Institute for Parallel and Distributed Systems
Machine Learning and Robotics Lab

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit No. 0158

Policy Search for Imitation Learning

Andreas Doerr

Course of Study:	Computer Science B.Sc.
Examiner:	Prof. Dr. rer. nat. Marc Toussaint
Supervisor:	Ph.D. Nathan Ratliff
Commenced:	July 9, 2014
Completed:	January 9, 2015
CR-Classification:	G.1.6, I.2.6, I.2.9

Abstract

Efficient motion planning and possibilities for non-experts to teach new motion primitives are key components for a new generation of robotic systems. In order to be applicable beyond the well-defined context of laboratories and the fixed settings of industrial factories, those machines have to be easily programmable, adapt to dynamic environments and learn and acquire new skills autonomously.

Reinforcement learning in principle solves those learning issues but suffers from the curse of dimensionality. When dealing with complex environments and highly agile hardware platforms like humanoid robots in large or possibly continuous state and action spaces, the reinforcement framework becomes computationally infeasible. In recent publications, parametrized policies have been employed to face this problem. One of them, *Policy Improvement with Path Integrals* (PI^2), has been derived from the transformation of the *Hamilton-Jacobi-Bellman* (HJB) equation of stochastic optimal control into a path integral using the Feynmann Kac theorem. Applications of PI^2 are so far limited to *Dynamic Movement Primitives* (DMP) to parametrize the motion policy. Another policy parametrization, the formulation of motion primitives as solution of an optimization-based planner has been widely used in other fields (e.g. inverse optimal control) and offers compelling possibilities to formulate characteristic parts of a motion in an abstract sense without specifying too much problem-specific geometry.

Imitation learning or learning from demonstration can be seen as a way to bootstrap the acquisition of new behavior and as an efficient way to guide the policy search into a desired direction. Nevertheless, due to imperfect demonstrations, which might be incomplete or contradictory and also due to noise, the learned behavior might be insufficient. As observed in the animal kingdom, a final trial-and-error phase guided by the cost and reward of a specific behavior is necessary to obtain a successful behavior. Interestingly, the reinforcement learning framework might offer the tools to govern both learning methods at the same time. Imitation learning can be reformulated as reinforcement learning under a specific reward function, allowing the combination of both learning methods.

In this work, the concept of probability-weighted averaging of policy roll-outs as seen in PI^2 is combined with an optimization-based policy representation. The reinforcement learning toolbox and direct policy search is utilized in a way that allows both imitation

learning based on arbitrary demonstration types and the imposition of additional objectives on the learned behavior. A black box evolutionary algorithm, *Covariance Matrix Adaptation Evolutionary Strategy* (CMA-ES), which can be shown to be closely related to the approach in PI^2 is leveraged to explore the parameter space. This work will experimentally evaluate the suitability of this algorithm for learning motion behavior on a humanoid upper body robotic system. We will focus on learning from different types of demonstrations. The formulation of the reward function for reinforcement learning will be depicted and multiple test scenarios in 2D and 3D will be presented. Finally, the capability of this approach to learn and improve motion primitives is demonstrated on a real robotic system within an obstacle test scenario.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Today's demand to automatize perseverative or possibly dangerous tasks has lead to an ongoing development of complex machinery and robotics hardware geared to operate in environments originally shaped for human interaction. One of those systems, the CMU[1] Highly Intelligent Mobile Platform (CHIMP), which is designed to accomplish tasks within dangerous and degraded environments of disaster scenarios is shown in figure 1.1. Due to their high Degree of Freedom (DoF), the programming of these anthropomorphic or humanoid robotic systems can become tedious or even impossible. So far, the programming of complex motor skills, allowing these machines to act and interact within a dynamic environment, has been mostly accomplished by human experts. This lead to fixed and therefore highly specialized behavior only suitable for very specific problems and required exhaustive manual parameter tuning. To keep track with the hardware development and modern robotic systems, it would be desirable to facilitate the development of new systems by enabling them to learn new skills from demonstrated behavior and to improve those skills autonomously, e.g. by trial-and-error.

When looking into the animal kingdom, two methods to acquire new skills can be distinguished. Learning is achieved by either imitating behavior observed from some other subject or by maximizing some sort of reward when randomly exploring possible actions. Both concepts are known in the robotics community and translate into Learning from Demonstration (LfD) [**?**] and Reinforcement Learning (RL) [**?**], respectively.

LfD can be seen as supervised learning. An external teacher offers demonstrations which can be used as training examples in an off-line learning process. The demonstrations may shorten the learning time by guiding the process into a specific direction. At the same time, demonstrations may not provide a complete picture of the skill to be learned. Additionally, the sensor values upon which the observation of the behavior is based may be noisy and the demonstrations themselves might contain noise and uncertainty. Because of all those sources of uncertainty and as the environment itself is non-deterministic, skills learned solely from demonstrations are most likely not sufficient without further

[1] Carnegie Mellon University

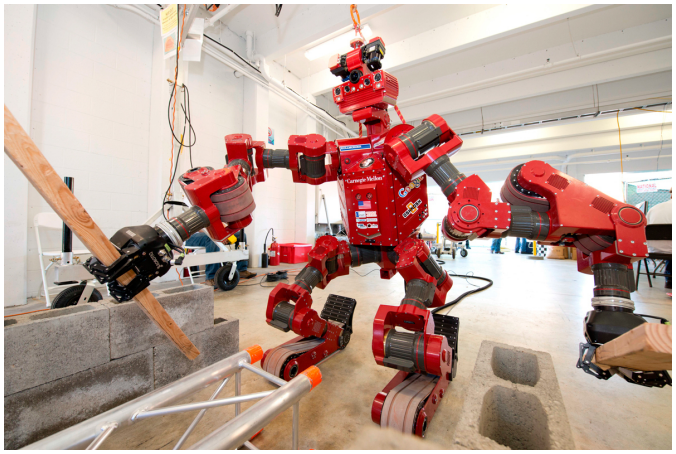


Figure 1.1: The CMU Highly Intelligent Mobile Platform (CHIMP), an anthropomorphic robot, has been built to compete in the DARPA Robotics Challenge. Because of its form factor, strength and dexterity, it can execute human-level tasks in dangerous and degraded environments[2].

adaptations. Moreover, LfD requires the learner to generalize from the restricted set of observed behavior, i.e. to learn the characteristics of a motion.

For reinforcement learning, the learner has to explore a huge space of possible solutions when dealing with high DoF systems and continuous state/action spaces. While the optimal solution is somewhere in this space, the current solution and the reward will not necessarily guide the learning process into the globally optimal direction. A more detailed analysis of recent developments within the field of reinforcement learning and the combination of both learning techniques will be given in the next chapter 2.

1.2 Goals

This work will propose a novel and robust algorithm for learning motion policies from arbitrary demonstrations on continuous state/action space and high DoF robotic systems. At the same time this method exposes only few open learning parameters and should be easily applicable to different types of motion problems and demonstration types. A probability-weighted averaging similar to PI^2 is used but augmented by the improved covariance matrix update rule from CMA-ES. In order to incorporate the PI^2-CMA approach as proposed by [**?**], an optimization-based policy representation will be used. This allows to define more abstract and less geometry-oriented motion characteristics

[2]Credit: Nick Letwin, http://thetartan.org/2014/1/20/scitech/chimp

within the optimal control objective. A high-level objective function will be formulated to leverage reinforcement learning for imitation learning based on one or multiple demonstrations. The stochastic evolutionary strategy as introduced by PI^2 and CMA-ES at the same time allows almost arbitrary formulations for this objective function. No derivative information will be necessary and no derivative information will be estimated.

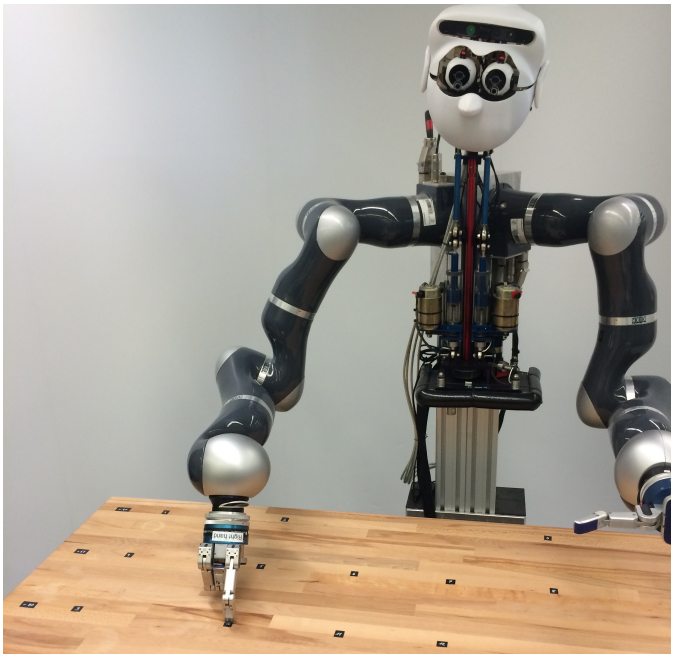


Figure 1.2: The humanoid robot Apollo is used in this work as a real-world testbed for reinforcement learning of motion primitives.

The humanoid upper body robot Apollo, as shown in figure 1.2 is used as a running example. This robot is made of two 7 DoF KUKA lightweight arms, two 4 DoF Barrett hands and a SARCOS humanoid head. The algorithm will be used to teach the robot several pointing and reaching motions in order to demonstrate the method's ability to generalize a motion policy from different types of demonstrated behavior and to further improve this policy.

1.3 Organization of this Thesis

Following this short introduction into the methods and reasons for learning motor primitives and the possibilities to facilitate the programming of modern robotic systems, a short non-comprehensive presentation of related work in the field of motion learning and representation is given in chapter 2. The recent developments in reinforcement learning

for continuous state/action spaces and the two most common motion policy representations are outlined. One of these frameworks, the optimal control inspired motion control, is utilized in this work as the underlying policy representation for the reinforcement policy search and consecutively to imitate and improve demonstrated behavior. The theoretical foundations of this concept are laid out in chapter 3. In order to motivate the use of this approach and to evaluate its performance, several experiments are conducted, ranging from 2D simulations to motion tasks executed on a real humanoid robotic system within an environment containing multiple obstacles. The experimental setup is explained in chapter 4. From the experimental results, as presented in chapter 5, the applicability of the approach to imitation learning of motor policies and also to shape the motion in a way to fulfill additional objectives can be shown, even on the high-dimensional continuous space of this problem formulation. This work is concluded in chapter 6 by summarizing the capabilities of the presented algorithm and by pointing out potential future work not addressed by this thesis.

2 Background and Related Work

In recent work, new methods have been developed within the reinforcement learning framework to deal with continuous state/action spaces. The most common of these algorithms are based on direct policy search. In contrast to the classic value-function approximation, direct policy search is better scalable to high-dimensional problems and many of those algorithms require fewer open hyperparameters. All of those methods utilize some sort of parametrized policy representation. The search space of possible policies is therefore reduced by more or less implicit assumptions contained in the policy representation.

Gradient-based methods like Inverse Optimal Control (IOC) [**?**], REINFORCE [**?**] and Natural Actor-Critic (NAC) [**?**] aim at iteratively updating a parametrized policy by estimating a gradient in parameter space. Based on the original work by [**?**] a new approach has been published recently which evolved from value-function approximation and stochastic optimal control. It has been shown that policy improvement can be turned into the estimation of a path integral with no tuning parameters except for exploration noise. This method, called Policy Improvement with Path Integrals (PI^2), is based on the stochastic Hamilton-Jacobi-Bellmann (HJB) equation and has been shown to be numerically robust due to the lack of matrix inversions or gradient approximations. At the same time this method has been empirically tested and shown to perform better than gradient-based methods even on complex real-world robotic scenarios [**?**].

In [**?**] a comparison of Policy Improvement with Path Integrals (PI^2), Cross-Entropy Methods (CEM) [**?**] and Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) [**?**] revealed the similarity of those algorithms. Despite being derived from different ideas and principles they all utilize a very similar probability-weighted averaging to update the policy parameters. As a combination of PI^2 and CMA-ES, the new algorithm PI^2-CMA is proposed by [**?**]. The main advantage over the original PI^2 is the automatic adaptation of the exploration noise magnitude in CMA-ES.

Policy learning has been successfully applied to many real-world applications. Machine learning has been used to learn policies to fly helicopters [**?**] [**?**], for quadrupedal locomotion [**?**] and also biped robots.

The underlying representation of the parametrized policy is of great importance since many implicit assumptions are incorporated in the specific representation, thus reducing the number of representable policies. The two most common approaches representing the policy as Dynamic Movement Primitives and as the solution of an optimal control problem are described in detail in the next sections.

