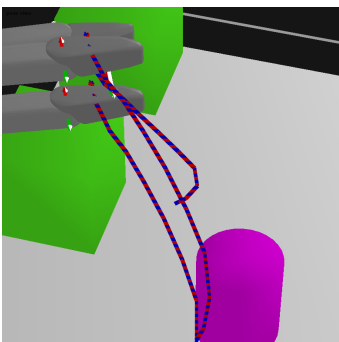
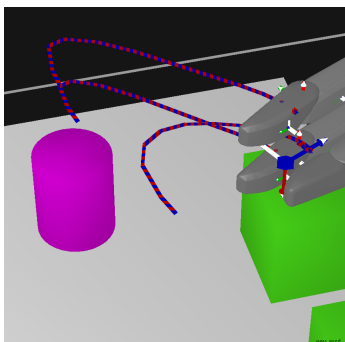
Viewpoint	View Top	View Right	View Left
Scenario			
Scenario 3			
Scenario 4			
Scenario 5			

Figure 5.13: The last three scenarios, which similarly to Figure 5.12 show identical trajectories obtained by manual parameters (blue,dashed) and optimized ones (red,solid).

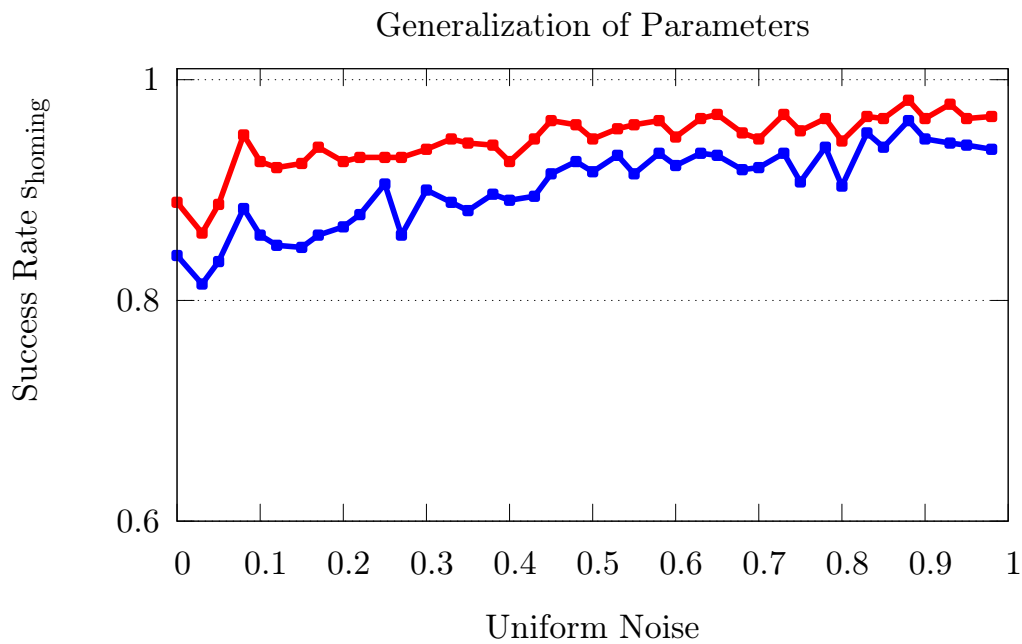


Figure 5.14: Generalization to unseen scenarios. Starting from the initial scenarios, we add an uniform noise to the objects in the scene, and measure the *success rate* on 540 scenarios. On average, the optimized parameters have a slight advantage, compared to manual specified parameters.

Uniform Noise	Symbolic NID Rules	
	With Manual $\mathcal{P}$	With Optimized $\mathcal{P}^*$
0.0	<b>Homing:</b> $\neg \text{home}(R), \text{robot}(R)$ $\rightarrow \begin{cases} 0.84 & \text{home}(R) \\ 0.16 & \text{noise} \end{cases}$	<b>Homing:</b> $\neg \text{home}(R), \text{robot}(R)$ $\rightarrow \begin{cases} \mathbf{0.89} & \text{home}(R) \\ 0.11 & \text{noise} \end{cases}$
0.5	<b>Homing:</b> $\neg \text{home}(R), \text{robot}(R)$ $\rightarrow \begin{cases} 0.92 & \text{home}(R) \\ 0.08 & \text{noise} \end{cases}$	<b>Homing:</b> $\neg \text{home}(R), \text{robot}(R)$ $\rightarrow \begin{cases} \mathbf{0.95} & \text{home}(R) \\ 0.05 & \text{noise} \end{cases}$
0.9	<b>Homing:</b> $\neg \text{home}(R), \text{robot}(R)$ $\rightarrow \begin{cases} 0.95 & \text{home}(R) \\ 0.05 & \text{noise} \end{cases}$	<b>Homing:</b> $\neg \text{home}(R), \text{robot}(R)$ $\rightarrow \begin{cases} \mathbf{0.97} & \text{home}(R) \\ 0.03 & \text{noise} \end{cases}$