2.1 Dynamic Movement Primitives

Dynamic Movement Primitives (DMP) have been proposed by [**?**] to express parametrized motion policies while maintaining certain robustness and stability criteria. Dynamic systems are used to express a spring-damper system which follows an attractor landscape given by a second dynamic system. The attractor surface is usually learned from some given motions by e.g. locally weighted regression. Single strokes but also rhythmic tasks can be formulated using DMPs.

Basic stability properties are inherent to the first dynamic system. The entire policy can be formulated to be linear in its parameters and can be rescaled in time, goal and amplitude. This concept has been used in multiple applications including T-ball batting [**?**], tennis swings [**?**], constrained reaching tasks [**?**] and also industrial applications.

The concept is mainly focused on learning by imitation but can be adopted to subsequent self-improvement and also dynamic re-planing, e.g. to react to dynamic obstacles cf. [**?**]. Due to the representation of the basic motion shape as an attractor landscape, the learned policy is centered around a specific geometric version of a motion primitive defined in the 3D euclidean space. No relation between the geometric and the characteristic aspect causing especially this type of behavior is given in this problem formulation. While this formulation might be sufficient for small and restricted motion problems (e.g. tennis swing), a policy representation expressing the necessity for different characteristics within the motion might be more useful for other problems.

2.2 Optimal Control for Motor Primitives

One possibility to link the typical aspects of a desired motion and the corresponding control outputs $\mathbf{u}_t$ of a system is given by the optimal control framework. Optimal control is based on the idea to compute the control outputs for a given system such that a given cost or objective function c_t at time step t is minimized. The objective function can be formulated in a way that explicitly expresses desired characteristics of a behavior such as smoothness, obstacle avoidance, acceleration and velocity limits and so

on. Therefore, the objective function is made of (possibly time-dependent) intermediate cost terms and a terminal cost term. Abstractly, this can be formulated as

$$\min_{\mathbf{u}_{0:T},\mathbf{s}_{0:T}} \sum_{t=1}^{T-1} c_t(\mathbf{s}_t, \mathbf{u}_t) + c_T(\mathbf{s}_T) \tag{2.1}$$

$$\text{s.t.} \quad \mathbf{s}_{t+1} = \mathbf{f}(\mathbf{s}_t, \mathbf{u}_t)\,. \tag{2.2}$$

In this discrete version, the manipulation of the system's state $\mathbf{s}_t$ is described by a trajectory $\xi = \{(\mathbf{s}_0, \mathbf{u}_0), \ldots, (\mathbf{s}_T, \mathbf{u}_T)\}$ which is a sequence of $T+1$ state-action pairs. This trajectory is the solution of an optimization problem over the space of all feasible trajectories denoted by $\Xi(\boldsymbol{\gamma})$ under the system dynamics $\mathbf{s}_{t+1} = \mathbf{f}(\mathbf{s}_t, \mathbf{u}_t)$. Most times, the desired behavior is shaped by multiple influences which can be expressed as sum over l intermediate and m terminal cost features. Since each feature may incorporate context dependencies and free hyperparameters, the individual features are modeled as functions of the current context $\boldsymbol{\gamma}$ and a hyperparameter vector $\boldsymbol{\theta}$. The context may be some representation of the environment surrounding the robot (e.g. obstacles) as well as problem-dependent parameters (e.g. start and goal configurations). Therefore, the optimal control problem can be reformulated as

$$\min_{\xi \in \Xi(\gamma)} \sum_{t=1}^{T-1} \left(\sum_{i=1}^{l} c_t^{(i)}(\mathbf{s}_t, \mathbf{u}_t, \boldsymbol{\gamma}, \boldsymbol{\theta}) \right) + \sum_{i=1}^{m} c_T^{(i)}(\mathbf{s}_T, \boldsymbol{\gamma}, \boldsymbol{\theta})\,. \tag{2.3}$$

While previous approaches such as IOC [**?**] have been centered around linear combinations of fixed feature terms, a more general approach allows arbitrary hyperparameters also within each feature formulation. The major drawback of this formulation is the complexity of finding an appropriate composition of the objective function. Even if the characteristic parts and therefore the cost features of a motion are easy to identify, there may be many parameters to tune in order to adapt the objective function in a way to really express the desired behavior. This process used to be solved manually by tuning all free parameters to achieve an acceptable behavior of the system.

For linear combinations of features, this problem has been shown to be reducible to an extremely large-scale multiclass classification problem [**?**]. In this work a framework will be developed (cf. section 3) and tested (cf. section 5) to face this problem in its general formulation for arbitrary, non-linear parametrization.

3 Theoretical Foundations

In most cases we have an intuitive understanding of what is a desirable motion for some manipulation task. We can instinctively specify characteristic parts of this motion. These characteristic parts can be direct restrictions to the configuration, speed or acceleration of the manipulator or requirements with respect to its position within the surrounding environment. Typically, a certain smoothness of the motion or the avoidance of obstacles within a certain radius is required. These claims can be formulated as cost features within an objective function. This concept of optimal control, as formulated in section 2.2, is widely used to represent motion policies. The individual motion, in most cases represented by a discrete-time trajectory, is generated by solving the optimal control problem, which means minimizing the objective function for a given scenario.

3.1 Representation of the Motion Policy

The representation of motion policies in the form described above offers a high degree of freedom to the user not only with respect to the composition of the objective function of all kinds of different cost features, but also by the possibility to individually weight each cost term and to tune additional parameters within the cost terms. The problem of selecting a sufficient number and the correct type of cost features will not be addressed by this work. In most cases, the required motion type directly relates to an obvious set of features which specifies the typical parts of this motion. The second part, namely weighting the features appropriately and selecting the hyperparameters in a goal-oriented way will be addressed in this chapter from a theoretical point of view. First of all, some possible feature representations are discussed, followed by the description of an optimization framework that allows to automatically tune these parameters to imitate demonstrated behavior and to subsequently improve the motion with respect to additional goals.

A schematic description of the motion policy representation is shown in figure 3.1. According to a set of parameters $\boldsymbol{\theta}$ and the current context $\boldsymbol{\gamma}$, a unique, discrete trajectory $\xi = (\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_T)$ is computed. The variable $\mathbf{q}_i$ denotes the full state of the robotic system at the discrete time step i. In practice, it usually consists of all current joint

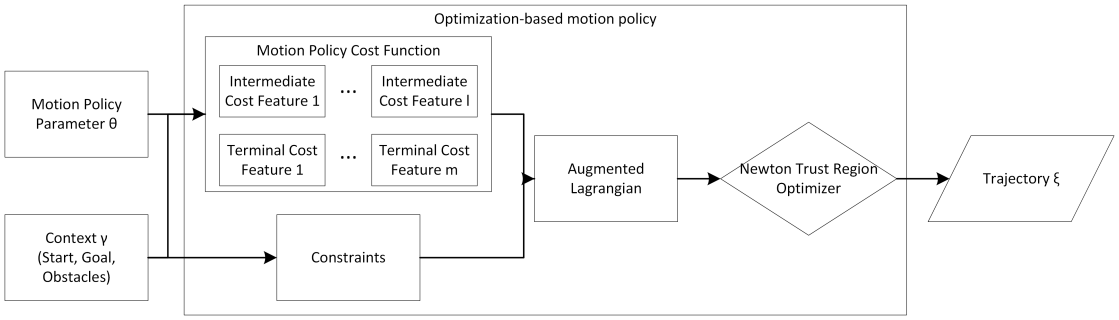


Figure 3.1: Components of the optimization-based representation of the parametrized motion policy.

positions. The control command $\mathbf{u}_i$ needed to proceed from state $\mathbf{q}_i$ to $\mathbf{q}_{i+1}$ is implicitly given by the actions necessary to achieve this next configuration within one time step. Usually an underlying low-level control is employed to execute the discrete trajectory.

The trajectory is generated by solving the optimization problem given by the objective function. Additional constraints can be imposed that represent joint or acceleration limits. In this work, the augmented Lagrangian method is used to transform the constrained optimization problem into an unconstrained one, which is then solved using a Newton Trust Region optimizer. The entire optimizer setup and implementation is part of a preset framework and will be considered as a black box policy representation except for the cost function formulation which will be explained in the following.

The optimization problem underlying the motion policy representation can be solved numerically in an efficient and fast manner since gradient and also second derivative information can be computed analytically for all cost feature terms used in this work. Depending on the cost features and the problem setting, the average computation time for one trajectory is around 100 ms to 2,000 ms. In the following the cost features as used in the experimental part of this work are explained.

3.1.1 Motion Objective Cost Features

If not denoted otherwise, all cost features employed in this work are constant over time. The features are defined over the $3 \cdot n$ dimensional space of the current state $\mathbf{q}$ ($\mathbf{q} \in \mathbb{R}^n$) and its first ($\dot{\mathbf{q}}$) and second ($\ddot{\mathbf{q}}$) time derivative. Kinematic maps $\phi(\mathbf{q})$ of the robotic system are employed to pull back cost terms defined over euclidean key-points on the robot (e.g. the end-effector position) into this configuration space. The first and second order derivatives are pulled back through these maps as well. The complete policy

parametrization is given by a parameter vector $\boldsymbol{\theta}$ composed of scalar parameters which are in the following individually identified by their indices (e.g. θ_{joint_deriv}).

Penalties for Task and Joint Space Derivatives

The fundamental requirement for every motion policy must be to avoid erratic and jerky behavior. One possibility to achieve smooth motions is to penalize high velocities and accelerations. The velocities and accelerations in joint space are computed using finite differencing such that a cost term can be formulated as follows:

Penalty for Joint Space Derivatives The penalty factors for velocity and acceleration have been set to $\alpha_{vel} = 1.0$ and $\alpha_{acc} = 100.0$. This joint space derivative penalty has been incorporated for all intermediate time steps and a modified version ($\alpha_{vel} = 300,000.0$ and $\alpha_{acc} = 100.0$) has been chosen for $t = 0$ to achieve an initial velocity close to zero. All joint space derivative terms are multiplied by the policy parameter θ_{joint_deriv}. The internal parameters α_{vel} and α_{acc} are currently not part of the policy parametrization.

$$c_{joint\_deriv}(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \theta_{joint\_deriv} \cdot \left(\alpha_{vel} \|\dot{\mathbf{q}}\|^2 + \alpha_{acc} \|\ddot{\mathbf{q}}\|^2 \right) \tag{3.1}$$

Penalty for Task Space Derivatives On Apollo's robot arm, a smooth motion in joint space results in curved end-effector motions in the euclidean workspace since all joints are revolute ones. A penalty term for the second derivative is therefore introduced which affects the task space velocity of points on the robot. From the robot's kinematic map, the function $\phi_{end-effector}$ mapping from configuration $\mathbf{q} \in \mathbb{R}^n$ to euclidean task space position $\mathbf{x} \in \mathbb{R}^3$ is given. A similar cost feature as discussed above is used to penalize end-effector velocities in the euclidean task space. The policy parameter θ_{task_deriv} is used to weight this term. Increasing the task space penalty encourages straight-line motions of the end-effector.

Individual Joint Velocity Penalties It may be reasonable in some robotic applications to favor the use of some degrees of freedom over others. Humans for example tend to use the fingers to play piano or type on a keyboard rather than moving the entire arm simply because this is more convenient in terms of mass distribution, energy consumption and

precision. To express this kind of preference, a third velocity penalty term in joint space is introduced to penalize velocities of every joint individually.

$$c_{indiv\_joint\_deriv}(\dot{\mathbf{q}}) = \frac{1}{2}\sum_{i=1}^{n} \theta_{indiv\_deriv\_i} \cdot \|\dot{q}_i\|^2 \tag{3.2}$$

In case of Apollo's robot arm used in the experimental section, a total of 8 additional parameters is employed to penalize motions of every joint individually.

Default Configuration Potential

The default configuration potential introduces a heuristic bias which tends to move the robot arm to a predefined default configuration $\mathbf{q}_{default}$. Due to the high degree of freedom, the robot may move into twisted and unfavorable positions while navigating the end-effector to a specific target position. Certain arm configurations may offer a much higher level of manipulability, which means a greater variety of possible configurations for consecutive motions.

$$c_{default\_config}(\mathbf{q}) = \theta_{default\_q} \cdot \frac{1}{2}\|\mathbf{q} - \mathbf{q}_{default}\|^2 \tag{3.3}$$

One or more quadratic potentials can be added to emphasis some 'natural' positions that offer a high level of manipulability. In order to not restrict the overall goal of the motion, the influence of this potential can be decreased over time by linearly decreasing its weight parameter $\theta_{default\_q}$.

Joint Limit Penalty

Due to the structure of the motion policy, the optimizer is constrained to respect the robot's joint limits. Still, the behavior when approaching the joint limits may be different between the joints and between different types of motions. The joint limit penalty is used to model this behavior. For joint positions above and below the upper and lower thresholds u_i and l_i a quadratic penalty term is added to the cost function. The penalty term for each joint i may be weighted individually by α_i and the thresholds may be set individually within $q_i^{min} \leq l_i \leq u_i \leq q_i^{max}$ for each joint $q_i \in [q_i^{min}, q_i^{max}]$ and its physical joint limits.

$$c_{joint\_limit}(\mathbf{q}) = \theta_{joint\_limit} \cdot \sum_{i=1}^{n} f_i(q_i) \tag{3.4}$$

$$f_i(q_i) = \begin{cases} (q_i - l_i)^2 & \text{if } q_i < l_i \\ (u_i - q_i)^2 & \text{if } q_i > u_i \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

In this work, the upper and lower thresholds have been set to the upper and lower third of the entire joint range. All joints have been equally weighted ($\alpha_i = 1$) and only the overall weighting factor θ_{joint_limit} is part of the motion policy parametrization. As shown at the example of individually weighted joint velocities, it should also be possible to learn those joint limit specific parameters if required by some special type of motion.

Terminal Velocity Penalty

As it has been described previously, the initial velocity of the system is forced to be closed to zero by the choice of the joint space derivative penalty at time step zero. Similarly, the velocity should decelerate smoothly at the end of the motion. In order to achieve that, a quadratic terminal cost feature is added to penalize high velocities. The smoothness property of the entire motion consequently ramps up and down the velocity.

$$c_T(\dot{\mathbf{q}}) = \theta_{terminal\_velocity} \cdot \frac{1}{2}\|\dot{\mathbf{q}}\|^2 \tag{3.6}$$

Postural Potentials

For certain manipulations and grasping movements, a distinct posture of the manipulator is necessary. For example, a hand should be held horizontally to carry a full glass of water. A quadratic potential is used to pull the axis of joint i which is given by the forward kinematic map $\boldsymbol{\phi}_i : \mathbb{R}^n \to \mathbb{R}^3$ into the direction of a normal vector $\mathbf{n}$.

$$c_{orientation}(\mathbf{q}) = \theta_{orientation} \cdot \frac{1}{2}\|\mathbf{n} - \boldsymbol{\phi}_i(\mathbf{q})\|^2 \tag{3.7}$$

The normal vector $\mathbf{n}$ is currently set to a fixed value, e.g. to the upward z-axis in order to maintain a horizontal hand posture by aligning the finger knuckle's joint axis.

Surface Potential

The surface potential is a linear term which is proportional to the distance from a given surface. In its simplest form, the normal vector $\mathbf{n}$ and a point on a surface $\mathbf{p}$ are given to describe a flat surface (e.g. a tabletop). Given a kinematic map $\phi(\mathbf{q})$, this potential can be applied to any point on the robot. In this work, a potential is utilized to force the end-effector onto the table surface or to push it away from the table. The normal and position vector are not learned and set to a fixed value. Only the weight of this potential is exposed to the optimization.

A similar quadratic term has been introduced for cylindrical obstacles. This term forms a barrier function when approaching the obstacle in radial direction and thus pushes the end-effector away from the obstacle. Both terms became obsolete when switching to the Riemannian workspace representation as described in 3.1.2 and could be replaced by independently-weighted velocity penalties within the influence spheres of each obstacle.

3.1.2 Workspace Representation

Previous cost features centered around the configuration of the robotic system itself and around its derivatives as well as around attractors and potentials either in the configuration/joint space $\mathbb{R}^n$ or in the ambient euclidean workspace $\mathbb{R}^3$. Not only these robot specific features are important to express motion policies but also the relation to the surrounding environment and the behavior with respect to obstacles must be expressed.

So far, no potential has been introduced which guides the motion towards the goal position. The simplest possibility is to use a euclidean metric as a goal attractor in joint space or in task space. In an empty workspace, the straight-line motion resulting from the euclidean distance metric could be the right choice. But moving from one side of a cylindrical obstacle to the other, this metric will pull the end-effector straight into the obstacle. Additional potential fields can be utilized to push the trajectory away from obstacles. This approach has been tested for the 2D experiments, where a linear potential is applied for a flat surface and a radial potential is used to push the end-effector away from a point obstacle.

Ideally, the goal attractor itself would guide the motion into feasible trajectories. Therefore, a new workspace representation based on differential geometry is used to formulate the goal attractor potential. When minimizing the difference between current and goal

position in this new space, the motion translated to the actual 3D workspace automatically avoids obstacles. Details about the workspace representation using Riemannian geometry can be found in [**?**].

The higher dimensional space representing the surrounding workspace can be thought of as a combination of multiple coordinate systems each of which represents one basic geometric object within the workspace e.g. a cylindrical obstacle. These coordinate systems are blended together for each position of the three-dimensional workspace and weighted according to the distance to the individual object. In this work, an ambient euclidean system is composed with cylindrical coordinate systems for each cylindrical obstacle. The influence of the cylindrical system decreases exponentially with the distance from the obstacle and vanishes at a certain radius around the obstacle. This workspace representation has been used in all 3D experiments involving obstacles.

Additionally to the goal attractor defined in the higher dimensional space of the Riemannian geometry, velocity penalties can be added to every dimension. This way, different motion types around the obstacles can be realized. Four additional velocity penalties have been introduced. Three of them penalize the velocities within the cylindrical system and one of them is used to penalize motion in any direction within the ambient euclidean system.

3.1.3 Motion Objective Constraints

In the general formulation of a constrained optimization program, equality and inequality constraints can be given:

$$\begin{array}{lllll} \min & c(\mathbf{x}) & & & \\ \text{s.t.} & g_i(\mathbf{x}) = 0 & \text{for } i = 1, \ldots, k & \text{Equality constraints} & \\ & h_j(\mathbf{x}) \geq 0 & \text{for } j = 1, \ldots, l & \text{Inequality constraints} & \end{array} \tag{3.8}$$

Those hard constraints are used to express requirements which must be fulfilled in order to operate the robotic system safely. The two hard constraints employed in this work are surface and joint limit constraints.

The distance between any point within the workspace and the obstacle is given as an analytic function since every object is represented as a geometric primitive. In this work, constant offset thresholds to the obstacle's surface are given for the finger and wrist joints. Those offsets are part of the policy parametrization.

The joint limit constraints are defined by the physical joint limits and are not part of the policy parametrization.

3.2 Policy Search Reinforcement Learning

Following the discussions in previous sections, a parametrized policy is now available which outputs a solution trajectory $\xi(\boldsymbol{\theta}, \boldsymbol{\gamma})$ depending on the problem's context $\boldsymbol{\gamma}$ and the parameter vector $\boldsymbol{\theta}$. Direct policy search is an iterative method to find the policy that maximizes the expected reward or minimizes the expected loss in contrast to indirect policy search methods which compute the policy by estimating the value function.

In this work, a loss function $\mathcal{L}(\xi(\boldsymbol{\theta}, \boldsymbol{\gamma}))$ is formulated (also called a fitness function in the following). The optimal policy, represented by the optimal parameter vector $\boldsymbol{\theta}^*$, is obtained by solving the following optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}[\mathcal{L}(\xi(\boldsymbol{\theta}, \boldsymbol{\gamma}))] \tag{3.9}$$

In general, the expected loss is minimized, since both the system's state/action transitions and the policy itself are probabilistic. For the specific problems presented in this work, both components are assumed to be deterministic. The policy is fully defined by the parameter vector $\boldsymbol{\theta}$ and the problem context $\boldsymbol{\gamma}$. The computed trajectory $\xi(\boldsymbol{\theta}, \boldsymbol{\gamma})$ deterministically defines the actions to be taken for every state. Furthermore, the system is assumed to precisely follow the given trajectory such that all state/action transitions are deterministically defined as well. Generally, it is not valid to assume a perfect response of the system, but because of Apollo's fast low-level control, we can suppose that it is the case for our first experiments (cf. section 6.1.1).

This section describes the two main components of the optimization program that is used in this work. First, different ways to formulate the loss function are presented 3.2.1. They aim at either mimicking demonstrated behavior or complying with additional requirements. The optimizer suitable for this problem will be depicted in section 3.2.2.

A schematic view of the reinforcement policy search framework is shown in figure 3.2. The underlying optimization-based policy representation has been designed according to 3.1.

3.2.1 Formulation of the Loss Function

The loss function $\mathcal{L}$ must be selected carefully in order to properly represent the reinforcement learning goals. The subsequent sections present formulations suitable to imitate behavior respectively to improve a given policy.

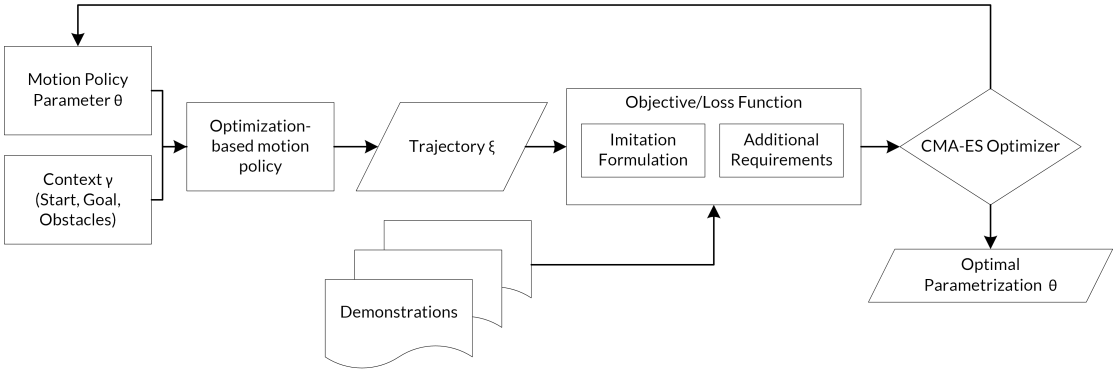


Figure 3.2: Components of the high-level optimization setup for reinforcement policy search.

Imitation Learning

In a first step, the reinforcement framework should be leveraged to imitate demonstrated behavior. This behavior is described by a set $\mathcal{D} = \{(\boldsymbol{\gamma}_i, \xi_i)\}_{i=1}^{k}$ containing k pairs where the first component is the problem context γ_i and the second component is the corresponding discrete demonstration trajectory ξ_i. Each demonstration may be composed of arbitrarily many discrete configurations: $\xi_i = (\mathbf{q}_0, \ldots, \mathbf{q}_{T_{demo}^{(i)}})$.

For each problem context, the solution trajectory ξ_{gen} can be generated according to the current set of policy parameters. The number of configurations of the solution can be chosen freely. In this work, $T = 10$ is usually chosen for the 2D experiments and $T = 30$ for 3D experiments.

To optimize the similarity between the observed behavior $\mathcal{D}$ and the motions generated by the current policy, different loss functions are possible. We assume that the data available for each demonstration describes the full state of the robotic system. Additionally, we assume that demonstration and solution trajectory have the same length such that for every time step, the solution configuration can be directly compared to the according demonstration configuration. For this scenario, three distance metrics have been evaluated in the 2D experiments.