Figure 5.15: Comparison of the average state transition probability of the *Homing* action. In each row, a different level of uniform noise is added to the starting scenarios, on which we optimized  $\mathcal{P}$ . It can be seen, that on average, there is a slight advantage for the optimized  $\mathcal{P}^*$ .

## 5.4 Discussion of Results

We conducted an offline optimization of the *precision parameters*  $\alpha$  of a motion primitive  $\mathcal{P}$  for three symbolic actions: *GraspCylinder*, *PlaceCylinder* and *Homing*. We have seen, that CMA is able to find a robust set of *precision parameters*  $\alpha$  for each symbolic action, and that the final trajectories are indeed more successful to reach the desired outcomes, in comparison to a manual specified set of parameters. Given the best optimized  $\alpha$ , we conducted a second experiment, where we investigated, if  $\alpha$  is able to generalize to unseen scenarios. The outcome showed, that the optimization approach indeed leads to better generalization than manually specified ones.

The NID rules, which we obtained in these experiments can be directly incorporated into a symbolic planner. Each rule is optimized to achieve a higher probability for a desired outcome. This leads to more feasible symbolic plans, because the planner can rely on a set of more robust symbolic actions, which have been optimized in a physical environment.

# Chapter VI

## Conclusion

Planning a high level task is usually approached by using two separated planners. First, a symbolic planner, which can solve the high level task by breaking it down into smaller subtasks. Second, for each subtask, a motion planner is started to plan a feasible trajectory. We discussed in Section 3.1, that the abstraction of the symbolic level leads to several difficulties, which in turn lead to a lower probability to find a feasible trajectory for a specific symbolic state transition. Our approach, which we introduced in Chapter IV focuses on integrating both planning levels by computing an approximate optimal motion primitive  $\mathcal{P}^*$ , which increases a desired state transition probability. This motion primitive is based on two variables, first the constraints of the motion planner, and second, the importance of each constraint. Given a motion primitive, the underlying motion planner can compute an optimal trajectory, which is more robust against noise in comparison to a non optimal, feasible trajectory. The goal of this thesis was to find  $\mathcal{P}^*$ , the approximate optimal motion primitive, which could increase the desired *success rate* of a symbolic action. The *success rate* was defined for each symbolic action, as the desired state transition probability.

To measure different  $\mathcal{P}$  and compare them to each other, we defined a mapping from a motion primitive  $\mathcal{P}$ , of a symbolic action, onto a single scalar value  $\gamma$ , which was inversely proportional to its *success rate*. This global mapping was defined as a concatenation of internal mappings, which we defined in Chapter IV. We started by defining a mapping from  $\mathcal{P}$  to a trajectory  $q$ , which is obtained by invoking a motion planner. The input of the motion planner is given by a constant start configuration, constant goal configuration, and a specific scenario, on which we evaluate  $\mathcal{P}$ . Afterwards, we devised a mapping from trajectory  $q$  to a set of variables  $\beta$ , which measures the trajectory in terms of physical variables like avoidance of collisions, limits or the symbolic success rate. Eventually, a function from  $\beta$  to  $\gamma$  then maps all measured variables to a scalar value  $\gamma$ , which describes the goodness of a specific  $\mathcal{P}$ . Based on the mappings, we defined an optimization problem for the *precision parameters* of  $\mathcal{P}$ , which defines the importance of the involved constraints. By using covariance matrix adaption (CMA), we could demonstrate, that for each symbolic action, an approximate optimal  $\mathcal{P}^*$  can be found, which increases the desired state transition probability on a fixed set of scenarios. Finally, we added different uniform noise to the fixed set of

scenarios, and measured the *success rate* of the symbolic actions. This demonstrated that even in random scenarios, the optimized motion primitive can lead to a better *success rate*, in comparison to manually specified primitives.