Average euclidean distance in joint space

$$d_{AvgEucl}(\xi_{gen}, \xi_{demo}) = \frac{1}{T+1} \sum_{t=0}^{T} \|\mathbf{q}_{demo}^{(t)} - \mathbf{q}_{gen}^{(t)}\| \tag{3.10}$$

Maximal euclidean distance in joint space

$$d_{MaxEucl}(\xi_{gen}, \xi_{demo}) = \max \left(\|\mathbf{q}_{demo}^{(t)} - \mathbf{q}_{gen}^{(t)}\| \right)_{t=0}^{T} \tag{3.11}$$

Combined objective to reach the motion's goal point and imitate the demonstration

$$\begin{aligned} d_{Combined}(\xi_{gen}, \xi_{demo}) = \frac{1}{T_{gen}+1} \sum_{t_{gen}=0}^{T_{gen}} \min \left(\|\mathbf{q}_{demo}^{(t_{demo})} - \mathbf{q}_{gen}^{(t_{gen})}\| \right)_{t_{demo}=0}^{T_{demo}} \\ + w_{goal} \cdot \|\mathbf{q}_{demo}^{(T)} - \mathbf{q}_{gen}^{(T)}\| \end{aligned} \tag{3.12}$$

In most cases, the generated trajectories are restricted to a small number of steps ($T \ll 100$), to allow for efficient and fast computation. The demonstration in contrast may encompass many more discrete configurations e.g. arising from sampling the state of a manually moved robotic system at a high rate. A demonstration that is given by a human teacher usually incorporates jerky behavior which is not characteristic for the desired movement but originates from the hurdles of manually moving the robot arm. The same applies for sketched end-effector paths or other types of manually taught motions. The velocity and acceleration profiles of such demonstrations must not be taken as a reference since intermediate stops or slow-downs may be necessary to reposture the arm but may not be part of the desired behavior. To get rid of these artifacts, only the path in joint or task space is taken into account. We abstract away from the actual velocity and acceleration profile by subsampling a trajectory of configurations having constant euclidean distance (thus representing a motion at constant speed) from the observed trajectory. The resulting trajectory is further down-sampled to the size of the generated trajectory by selecting T_{gen} configurations from the demonstration in an equidistant fashion.

$$\xi_{demo} = (\mathbf{q}_{demo}^{(t)})_{t=0}^{T_{gen}} \quad \text{s.t. } \|q_t - q_{t+1}\| = \text{const. } \forall t \in [0, \ldots, T_{gen} - 1] \tag{3.13}$$

As both trajectories now have an equal number of steps, they can be compared using the distance metrics explained above. Alternatively, these metrics can be applied to the euclidean distance of given key-points on the robotic system in task space. In this case, the forward kinematic map $\phi(\mathbf{q})_{keypoint}$ is employed to map a configuration $\mathbf{q}$ at time step t to the vector $\mathbf{x}$ containing the task space coordinates of the specified key-point. In case of Apollo's arm, a total of 10 key-points at the center of each revolute joint is taken into account leading to a $3 \cdot 10 = 30$ dimensional output vector for the entire kinematic map.

Whereas the first two distance metrics (cf. equations 3.10 and 3.11) are only applicable to demonstrations and generated trajectories of equal length ($T_{demo} = T_{gen}$), the third metric can also be applied in case $T_{demo} \neq T_{gen}$. This last objective expresses two requirements. First of all, the generated motion has to reach the goal point (i.e. the distance between $\mathbf{q}_{demo}^{(T)}$ and $\mathbf{q}_{gen}^{(T)}$ needs to be minimized). Secondly, the distance between each configuration of the generated solution and *any* demonstrated configuration must be minimized. This formulation might help to detect the compliance of a generated solution with jerky demonstration behavior because it considers compliant behavior outside the jerky regions. All three objectives have been evaluated on different problems to assess their applicability to real-world problems (cf. chapter 5).

In general, the policy should be learned from multiple demonstrations $\xi_{i=0}^{k}$ in order to avoid adaptation to specific characteristics of a single, maybe deficient, demonstration. The loss function is therefore given by the average distance between the generated trajectory and all demonstrations. This way, the influence of dominant characteristics persistent within the given set of demonstrations is further increased.

$$\mathcal{L}_{imitation}(\xi_{gen}, \xi_{i=0}^{k}) = \frac{1}{k} \sum_{i=0}^{k} (d(\xi_{gen}, \xi_i)) \tag{3.14}$$

Imposing Additional Requirements

Especially in scenarios where only a part of the system is constrained by the demonstrated behavior, additional requirements can be formulated to resolve the redundancy arising for the rest of the system and to support a specific behavior. Two examples are given below and evaluated in the experimental section.

Elbow Criteria When the demonstration only incorporates the end-effector's position (which is the trajectory of the left fingertip in the given examples), the complex kinematic of the 8 DoF system offers many redundant ways to fit this demonstration. For example, the position of the elbow can be changed arbitrarily without affecting the end-effector's position at all. As it was observed that a low positioning of the elbow leads to arm poses that offer greater flexibility, this redundancy is resolved by penalizing movements of the elbow to an upper position. In order to achieve this, a new term $\mathcal{L}_{elbow}$ is added to the loss function which consists of a downward potential that favors low elbow positions.

The elbow height with respect to a horizontal plane is summed over all time steps. The plane is specified by a point on its surface $\mathbf{p}$ and the normal vector $\mathbf{n}$.

$$\mathcal{L}_{elbow}(\xi_{gen}) = \sum_{t=0}^{T_{gen}} (\phi_{elbow}(\mathbf{q}_{gen}^{(t)}) - \mathbf{p}) \cdot \mathbf{n} \tag{3.15}$$

Smoothness Criteria By exposing all parameters of the underlying objective function to the policy search step, the optimizer can run into policies that generate jerky and non-smooth trajectories. This can be the case for jerky or complex demonstrations as well as for scenarios that are not easily representable by the specific formulation of the optimal control problem. But the smoothness of the trajectory is crucial for its successful execution on real hardware. To achieve smooth trajectories, an additional term is incorporated in the loss function

The non-smoothness is measured by the average acceleration that arises when executing the trajectory. Central differencing is used to compute the acceleration for each intermediate step of the trajectory. The penalty term is the average over those single accelerations (cf. equation 3.17).

$$a_t = \|\mathbf{q}_{t-1} - 2\mathbf{q}_t + \mathbf{q}_{t+1}\| \tag{3.16}$$

$$\mathcal{L}_{smoothness}(\xi_{gen}) = \frac{1}{T_{gen}+1} \sum_{t=0}^{T_{gen}} a_t^2 \tag{3.17}$$

3.2.2 Covariance Matrix Adaptation Optimization

To express similarity to a set of demonstrations and to measure the fulfillment of additional requirements, arbitrary formulations for loss functions are possible. Usually the analytic formulation of the gradient or second derivative information is not available for all of these formulations. Therefore a black box optimizer should be employed for this type of problem in order to optimize with respect to the policy parametrization. Black box optimization refers to the scenario where only function evaluations are possible and no gradient information is available. The performance or cost of such an optimizer is usually measured in the number of function evaluations that are necessary to to meet a certain termination criterion.

Since the underlying policy is fully characterized by its parametrization $\boldsymbol{\theta} \in \mathbb{R}^n$ and the representation of the environment $\boldsymbol{\gamma}$, the policy search reinforcement learning can be easily formulated as a standard black box optimization problem $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\theta})$

The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) is a stochastic and derivative-free optimization method proposed by [**?**]. Their publication also contains an instructional tutorial for the algorithm. The method is applicable for non-linear, non-convex problems on a continuous real-valued domain. Whereas the CMA-ES algorithm is a local optimizer in its original formulation, special adaptations can be made to turn it into a global optimizer (cf. section 3.2.3).

Inspired by evolution in nature, evolutionary algorithms strive to optimize certain characteristics of a population by evolving it over multiple generations. Each generation arises from the fittest members of the preceding generation; this way, the fitness criterion improves over time. In the following, the main steps of one iteration of the CMA-ES algorithm are listed:

Sampling:

$$\boldsymbol{\theta}_{i=1,\dots,\lambda} \sim \mathcal{N}(\boldsymbol{\theta}, \sigma^2\Sigma) \tag{3.18}$$

Evaluation:

$$L_i = \mathcal{L}(\theta_i) \tag{3.19}$$

Sorting:

$$\boldsymbol{\theta}_{i=1,\dots,\lambda} \leftarrow \text{sort}\boldsymbol{\theta}_{i=1,\dots,\lambda}\text{w.r.t}L_{i=1,\dots,\lambda} \tag{3.20}$$

Update:

$$\boldsymbol{\theta}^{new} = \text{update}_{\boldsymbol{\theta}} \tag{3.21}$$

$$\boldsymbol{\Sigma}^{new} = \text{update}_{\boldsymbol{\Sigma}} \tag{3.22}$$

In the sample step, λ parent points are drawn from a multivariate Gaussian distribution. The distribution is given by a mean $\boldsymbol{\theta}$ and a covariance matrix which is split into a scalar step size σ^2 and a matrix $\boldsymbol{\Sigma}$ defining the distribution's shape. This way, shape and magnitude of the Gaussian can be modified independently. The automatic adaption of the exploration noise (which is defined by the step size σ) is one big advantage of CMA-ES over other approaches (e.g. CEM, PI^2). The loss function is evaluated at every parent point and the μ best offsprings (called elite samples) from each generation are used to update the distribution parameters.

As pointed out in [**?**], the update steps are the crucial component of the CMA-ES method which sets it apart from CEM and PI2. A probability-weighted averaging of all offspring

sample points is used to determine the new distribution mean. Any arbitrary choice of probabilities $P_{i=1,\dots,\mu}$ for the elite samples is possible as long as $\sum_{i=1}^{\mu} P_i = 1$ and $P_1 \geq \cdots \geq P_\mu$. The default choice given by equation 3.23 as suggested by [**?**] is used in this work. The update of $\boldsymbol{\theta}$ is therefore given by:

$$P_i = \ln(0.5(\lambda + 1)) - \ln(i) \quad \forall i \in \{1, \dots \mu\} \tag{3.23}$$

$$\boldsymbol{\theta}^{new} = \sum_{i=1}^{\mu} P_i \boldsymbol{\theta}_i \tag{3.24}$$

To compute the new covariance matrix of the underlying distribution, both step-size (σ) and shape (Σ) are updated separately. The 'evolutionary path' for both parameters ($\boldsymbol{p}_\sigma$ and $\boldsymbol{p}_\Sigma$) maintains information about the history of $\boldsymbol{\theta}$ from previous generations. This way, CMA-ES can exploit trends like several consecutive changes in the same direction by increasing the step size.

Step Size Update:

$$\boldsymbol{p}_\sigma \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}\boldsymbol{\Sigma}^{-1}}\frac{\boldsymbol{\theta}^{new} - \boldsymbol{\theta}}{\sigma} \tag{3.25}$$

$$\sigma^{new} = \sigma \cdot \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0}, \boldsymbol{I})\|} - 1\right)\right) \tag{3.26}$$

Distribution Shape Update:

$$\boldsymbol{p}_\Sigma \leftarrow (1 - c_\Sigma)\boldsymbol{p}_\Sigma + h_\sigma\sqrt{c_\Sigma(2 - c_\Sigma)\mu_p}\frac{\boldsymbol{\theta}^{new} - \boldsymbol{\theta}}{\sigma} \tag{3.27}$$

$$\begin{aligned}\boldsymbol{\Sigma}^{new} = (1 - c_1 - c_\mu)\boldsymbol{\Sigma} + c_1(\boldsymbol{p}_\Sigma\boldsymbol{p}_\Sigma^T + \delta(h_\sigma)\boldsymbol{\Sigma}) \\ + c_\mu\sum_{k=1}^{K_e} P_k(\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^T\end{aligned} \tag{3.28}$$

There are some open learning parameters in CMA-ES which are chosen according to the default values as suggested by Hansen. The algorithm was designed to work robustly on a large set of problems given these default parameters. It has been extensively tested amongst others in the course of the Black Box Optimization Benchmarking workshop which revealed the algorithm's applicability to a large variety of both noisefree and noisy test problems [**?**]. The learning parameters and general settings as used in this work are given below in accordance to [**?**]:

Selection and Recombination:

$$\lambda = 4 + \lfloor 3 \ln n \rfloor, \quad \mu = \lfloor \frac{\lambda}{2} \rfloor \tag{3.29}$$

Step-size control:

$$c_\sigma = \frac{\mu_{eff} + 2}{n + 4 + 2\mu_{eff}/n}, \quad d_\sigma = 1 + 2 \max \left(0, \sqrt{\frac{\mu_{eff} - 1}{n+1}} - 1 \right) + c_\sigma \tag{3.30}$$

Covariance matrix adaptation:

$$c_c = \frac{4 + \mu_{eff}/n}{n + 4 + 2\mu_{eff}/n} \tag{3.31}$$

$$c_1 = \frac{2}{(n+1.3)^2 + \mu_{eff}} \tag{3.32}$$

$$c_\mu = min \left(1 - c_1, \alpha_\mu \frac{\mu_{eff} - 2 + 1/\mu_{eff}}{(n+2)^2 + \alpha_\mu \mu_{eff}/2} \right) \quad \text{with } \alpha_\mu = 2 \tag{3.33}$$

A sample run of CMA-ES on a simple two-dimensional quadratic problem is shown in figure 3.3 to illustrate the distribution's development over several iterations of the algorithm. A larger population size is chosen to clearly visualize the sample points (black dots) drawn in each generation. The shape and size of the current distribution is visualized by the 2σ border (given as orange dotted line). Lines of constant loss function value are used together with a color gradient to visualize the spherical optimization problem. The evolution of the population over the course of six generations clearly shows the fast convergence of the distribution towards the problem's optimum.

3.2.3 BiPop CMA-ES

The original CMA-ES implementation can be seen as a local optimizer refining the initial guess. CMA-ES does not depend overly strongly on the initial step size. This has been proven in [**?**] and it is also visible in figure 3.3 where the inappropriate initial step size is modified considerably between generation one and three. The algorithm automatically adapts to sensitivity differences between the search space dimensions and also to errors in the initial guess as long as they do not exceed one to two magnitudes. To solve even more global optimization problems, different concepts can be applied. Global search can be achieved by increasing the population size as shown in [**?**]. A modified version of

[1] `http://upload.wikimedia.org/wikipedia/en/d/d8/Concept_of_directional_optimization_in_CMA-ES_algorithm.png`

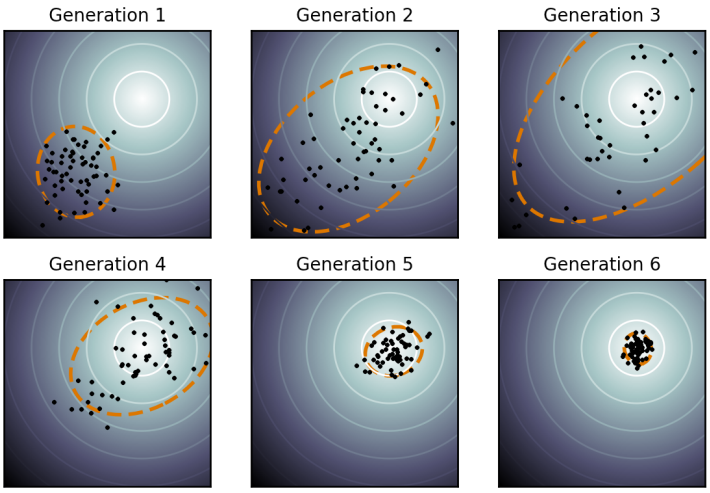


Figure 3.3: Visualization of multiple CMA-ES generations (black dots) on a two-dimensional spherical problem [1].

CMA-ES was proposed to further increase applicability to global optimization problems. BiPop CMA-ES as proposed by [**?**], implements a two population restart scheme based on independent CMA-ES runs. The first restart regime utilizes a growing population size to achieve greater coverage of the parameter space while the other regime is limited to a small number of samples for faster convergence.

The BiPop CMA-ES has been implemented in this work to evaluate whether global optimization approaches are necessary for the given problem setup. Details about the practical application of both optimizers, especially parameter encoding and normalization as well as boundary and domain space handling, are given in chapter 4.3.

4 Experimental Setup

Even for an autonomous policy search algorithm as presented in the last chapter, certain arrangements are necessary to learn and operate successfully on a real robotic system. In this chapter, Apollo, the robotic platform used in this work for more complex real-world test scenarios, is introduced. Thereafter, some practical hints are given for the use of the motion objective formulation and finally the configuration of the CMA-ES optimizer is explained.

The software framework used for the experimental evaluation is written in C++ based on the underlying optimization framework written by Nathan Ratliff. The biggest part of the newly developed code is formed by the representation of the reinforcement objective function, the high-level optimizers (CMA-ES and BiPop CMA-ES) and helper classes to store demonstration and context data. The Robot Operating System (ROS) is mainly used for visualization (RViz) and as an interface to the real-time operating system running the Simulation Laboratory (SL) software to control the actual robot. The loss function evaluation for each sample point of the current optimization generation has been parallelized using OpenMP to speed up the optimization on multi-core systems.

4.1 Apollo Humanoid Robot Platform

This work focuses on reaching and pointing motions executed by a humanoid robot arm. The actual experimental platform is a humanoid upper body, called Apollo. The robot is equipped with two KUKA lightweight robot arms and attached Barret manipulator hands. Each KUKA arm offers 7 DoF which allows manipulability similar to a human arm but its joint positioning is slightly different (cf. figure [4.1]). The joint centers are referred to as key-points within this thesis.

The Barrett hand is equipped with three fingers each one having two joints. Additionally, the spread between the thumb and the other fingers can be varied. For the experiments presented in this thesis, both finger joints have been coupled to be controllable as one joint and the finger spread has been set to a fixed value. Only the left finger is used as shown in image 4.1 to allow the robot to point to goal positions.

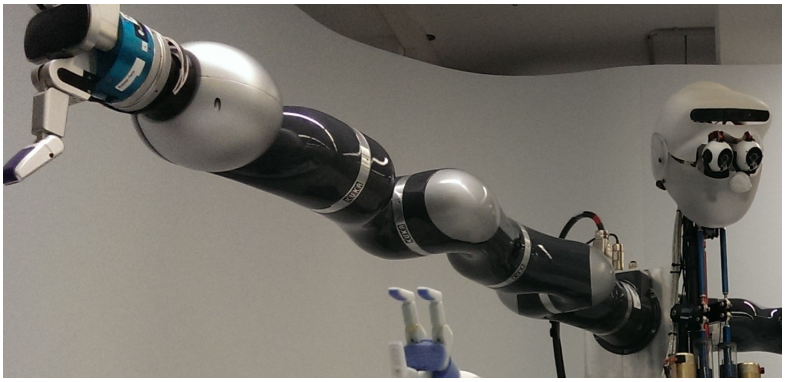


Figure 4.1: Close up of Apollo's right arm and hand as used for the experiments.

4.2 Motion Objective Optimization

In table 4.1, a summary of all currently implemented cost features is given. A total of 25 parameters can be used to parametrize motion policies. For each motion experiment, the actually employed features are marked. Some features may be unnecessary to express a specific motion and can be excluded or set to a fixed value.

Apparently, the set of features exhibits some redundancy: there are different features which can be used interchangeably to express a certain characteristic of a motion. Therefore, two distinct policies may lead to similar resulting motions. The following pairs describe some redundancies that have not been resolved in this work:

- Derivative penalty (joint space) – Individual joint velocity penalty
- Upper wrist (finger) posture – Lower wrist (finger) posture
- Derivative penalty (task space) – Euclidean velocity (euclidean system)

From the experimental results as presented in chapter 5, no faulty behavior was observed due to those redundancies. The policy optimizer turns out to be robust against redundant parametrization and is capable to automatically spread out contributions to a specific motion characteristic over multiple redundant terms. For example in the case of upper and lower wrist posture, this may lead to the use of only one of the terms to position the hand. Since both wrist positions are close to each other, this still results in the desired motion. Nevertheless, certain scenarios and motion problems could require the usage of both terms. Learning from additional demonstrations for these situations might therefore prioritize the other wrist term and automatically resolve this redundancy.

#	Cost feature	Normalization	Default	1	2	3	4	5	6
1	Derivative penalty (joint space)	0.01	1	x	x	x	x	x	x
2	Derivative penalty (task space)	100	2		x	x	x	x	x
3	Joint potential	0.1	1	x	x	x	x	x	x
4	Default configuration	10	3	x	x	x	x	x	x
5	Table potential	1	0		x	x			
6	Cylinder potential	1	0				(x)	(x)	
7	Terminal position	10,000	10	x	x	x	x	x	x
8	Terminal velocity	1,000	3		x	x	x	x	x
9	Upper wrist posture	0.1	1.7		x	x	x	x	
10	Lower wrist posture	0.1	1.4		x	x	x	x	
11	Upper finger posture	0.1	0.6				x	x	
12	Lower finger posture	0.1	0.1				x	x	
13	Hand orientation	1	0						x
14	Radial velocity (cylindrical system)	1	10				x	x	
15	Circumferential velocity (cylindrical system)	1	0				x	x	
16	z-velocity (cylindrical system)	1	0				x	x	
17	Euclidean velocity (euclidean system)	1	1				x	x	
18	Joint velocity (Joint 1)	1	0			x		x	x
19	Joint velocity (Joint 2)	1	0			x		x	x
20	Joint velocity (Joint 3)	1	0			x		x	x
21	Joint velocity (Joint 4)	1	0			x		x	x
22	Joint velocity (Joint 5)	1	0			x		x	x
23	Joint velocity (Joint 6)	1	0			x		x	x
24	Joint velocity (Joint 7)	1	0			x		x	x
25	Joint velocity (Joint 8)	1	0			x		x	x
26	Point obstacle potential	1	5	x					

Table 4.1: Summary of all cost feature terms used in this work. A normalization factor and the default value are given. For each presented experiment, the composition of the objective function is indicated. 1=2D Experiments, 2=Task/Joint space, 3=Predominant joints, 4=Obstacle avoidance, 5=Obstacle avoidance + Elbow, 6=Hand orientation

4.3 CMA Optimizer

All parameters of the CMA-ES algorithm have been set according to Hansen's default implementation (cf. section 3.2.2). The only parameters of the optimizer that need to be chosen in a problem-specific way are the initial solution point $\boldsymbol{\theta}_0 \in \mathbb{R}^d$ and a scalar initial step size σ_0. Moreover, some assumptions about the underlying search space and the sensitivity of individual dimensions as well as specific termination criteria are necessary to adapt the algorithm to our problem. The choices made for the experimental evaluations presented in chapter 5 are described in the following.

4.3.1 Termination Criteria

Several termination criteria as proposed by [**?**] have been implemented in the policy search optimization. With a combination of two among those (fitness tolerance termination and fitness history termination), reasonable termination can be guaranteed for all problem formulations used in this work. The first criterion terminates optimization as soon as the fitness level reaches a certain threshold. But only in case of the noisefree 2D experiments, it has been possible to precisely reproduce the policy of the demonstration and thus to reduce the loss to zero. In all other cases, a reasonable threshold for terminating the optimization can not be known a priori. The fitness history termination criteria is therefore employed in those scenarios: In case the fitness level stagnates for at least 30 generations, the optimization will terminate.

The thresholds for both termination criteria have been set to 0.01 for all experimental evaluations. No significant improvement of the policy's fitness was observable when using stricter termination criteria, especially not in the experiments on the real system.

4.3.2 Boundaries and Constraints

To achieve the optimal search performance, all parameters (i.e. all search dimensions) should have similar sensitivity. The raw parameter values as defined in section 3.1.1 cover more than six magnitudes. To achieve a reasonable parameter encoding, the dimensions have been rescaled to the range $[0, 10]$. The normalization factors as used in this work are listed together with all cost features and some default values[1] in table 4.1. The normalization factors have been determined experimentally by manually weighting the cost features in order to achieve a smooth motion in unobstructed space.

[1] Default values are given after normalization.

The motion policy as given in section 3.1 is only defined on positive parameters but the CMA-ES default implementation is originally meant to operate on $\mathbb{R}^d$. To get around the need to actively constrain the search space to the non-negative hemisphere, the objective function is evaluated on the absolute parameter, i.e. $c(\boldsymbol{\theta}) = c(-\boldsymbol{\theta})$. Other methods to constrain the search space by a lower and/or upper bound (e.g. quadratic or exponential replacement) have been proposed in [**?**] but performed poorly for the problems investigated in this work.

The initial solution point is chosen randomly from the assumed parameter range: $\boldsymbol{\theta}_0 \sim \mathcal{U}(0, 10)^d$. In [**?**], CMA-ES has been shown to successfully adapt the individual exploration noise of each dimension to explore up to several magnitudes from the initial starting point. In accordance to [**?**], the initial step size is set to a fifth of the domain size. Tests with different domain sizes, population sizes and initial step sizes σ have been conducted in this work to verify the adequateness of the normalization and the selected search space.

Active box constraints have been tested to limit the search space. Outside of the search domain, a quadratic penalty proportional to the distance to the legal search space was added to focus search on the chosen domain. The experiments revealed no significant improvement as a result of applying active box constraints, neither in convergence speed nor in the fitness of the objective function. The chosen step size already suffices to limit the search space to an area of the size of two magnitudes around the initial guess.

5 Experimental Results

The proposed reinforcement policy search algorithm has been experimentally evaluated on multiple problems. An excerpt of the conducted experiments will be presented in this chapter to flesh out the algorithm's suitability for direct policy search, imitation learning and subsequent policy improvement. In detail, several capabilities that are unique to this approach, especially in contrast to the recently developed DMP-based policy search approaches, are of interest:

1. Feasibility of direct policy search on *optimization-based policy representations.*
2. Imitating demonstrated behavior, *independent of the concrete problem's geometry.*
   a) Learning *characteristic concepts* from demonstrated motion primitives.
   b) *Generalizing* to untrained scenarios (including new environments).
   c) Learning from *incomplete* or noisy demonstrations.
   d) Learning from demonstrations that only focus on *some aspects* of the desired motion.
3. Imposing *additional requirements* to the learned behavior.