Altogether it is evident, that an integration of symbolic planning and motion planning can indeed lead to more robust and feasible trajectories, which are able to increase the state transition probability of a desired symbolic outcome. This work can be seen as a first step towards finding the optimal constraints of a motion planner, which in turn increase the performance of the symbolic planner. We therefore conclude this thesis by showing possible future research directions.

## 6.1 Future Research Directions

There are several possible tracks to advance the presented optimization approach to motion primitives. An important aspect of all tracks will be to optimize primitives offline, and using them for future planning tasks.

Our first future research question affects our optimization algorithm. To optimize  $\mathcal{P}$ , we used an elitist CMA algorithm, which consisted of a population of 1  $\mu$  plus 40  $\lambda$ . An interesting question is, how we can speed up the optimization, by using different  $\mu$  and  $\lambda$  combinations.

In Chapter V, we demonstrated that the optimized parameters can generalize to unseen scenarios. But it is still an open question, about how we can increase the *success rate* of a symbolic action in unseen scenarios, i.e. how we can better generalize. To generalize, it is important to introduce a regularization function, in order to prevent overfitting. An important research topic is how we can define such a regularization function, and how this changes our mappings.

Besides changing the algorithm and introducing regularization, we can also change the input to our objective function. Optimizing  $\mathcal{P}$  was based on a static set of tasks. But it is also possible to introduce a dynamic set of tasks, for example by manipulating the task targets or including all possible tasks available. This would require us to introduce new variables, which could directly be incorporated into the mappings we devised.

Besides dynamic tasks, another important aspect is to concentrate on feedback plans. While we generally assumed a static environment, a feedback plan could include perceptual variables, which directly act on dynamic changes in the environment. An optimization of  $\mathcal{P}$  for feedback plans could additionally improve the number of feasible trajectories, and therefore lead to a higher *success rate*.

Finally, we note that our approach can only find the approximate optimal  $\mathcal{P}^*$  due to the fact that we do not have a guarantee to find the global optimum by using CMA. An important future research question will therefore be, if we can find an algorithm,

which can guarantee to find the global optimum for a fixed set of scenarios.



# Bibliography

- Auger, A. and Hansen, N. (2005). A restart cma evolution strategy with increasing population size. In *Congress on Evolutionary Computation*, pages 1769–1776. IEEE.
- Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G. J. (2007). Symbolic planning and control of robot motion: Finding the missing pieces of current methods and ideas. *Robotics and Automation Magazine*, 14(1):61–70.
- Bretl, T., Lall, S., Latombe, J.-C., and Rock, S. (2004). Multi-step motion planning for free-climbing robots. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.
- Cambon, S., Alami, R., and Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *Int. J. Rob. Res.*, 28(1):104–126.
- Choi, J. and Amir, E. (2009). Combining planning and motion planning. In *ICRA*, pages 238–244.
- Cohen, B. J., Chitta, S., and Likhachev, M. (2010). Search-based planning for manipulation with motion primitives. In *ICRA*, pages 2902–2908. IEEE.
- Cohen, B. J., Subramania, G., Chitta, S., and Likhachev, M. (2011). Planning for manipulation with adaptive motion primitives. In *ICRA*, pages 5478–5485. IEEE.
- Craig, J. J. (1989). *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.
- Dornhege, C., Eyerich, P., Keller, T., Brenner, M., and Nebel, B. (2010). Integrating task and motion planning using semantic attachments. In *Bridging the Gap Between Task and Motion Planning*, volume WS-10-01 of *AAAI Workshops*. AAAI.
- Fainekos, G. E., Girard, A., Kress-Gazit, H., and Pappas, G. J. (2009). Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352.
- Fikes, R. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *IJCAI*, pages 608–620.
- Hansen, N. (2006). The CMA evolution strategy: a comparing review. In Lozano, J., Larranaga, P., Inza, I., and Bengoetxea, E., editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer.