5.1 2D Point Experiment

This experiment has been conducted to gain first insights into the applicability and performance of the new method. Therefore, the problem has been reduced to a two-dimensional state space which can be easily visualized. A test context is given by start and goal position and one of the obstacle positions as summarized in table 5.1.

Start position	$(0, 0)$
Goal position	$(2, 0)$
Default position	$(0, 1)$
Obstacle positions	$\{(0.5, 0.5), (1, 0.5), (1.5, 0.5), (0.5, 0.1), (1, 0.1), (1.5, 0.1)\}$

Table 5.1: Test scenario for the 2D experiments.

In order to come up with demonstrations for the imitation learning, trajectories have been generated from a manually tuned policy. The objective function is a linear combination of five terms (cf. section 3.1.1), each of them being weighted by its own parameter. The cost features used in the 2D experiments and the manually selected weights are listed in table 5.2. By using the same policy representation to generate demonstrations and to learn the policy, we make sure that the desired policy is within the span of all possible policies. Therefore, the algorithm is expected to learn a parametrization that replicates the demonstrated behavior precisely. In the best case, the algorithm should learn the exact parametrization of the policy used for demonstration. In this case, the approach would not only replicate the precise trajectory for the demonstration scenarios but would have generalized from the shown demonstration to the correct policy underlying those demonstrations.

Cost feature $c_i(\boldsymbol{\xi}, \boldsymbol{\gamma})$	**Form**	**Demonstrated policy**
Terminal configuration potential	quadratic	10
Derivative penalty	quadratic	0.1
Default configuration potential	quadratic	0.1
Upward potential	linear	1
Obstacle potential	exponential	5

Table 5.2: Motion policy's cost features for the 2D experiments.

The parameters used to generate the demonstrations have been selected to achieve a smooth motion from start to goal point and a slight obstacle avoidance. The default configuration is almost neglected. A linear potential is pushing the trajectory upwards (i.e. pushing the end-effector away from a tabletop).

5.1.1 Synthetic Noisefree Demonstrations

The setting and results of the first experiment are shown in figure 5.1. The reinforcement, high-level objective, is formulated to imitate the observed behavior. Since both demonstration and solution trajectory are generated from the same class of policies and both are composed of an identical number of discrete configurations ($T = 10$), the imitation objective can directly compute the distance between demonstration and solution trajectory for each time step.

On the left-hand side of figure 5.1, the resulting average euclidean distance between demonstration and solution trajectory (which is used as loss function in this case) is shown in the course of the CMA-ES-generations. The policy search was terminated

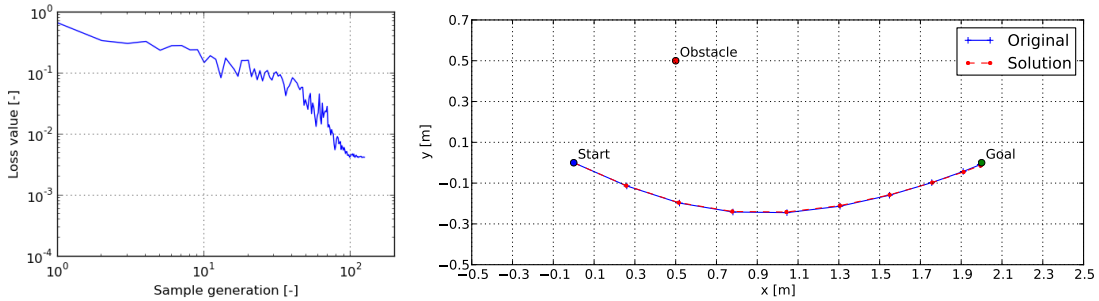


Figure 5.1: Setting and results of the first 2D experiment. The policy is learned from a single noisefree demonstration trajectory.

when a change in fitness ≤ 0.001 was observed. For this simple example, the proposed framework has been able to exactly derive the policy parametrization from one demonstration trajectory up to a constant factor[1].

5.1.2 Synthetic Noisy Demonstrations

The same setting as described for the first experiment has been utilized for the second test. Additionally, Gaussian noise ($\Delta x, \Delta y \sim \mathcal{N}(0, 0.05)$) has been added to all demonstrated configurations except for the start and goal position. The results of this experiment are shown in figure 5.2. The noisy trajectory is shown in green.

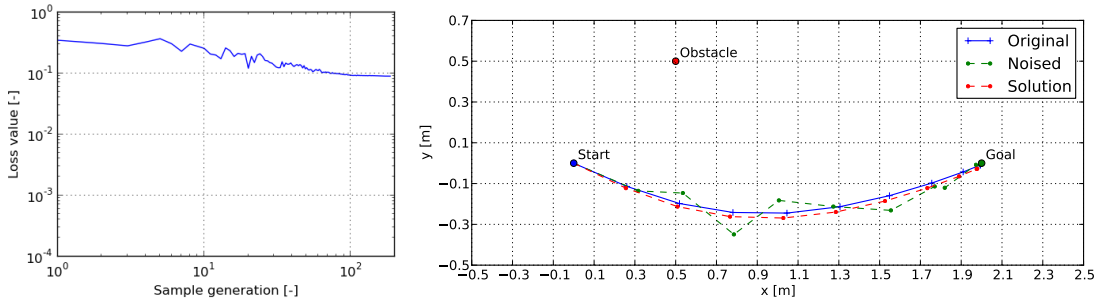


Figure 5.2: 2D policy learned from a single noisy demonstration trajectory. Gaussian noise was added to every intermediate trajectory point.

The solution trajectory (red dashed line) for this problem setting as produced by the learned policy clearly shows the basic characteristics of the demonstrated motion. The slight deviation of the solution trajectory from the noisefree one reflects the divergence

[1] Rescaling of the parameter vector does not change the policy since the objective function is a linear weighted composition of constant (with respect to the parametrization) cost features.

of the noisy demonstration from the noisefree original. The loss function can not be minimized as far as seen for the first example. This is because the noisy demonstration with its jerks and non-smooth behavior can not be exactly represented by this class of policies.

5.1.3 Learning from Multiple Demonstrations

Generally, the deviation of the learned policy from the original demonstration trajectory and the underlying policy due to noisy demonstrations as seen in the last experiment (cf. section 5.1.2) should diminish when taking into account a larger set of demonstrations. The same experiments as outlined before have been repeated for a set of six demonstrations each of them having a different obstacle position and therefore a different shape. Three obstacle positions and the corresponding demonstration trajectories are shown in figure 5.3.

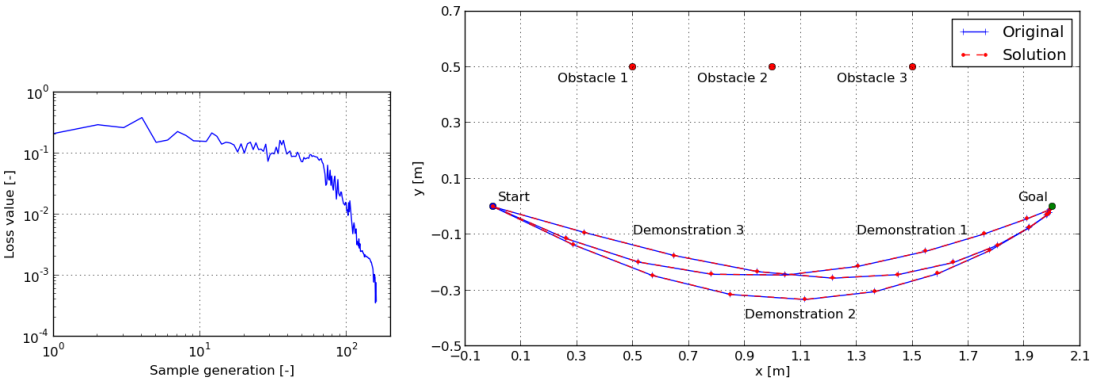


Figure 5.3: 2D policy learned from 6 noisefree demonstration trajectories.

In case of noisefree demonstrations, the results as shown in figure 5.3 are very similar to the ones obtained when learning from a single noisefree demonstration. The difference between solution and demonstration is successfully minimized and the solution policy produces trajectories which align with the given demonstrations. The learned policy parametrization again matches the manually picked values up to a constant factor.

Gaussian noise is now added to every intermediate point of every trajectory to simulate a noisy demonstration. Three out of six noisy demonstrations (green dashed lines) are shown together with their corresponding noisefree original demonstrations (blue lines) and the trajectories produced by the learned optimal policy (red dashed line) in figure 5.4. Comparing the solution trajectories with the ones produced after learning from a single noisy demonstration, it is most obvious that the former ones match the original

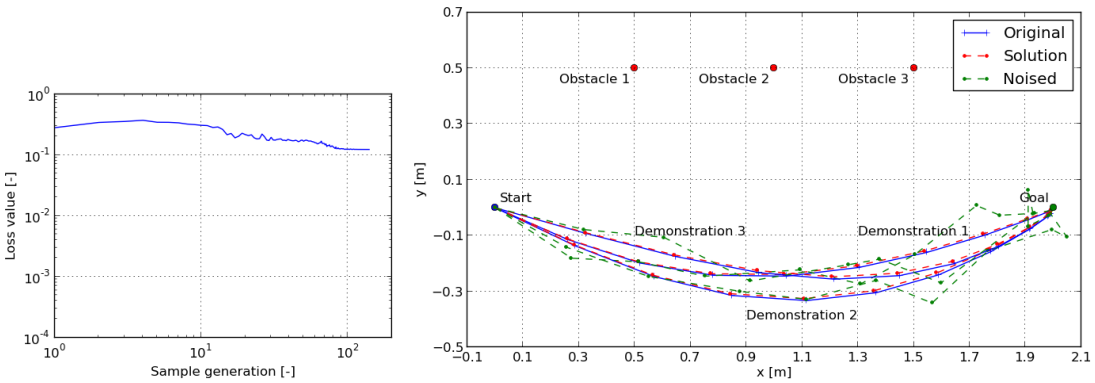


Figure 5.4: 2D policy learned from 6 noisy demonstration trajectories.

(noisefree) trajectories almost perfectly despite the policy being learned from the noisy demonstrations.

The learned policy parametrization matches the manually selected parameters up to a constant factor. As seen in the second experiment presented in section 5.2, the improvement between the loss of the initial policy and the loss of the optimized policy is less pronounced than in case of learning from noisefree demonstrations. Nevertheless, the proposed method is capable to identify the original policy parametrization.

5.1.4 Performance and Test Summary

In table 5.3, the conducted 2D experiments are summarized together with main performance figures to compare the algorithm's performance on those problems. The results and their sample-based standard deviation σ are computed from 5 independent runs. The magnitude of necessary function evaluations ($\sim 10^3$) to achieve termination seems to be constant over all experiments independent of the noise or the number of given demonstrations. A slight tendency to faster convergence for learning from single demonstrations is noticeable. The time consumption in contrast is visibly influenced by the number of given demonstrations. On a single core system without parallelization, the evaluation time is almost proportional to the number of samples. The time required for CMA-ES computations (sampling, sorting and update) can be neglected compared to the time needed for the evaluation of the objective function.

To asses the performance of different distance metrics as presented in section 3.2.1, the number of generations up to termination and the fitness of the resulting solution policy have been statistically analyzed. The 2D experiments, namely learning from noisefree as well as noisy demonstrations and learning from a single demonstration as well as from

Test case	**1**	**2**	**3**	**4**
# of demonstrations	1	1	6	6
Gaussian noise	-	x	-	x
Cost features	5	5	5	5
Population size λ	8	8	8	8
Trajectory size T	10	10	10	10
# of generations [-] (σ)	122 (18)	102 (17)	160 (27)	135 (21)
Best fitness [mm] (σ)	2.4 (1.3)	36.9 (2.3)	0.5 (0.2)	43.2 (6.0)
Time per generation [ms] (σ)	262 (66)	246 (41)	1824 (362)	1762 (283)

Table 5.3: Summary of test setups and optimizer performances for 2D experiments.

multiple demonstrations have been used benchmarks. From five independent runs, the average results and standard deviations σ have been computed as given in table 5.4. The fitness of the resulting policy is measured by the average euclidean distance, independent of the distance metric used to learn the policy.

#	**Noise**		**AvgEucl**	**MaxEucl**	**Combined**
1	-	# of generations [-] (σ)	101 (3)	138 (20)	460 (209)
		Best fitness [mm] (σ)	4.12 (1.27)	3.64 (2.88)	3.32 (4.76)
	x	# of generations [-] (σ)	108 (28)	131 (13)	193 (43)
		Best fitness [mm] (σ)	52.40 (8.23)	58.93 (7.44)	56.29 (18.1)
6	-	# of generations [-] (σ)	135 (19)	154 (17)	149 (12)
		Best fitness [mm] (σ)	2.35 (2.35)	0.35 (0.17)	0.035 (0.014)
	x	# of generations [-] (σ)	128 (17)	148 (32)	196 (30)
		Best fitness [mm] (σ)	50.00 (7.37)	45.05 (5.32)	42.19 (5.01)

Table 5.4: The optimization performance (average number of generations) and quality (average euclidean distance) is compared for three different distance metrics. Policies are learned from all combinations of noisy and noisefree as well as single and multiple 2D demonstrations.

The distance metric employed in the learning process has no significant effect on the fitness of the learned policy. The convergence speed in contrast varies. Consistent throughout all test cases, the average euclidean distance metric leads to the fastest convergence, followed by the maximum distance metric and the combined distance metric. Therefore, the average euclidean distance metric will be the default metric for all upcoming experiments.

In the 2D experiments, based on the motion of a point in 2D space, there was no joint space since the motion is fully described by the point's position. Switching to the real-world robot arm experiments, joint space and task space must be distinguished. A small deviation in one of the robot arm's shoulder joints results in a much stronger deviation of the finger tip position. The average euclidean distance metric will therefore be applied to the task space deviations computed for the 10 key-point positions on the robot arm.

5.2 Full Joint State Demonstrations

In the following, the results from experiments on the actual robot are presented. The demonstrations have been recorded by manually moving the robot arm. In its gravity assist mode, the arm can be moved freely. The joint angles have been recorded as soon as a motion in joint space above a certain threshold distance d_{rec} has been recognized ($\|\Delta\mathbf{q}\|^2 \geq d_{rec}$). The resulting demonstration has been downsampled to match the size of the generated solution trajectory. A trajectory size $T = 30$ has been selected for all examples in this section. The full joint state has been taken into account to measure the distance between demonstration and solution at every time step as discussed in section 3.2.1.

5.2.1 Task Space vs. Joint Space Optimality

Demonstrations are given for pointing motions on a tabletop in front of the robot. In this experiment, a policy should be learned that replicates straight motions in task space and slides the end-effector on the table surface as shown in all demonstrations. The arm has been moved close to its default configuration for all demonstrations to emphasize this 'natural' position.

As it is visualized in figure 5.5 for one demonstrated scenario, the learned policy matches the given demonstrations very well. The characteristic parts of the motion have been successfully generalized and are represented by this policy. Straight end-effector motions are incorporated in the ratio of task and joint space derivative and the linear potential field is pulling the end-effector towards the table.

5.2.2 Predominant Joints

Due to the high degree of freedom of Apollo's robot arm system, a certain motion within an obstacle-free space can be executed in different ways. In this example, demonstrations

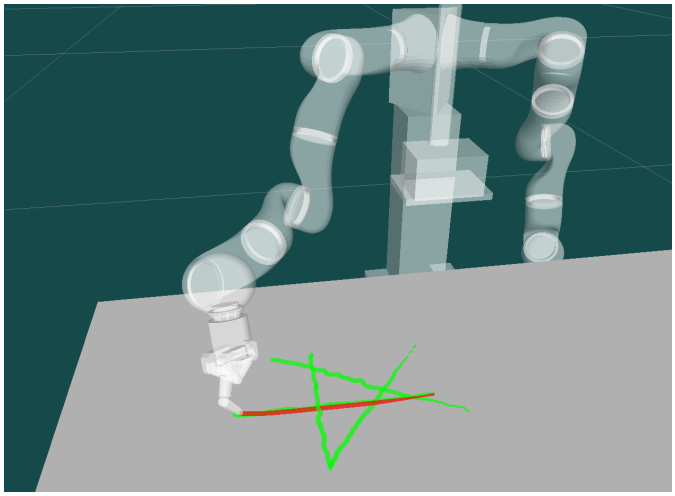
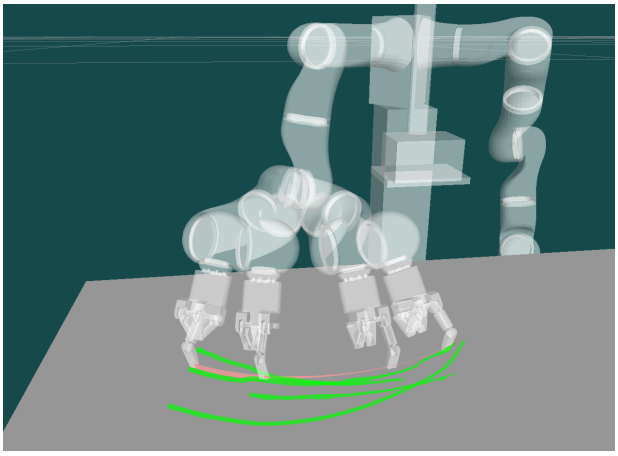


Figure 5.5: Visualization of demonstrations and learned policy for straight end-effector motions.

are given that make mostly use of a subset of the degrees of freedom. Sets of demonstrations are given for pointing movements between two points on the table where mostly the shoulder, elbow or wrist joints are used. To allow the policy to represent this kind of characteristics, the standard objective function, as detailed above, is augmented by individual velocity penalties for each joint.

The reinforcement policy search is expected to penalize those weights that have not been used in the demonstrations in order to prioritize movements of the remaining joints. The results from two experiments are shown in figure 5.6 and figure 5.7. The RViz visualization of the robot is shown together with all demonstrations (only the end-effector position is visualized as a green line) and the solution, produced from the learned policy for one of the demonstrated problems.

In figure 5.6, mainly the forearm is moved to reach the goal point. As it can be seen from the four superimposed intermediate robot positions from the solution execution, both the upper arm and the shoulder are mostly fixed. Additionally, the policy incorporates the demonstrated hand and wrist posture in order to touch the table with its fingertip but not to penetrate the table any further. The end-effector also remains on the table surface at all times. This behavior is not only valid for one of the given demonstration scenarios but fully represented by the policy parametrization as given in table 5.6. The offset between the upper respectively lower wrist and the table which has been learned from the demonstrations makes sure that the end-effector does not penetrate the table. The velocity penalties for the individual joints are chosen in a way that favors the elbow joints.



Learned parameters	
Velocity penalty	θ_{joint_i} [-]
Joint 1 (shoulder)	16.06
Joint 2	7.33
Joint 3	**0.02**
Joint 4	7.05
Joint 5	**0.03**
Joint 6	7.98
Joint 7	3.96
Joint 8 (finger)	4.03

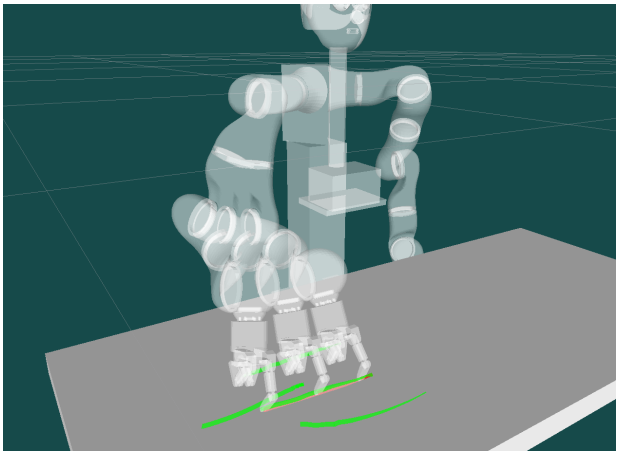
Figure 5.6: Four demonstrations (green) of pointing movements on the tabletop and the learned optimal policy executed by the Apollo robot arm (red). The predominant rotation of the elbow joints has been learned from the demonstrations. The predominant joints identified by low velocity penalties are highlighted.

Similar results have been achieved in case of motions produced by shoulder rotations as shown in figure 5.7. Again, four demonstrations are given and the corresponding end-effector traces are shown as green lines. The learned optimal policy generates a motion based on a shoulder rotation. This behavior is clearly visible for the generated solution as shown by the superimposed arm configurations but also in the resulting policy parametrization as listed in table 5.7.

5.3 Sketched End-Effector Demonstrations

Demonstrating precisely the desired behavior on the entire robotic system is usually impossible due to several reasons. Most times the kinematic of the robotic system is not compatible with our human kinematic and we can not manipulate all joints simultaneously. Additionally, human motions guiding the demonstrations tend to be imprecise. Moreover, they might not be smooth because repositioning might become necessary during the demonstration in order to grip a different part of the robot.

The following experiment will explore the ability of this approach to learn from consciously incomplete demonstrations that focus on some special aspect of a motion. In the given example, the behavior of the end-effector when approaching and operating around obstacles should be learned. To define a desired motion, we sketch end-effector trajectories as shown in figure 5.8 and figure 5.9. In this experiment, the robot is taught



Learned parameters	
Velocity penalty	θ_{joint_i} [-]
Joint 1 (shoulder)	3.12
Joint 2	**0.16**
Joint 3	31.14
Joint 4	**0.01**
Joint 5	2.18
Joint 6	33.49
Joint 7	4.60
Joint 8 (finger)	6.26

Figure 5.7: Four demonstrations (green) of pointing movements on the tabletop and the learned optimal policy executed by Apollo's robot arm (red). The predominant rotation of the shoulder joints has been learned from the demonstration. The predominant joints identified by low velocity penalties are highlighted.

two different ways to approach a cylindrical obstacle from those sketched demonstrations. The first set of trajectories has in common that the end-effector avoids the obstacle as long as possible and approaches it in a perpendicular way, whereas in the second set of trajectories, the end-effector describes a straight line towards a goal point on the obstacle's surfaces.

For both motion types, 24 sketched end-effector demonstrations are given, partitioned into a training set of size six and a test set of size 18 to cross-validate the resulting optimal policy. The optimal policy is learned from all training demonstrations by using the averaged distance between the end-effector's trace and the sketched trajectory as loss function.

As shown in figure 5.8 and figure 5.9, the Riemannian representation of the workspace geometry for those scenarios consists of a superposition of the ambient euclidean system and the cylindrical system surrounding the cylinder obstacle. Both systems are blended into each other where the influence of the latter one vanishes at a certain distance from the cylinder. For this experiment, velocity penalty parameters are introduced for each coordinate system and each coordinate individually. Therefore, motions around the cylinder can be penalized differently than motions approaching the cylinder.

The average fitness for both learned motion policies and one initial random policy evaluated for all 24 demonstrated scenarios is shown in figure 5.10. Both learned policies clearly outperform the initial random policy by a factor of four. The learned policy does not only perform well on its training examples but shows similar fitness on the test

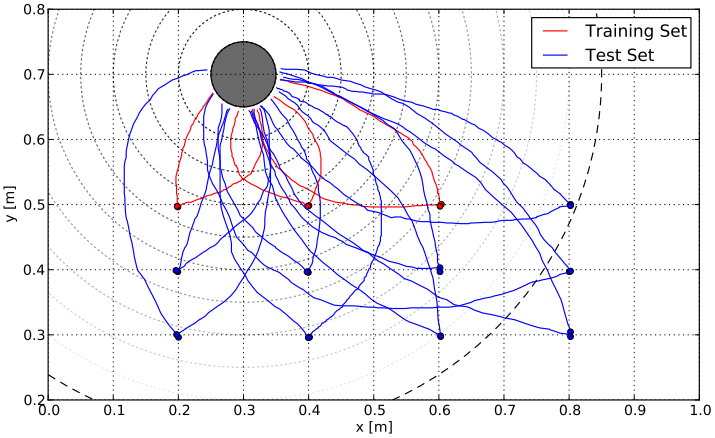


Figure 5.8: Training and test demonstrations of pointing movements towards a cylinder with high obstacle avoidance. Only the end-effector positions are shown as hand-drawn trajectories.

problems. The learned behavior generalizes from training examples to similar test cases. Comparing the policy learned for low obstacle avoidance and the one for high obstacle avoidance reveals that each of them performs clearly better for its own problem type and worse on the other one. The chosen fitness function is therefore appropriate to separate both types of behavior.

5.4 Imposing Additional Objectives

As discussed earlier, the given demonstrations may not be sufficient to fully define the motion or to achieve a certain objective on the real system. To resolve redundancy, an additional objective can be introduced. This objective can affect arbitrary aspects of the motion since only the evaluation of the objective function and no additional gradient information for a given trajectory and context is relevant for the optimizer. The loss function of reinforcement learning, previously formulated to imitate a demonstration, is therefore augmented by an additional term as discussed in section 3.2.1. In this example, the average height of the elbow should be minimized in order to maintain a 'natural' arm position.

Since only the end-effector position in workspace coordinates is available from observed behavior, the average euclidean distance of the current policy's solution to this point is employed. The forward kinematic map of the robotic system is given by ϕ.