- Hansen, N. and Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. In Yao, X. et al., editors, *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*, pages 282–291. Springer.
- Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *International Conference on Evolutionary Computation*, pages 312–317.
- Hauser, K. and Latombe, J.-C. (2009). Integrating task and prm motion planning: Dealing with many infeasible motion planning queries. In *Workshop on Bridging the Gap Between Task and Motion Planning*. ICAPS.
- Hauser, K. K., Bretl, T., Harada, K., and Latombe, J.-C. (2006). Using motion primitives in probabilistic sample-based planning for humanoid robots. In Akella, S., Amato, N. M., Huang, W. H., and Mishra, B., editors, *WAFR*, volume 47 of *Springer Tracts in Advanced Robotics*, pages 507–522. Springer.
- Hespanha, J. P. (2009). *Linear Systems Theory*. Princeton Press, Princeton, New Jersey.
- Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002a). Learning rhythmic movements by demonstration using nonlinear oscillators. In *In Proceedings of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS2002)*, pages 958–963.
- Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002b). Movement imitation with nonlinear dynamical systems in humanoid robots. In *In IEEE International Conference on Robotics and Automation (ICRA2002)*, pages 1398–1403.
- Kaelbling, L. P. and Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *ICRA*, pages 1470–1477.
- Kober, J., Mülling, K., Kroemer, O., Lampert, C. H., Schölkopf, B., and Peters, J. (2010). Movement templates for learning of hitting and batting. In *ICRA*, pages 853–858.
- Kober, J. and Peters, J. (2009). Learning motor primitives for robotics. In *ICRA*, pages 2112–2118.
- Kober, J. and Peters, J. (2011). Learning elementary movements jointly with a higher level task. In *IROS*, pages 338–343.
- Koenig, S. (2010). Creating a uniform framework for task and motion planning: A case for incremental heuristic search? In *ICAPS*, pages 254–258.
- Lang, T. (2011). *Planning and Exploration in Stochastic Relational Worlds*. PhD thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin.
- Lang, T. and Toussaint, M. (2009). Approximate inference for planning in stochastic relational worlds. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 585–592.

- Lang, T. and Toussaint, M. (2010a). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39:1–49.
- Lang, T. and Toussaint, M. (2010b). Probabilistic backward and forward reasoning in stochastic relational worlds. In *Proc. of the Int. Conf. on Machine Learning (ICML)*.
- LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K.
- LaValle, S. M. (2011a). Motion planning: The essentials. *IEEE Robotics and Automation Society Magazine*, 18(1):79–89.
- LaValle, S. M. (2011b). Motion planning: Wild frontiers. *IEEE Robotics and Automation Society Magazine*, 18(2):108–118.
- Loeliger, H.-A. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, 21(1):28–41.
- Meier, F., Theodorou, E., Stulp, F., and Schaal, S. (2011). Movement segmentation using a primitive library. In *International Conference on Intelligent Robots and Systems (IROS)*.
- Park, D.-H., Hoffmann, H., Pastor, P., and Schaal, S. (2008). movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *IEEE International Conference on Humanoid Robots, 2008*.
- Pasula, H. M., Zettlemoyer, L. S., and Kaelbling, L. P. (2004). Learning probabilistic relational planning rules. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*.
- Pasula, H. M., Zettlemoyer, L. S., and Kaelbling, L. P. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352.
- Peters, J., Mülling, K., Kober, J., Nguyen-Tuong, D., and Krömer, O. (2009). Towards motor skill learning for robotics. In *ISRR*, pages 469–482.
- Plaku, E. and Hager, G. D. (2010). Sampling-based motion and symbolic action planning with geometric and differential constraints. In *ICRA*, pages 5002–5008.
- Rawlik, K., Toussaint, M., and Vijayakumar, S. (2010). Approximate inference and stochastic optimal control. *CoRR*, abs/1009.3958.
- Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.
- Sucan, I. A. and Kavraki, L. E. (2011a). Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *ICRA*, pages 5492–5498.

- Sucan, I. A. and Kavraki, L. E. (2011b). On the advantages of task motion multigraphs for efficient mobile manipulation. In *IROS*, pages 4621–4626.
- Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. Intelligent robotics and autonomous agents. The MIT Press.
- Toussaint, M. (2009). Robot trajectory optimization using approximate inference. In *ICML*, page 132.
- Toussaint, M., Gienger, M., and Goerick, C. (2007). Optimization of sequential attractor-based movement for compact behaviour generation. *Humanoid Robots 2007 7th IEEE/RSJ International Conference on*, pages 122–129.
- Toussaint, M. and Goerick, C. (2010). A bayesian view on motor control and planning. In *From Motor Learning to Interaction Learning in Robots*, pages 227–252.
- Wolfe, J., Marthi, B., and Russell, S. (2010). Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling*.
- Younes, H. L. S. and Littman, M. L. (2004). Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical report, Carnegie Mellon University, Pittsburgh, PA.

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den

.....

Unterschrift