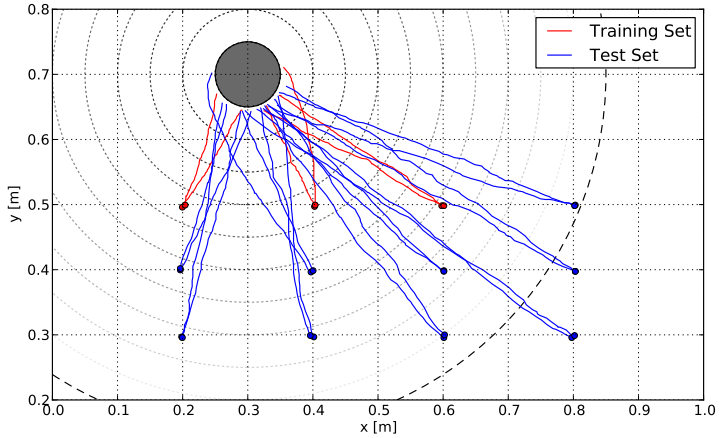


Figure 5.9: Training and test demonstrations of pointing movements towards a cylinder with almost no obstacle avoidance. Only the end-effector positions are shown as hand-drawn trajectories.

$$\mathcal{L}(\xi_{gen}, \xi_{i=0}^{k}) = \frac{1}{k}\sum_{i=0}^{k}(d_{AvgEucl}(\xi_{gen}, \xi_i) \quad + \quad \mathcal{L}_{elbow}(\xi_{gen}, \xi_i)) \tag{5.1}$$

$$= \frac{1}{k}\sum_{i=0}^{k}\left(\frac{1}{T_{gen}+1}\sum_{t=0}^{T_{gen}}\|\phi_{end}(\mathbf{q}_i^{(t)}) - \phi_{end}(\mathbf{q}_{gen}^{(t)})\| + (\phi_{elbow}(\mathbf{q}_{gen}^{(t)}) - \mathbf{p})\cdot\mathbf{n}\right) \tag{5.2}$$

The same scenario as presented in the last experiment (cf. section 5.3) is used here. Sketched end-effector trajectories are employed to demonstrate two types of motions when approaching a cylindrical obstacle. The set of demonstrations is again split into 6 training demonstrations and 18 test demonstrations. For each motion type, a policy is learned from the pure loss function as presented in the last experiment and from the combined loss function which additionally contains the contribution of the elbow loss term.

For all possible combinations of two motion types and two loss functions, a total of four policies is learned whose fitness is visualized in figure 5.11. Additionally, the fitness of an initial random policy (orange) is shown for comparison. For each policy, the contribution of the end-effector distance (yellow) between demonstration and solution trajectory as well as the contribution of the elbow loss term (red) to the overall loss function is plotted. The resulting elbow loss is also displayed for the policies that have been learned using the pure loss function which does not consider the elbow height (for these policies, the elbow loss is shown in light red).

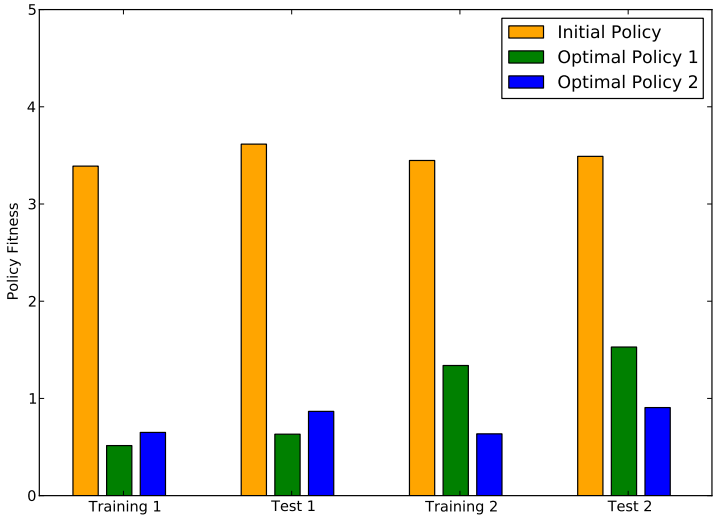


Figure 5.10: Comparison of policy fitness for two motion types cross-validated on training and test data.

The proposed policy search algorithm PI^2 CMA-ES demonstrates in this experiment several capabilities that we discussed at the beginning of this chapter:

Imitation of Observed Behavior A policy learned on a given set of training demonstrations is able to reproduce the observed behavior. On the same set of problems, the optimal policy clearly outperforms the initial random policy but also the policy which learned a different type of motion. This is true for both the policy learned using the pure loss function (as seen in the previous example) and the policy learned using the combined combined loss function.

Generalization to Similar Scenarios Concerning the fitness of the learned policies on the corresponding test problems, the policy learned for this type of motion clearly performs better than the random policy and the policies learned for the other motion type. This is especially the case for the policies learned for motion type 2 which perform significantly better on their own test set compared to the performance of the motion type 1 policies on this test set.

Optimizing Additional Objectives The policies learned using the combined loss functions can be directly compared to the ones which have been learned using the pure loss functions as they are plotted next to each other. The algorithm was capable to utilize the system's redundancies in case the combined loss function has been used. The policies

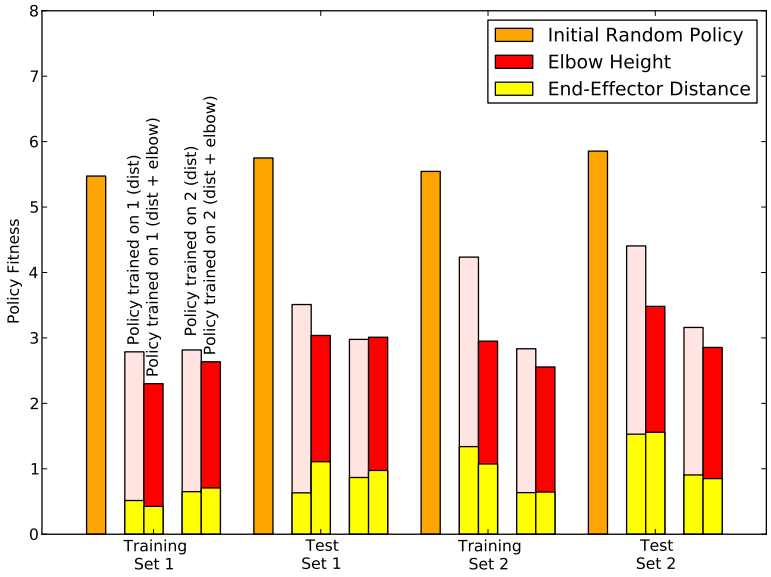


Figure 5.11: Cross-validation of the fitness of policies trained for two types of motions (high and low obstacle avoidance when pointing to a cylindrical obstacle) under two different loss functions (with and without consideration of the elbow height).

learned with the help of the combined loss function outperform the ones learned with the help of the pure imitation loss in terms of combined loss. In particular, the contribution of the elbow loss is significantly reduced for policies learned from the combined loss formulation. This result is not only visible in the fitness of the policies on the training set but it also generalizes to the policies' application to the according test set.

5.5 Effects of Optimizer Settings

The effects of the domain size on the optimizer's performance (e.g. in terms of convergence and best fitness, cf. section 4.3.2) have been evaluated on four learning problems. From the experiments on the real robotic system as presented above, the following motion types have been learned:

1. Task 1: Pointing to a cylinder (high obstacle avoidance); section 5.3
2. Task 2: Pointing to a cylinder (high obstacle avoidance) + additional minimization of the elbow height; section 5.4
3. Task 3: Motions with predominant elbow joint (forearm motion); section 5.2.2
4. Task 4: Straight pointing motions on the table; section 5.2.1

The range of the domain has been varied for all those experiments from $[0, 1]$ and $[0, 5]$ up to $[0, 10]$. No active constraints have been put on the domain's limits. Instead, the initial guess $\boldsymbol{\theta}_0$ has been chosen randomly within the domain and the initial step size has been set to $\sigma_0 = (\max - \min)/5$.

The results of evaluating the performance for those experiments are shown in figure 5.12. For each task, the optimal policy is computed based on the three different domain ranges. For each task, the bars represent from left to right the results of the experiment carried out for increasing domain ranges. The results are averages over at least 4 independent runs. The error bars display the 1σ standard deviation.

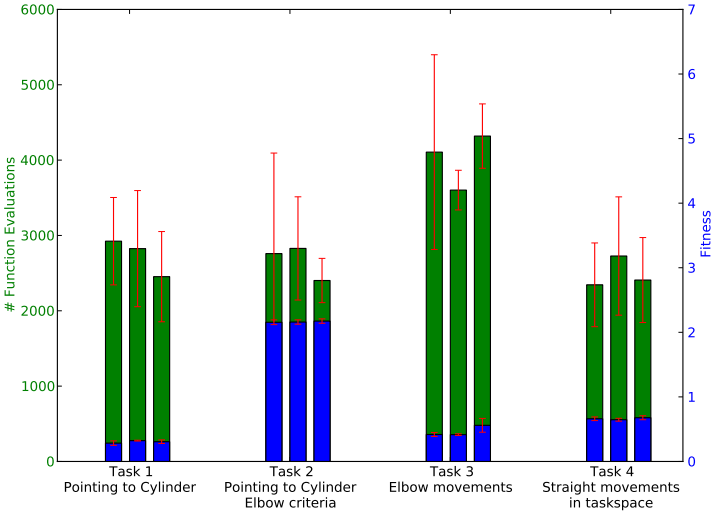


Figure 5.12: Effects of domain size and problem dimensionality on the optimization speed and quality. The average number of necessary function evaluations and the resulting fitness is evaluated on four motion problems. Three different initial configurations (step size and initial parametrization) are shown for each task.

Neither the number of function evaluations (shown in green) nor the resulting fitness (shown in blue) is significantly influenced by the initial domain size. The PI^2-CMA turns out to be robust against slightly inappropriate initial configurations. The automatic adaptation of exploration direction and magnitude is capable to find a similar solution in all evaluated cases.

Similar results have been achieved by employing the BiPop CMA-ES optimization method. No evident improvement of the global fitness could be achieved. In case of a reasonable parameter encoding as described in section 4.3.2, the pure CMA-ES setup is sufficient to explore the required parameter space.

6 Conclusion

In this work, a novel learning algorithm for motion policies has been presented and evaluated on synthetic and real-world experiments. The reinforcement toolbox has been successfully leveraged to represent the imitation learning problem for motor primitives as well as the consecutive improvement of the learned motion policy with respect to additional requirements. The underlying representation of a motion policy as the solution of an optimal control problem was utilized to parametrize a class of movements in a naturally understandable way. A derivative-free optimizer, CMA-ES, was employed to learn the optimal policy, in this case the optimal parametrization, in order to fit the demonstrated behavior and to resolve redundancies within the system by optimizing with regard to additional desired motion characteristics. The usage of probability-weighted averaging of policy roll-outs as an optimization method was motivated by previous work within the field of stochastic optimal control.

The algorithm proved its capability to learn motion policies which can not only imitate behavior within the specific setting of the given demonstrations but also to generalize from observed behavior. Cross-evaluations based on different types of demonstrated behavior demonstrated the reproducibility of learned motion characteristics also on new and unseen test scenarios.

6.1 Future Work

6.1.1 Trajectory Roll-Outs

The execution of computed discrete trajectories on the actual robot but also on the Apollo simulator as provided by SL revealed significant discrepancies between the computed configurations and the real motion. The real system's response is is limited by the performance of the underlying low-level controllers. Additional noise on torque and angle sensors decreases the precision. The trajectory is not executed as a continuous stream of configurations by one controller per joint but as a discrete set of configurations.

To improve the performance of the solution trajectory generated by the learned policy, a slight modification to the presented policy search should be analyzed. By evaluating the high-level objective function on the simulated trajectory roll-outs (e.g. on the SL Apollo simulator), the learned policy is expected to avoid commands that are infeasible on the real system and would lead to a poor imitation performance.

6.1.2 Combination with Geometric Motion Representation

Several motion primitives encountered even in the simplest problems are hard or even impossible to express as the result of an optimization procedure. As an example, the fosbury flop, nowadays the standard motion in high jump, may or may not be the 'best' solution to this motion problem. However, the formulation of the underlying physical motivation for precisely this motion based on the conservation of energy or other basic principles seems to be almost impossible. For certain parts of motion policies, it would be more favorable to combine the presented approach with concepts with concepts geared to learning motion primitives in a geometric context. Specialized shapes may be easier to address by learning and improving in a geometric space as shown in DMP approaches.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature