Learning Replanning Policies with Direct Policy Search

Lernen umplanender Strategien mit direkter Strategiesuche
Master-Thesis von Florian Brandherm
Tag der Einreichung:

1. Gutachten: Jan Peters
2. Gutachten: Riad Akrour

TECHNISCHE UNIVERSITÄT DARMSTADT

Learning Replanning Policies with Direct Policy Search
Lernen umplanender Strategien mit direkter Strategiesuche

Vorgelegte Master-Thesis von Florian Brandherm

1. Gutachten: Jan Peters
2. Gutachten: Riad Akrour

Tag der Einreichung:

Erklärung zur Master-Thesis gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden. Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, den 29. Mai 2018

____

(Florian Brandherm)

Abstract

In robot learning, episodic policy search is typically used to plan trajectories before executing them. Yet, common characteristics of real-world robotic tasks are partial observability and perturbations of the system state. Planning a trajectory based on such an inaccurate state measurement before executing the movement evidently leads to poor accuracy, especially with dynamical problems like table tennis or ball catching. Moreover, this state-of-the-art approach cannot react to external perturbations in the state. Therefore, it is sensible to replan trajectories during the movement, based on updated state measurements. This thesis presents a framework to learn replanning contextual policies with the ability to adapt to external perturbations and updated measurements via direct policy search. This is followed by a comparison of two different approaches to formulate such replanning policies. It is demonstrated that replanning policies can be learned efficiently by direct policy search and outperform non-replanning policies for problems with partially observable or perturbed states.

Zusammenfassung

Für lernende Roboter wird häufig eine episodische Strategiesuche durchgeführt, um Trajektorien im voraus zu planen. Allerdings sind partielle Beobachtbarkeit und Störeinflüsse charakteristisch für reale Probleme. Werden Trajektorien auf der Basis von solchen unvollständigen Messungen vor ihrer Ausführung vorausgeplant, führt dies in der Regel zu suboptimalen Ergebnissen. Das trifft besonders auf dynamische Probleme wie Tischtennis oder das Fangen von Bällen zu. Darüber hinaus können vorausgeplante Trajektorien nicht auf externe Störeinflüsse reagieren. Bei veränderter Informationslage aufgrund neuer Messungen ist daher ein Umplanen von Trajektorien sinnvoll. Diese Thesis stellt ein Konzept vor, umplanende, kontextabhängige Strategien direkt zu lernen. Die gelernten Strategien sind in der Lage, Trajektorien während der Durchführung an externe Störeinflüsse und neue Messungen anzupassen. Dazu werden zwei Varianten solcher Strategien vorgestellt und verglichen. Es wird außerdem gezeigt, dass umplanende Strategien effizient direkt gelernt werden können und das Potential haben, rein vorausplanende Strategien zu übertreffen, wenn partielle Beobachtbarkeit oder externe Störeinflüsse eintreten.

Keywords: robot learning, robot control, policy search, motion planning

Contents

1 Introduction

The application of reinforcement learning to learn robot motion trajectories has become widespread [2, 12, 20]. This thesis proposes a novel method to learn policies that can act in complex and changing environments by rapidly adapting to new observations.

More and more robots need to act in complex, uncertain environments. As robots leave the confines of rigid industrial settings and migrate to less controlled environments, it is crucial to be able to adapt motion trajectories to a variety of parameters. Performing motor tasks, such as picking up an object in an uncontrolled environment, pose the question how a motion can be generalized in order to adapt to a range of related problem settings, e.g., picking up an object from different locations and orientations. Other examples of such problems are catching or hitting a ball. This requires a robot to quickly adapt its motion to a variety of different ball trajectories since each throw results in a unique ball trajectory.

In order to achieve such an adaptation, a *contextual policy* must be learned. Such a contextual policy maps a set of context features to a set of trajectory parameters. Trajectory parameters define the trajectory of a robot motion while context features define the task setting. Context features are usually hand-crafted to suit a specific task. Learning a contextual policy can be viewed as a multi-task learning problem where each set of context features defines a separate task. For robotic tasks, a common choice for the policy is a linear model since it is simple enough to be learned efficiently while the right choice of features usually facilitates models that are adequate for robotic tasks.

Contextual policies can be learned in a supervised setting, imitating a collection of demonstrated trajectories. However, due to limitations of physical robot systems and trajectory representations, the imitation of trajectories is often imperfect [3]. Furthermore, providing good demonstrations is a difficult task for humans, considering the robot must be moved manually. It is especially difficult if the underlying task is subject to strong timing constraints, as is the case with highly dynamic tasks such as catching or hitting a flying ball. By definition, the performance of any policy that was learned through imitation can only be as good as the demonstrations [3]. Thus, *reinforcement learning* offers the potential to improve the performance of imitation-learned policies by finding better policies through iterative improvement by trial and error [3, 11].

Applying a contextual policy requires measuring the state of the system and/or the outside world in order to determine an action. In real-world scenarios, the external system state is typically only partially observable or affected by unforeseeable perturbations. Unforeseeable changes of the state can also be the result of hidden states that are not observable. For example, the spin of a ball can usually not be measured, yet it can cause erratic behavior when the ball bounces off a surface. If the available information at the beginning of a motion is incomplete or subject to later changes, it is clear that sufficient performance cannot be achieved by strictly planning trajectories ahead. Therefore, in many real-world scenarios, trajectories must react online to updated context measurements which can contribute additional information that was not available previously. Updated context measurements can also indicate unforeseen changes in the task definition. For example, in a real-world table tennis task, a good paddle trajectory cannot be determined by merely measuring the position and velocity of the ball prior to the bounce on the table. The unobservable spin can cause behavior that cannot be predicted with the available information. However, measuring the position and speed of the ball only after the bounce will not leave enough time to execute a motion that returns the ball. This is one example that highlights the necessity for continuous adaption to changes.

1.1 Problem Statement

The question this thesis aims to answer is how single-stroke robot motions can be corrected during the execution as a response to changes in the context. In many real-world tasks, such necessity for corrections arises from state perturbations or partial observability. However, the goal is to present a method that only results in a moderate increase in the number of learned parameters to preserve similar learning performance as can be achieved without corrections. It is a relevant question in robotics since many real-world tasks suffer from the aforementioned issues while most state-of-the-art approaches that can react to them either require to learn significantly more complex policies or hand-crafted adaptation mechanisms.

The contribution of this thesis is twofold. First, *replanning policies* are presented, which are a framework to learn policies that are able to adapt standard trajectory representations to changes. Second, replanning policies are analyzed with specially designed as well as realistic experiments. *Replanning* denotes the online correction of trajectory parameters during the motion execution. It is shown that it is possible to learn replanning policies with the same number of parameters as respective policies without replanning. These policies can be applied to the category of problems with stationary state distributions under the effects of perturbations and corrected observations. Under these circumstances, such policies are able reach far better performance with the same number of parameters. This thesis also presents how replanning policies can be learned with small modifications to the state-of-the-art policy search method MORE [1].

1.2 Thesis Structure

This thesis begins with a review of the necessary background and related work in Section 2. Section 3 presents the concept of replanning policies and explains how they are learned. Next, Section 4 explains how replanning policies are initialized and used to replan trajectories. A report of the experiments is given in Section 5. Section 6 discusses the shortcomings of the presented approach and suggests directions for future research to alleviate the problems. The thesis ends with a conclusion in Section 7.

2 Background and Related Work

From the robot's perspective, the issues of partial observability and state perturbations present the same challenge: The context, and with it the task definition, may change during the execution of a motion trajectory. To achieve robust performance, it is crucial to address such changes in the context by adapting trajectories online. This thesis presents a scheme for using direct policy search in order to learn replanning policies that can adapt a previously planned trajectory, reacting to changes in the context. This chapter gives a short introduction to the underlying concepts and algorithms that serve as a foundation for the contribution of this thesis. It also contains an overview of related approaches to address the online adaptation of robot motions.

2.1 Reinforcement Learning

Complex environments, in which reinforcement learning is applied, are typically modeled by a *Markov Decision Process* (MDP) [21]. Since this thesis focuses on single-stroke robot motions, a finite-horizon MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with continuous states and actions can be assumed, with a state space $\mathcal{S} \in \mathbb{R}^{d_s}$, an action space $\mathcal{A} \in \mathbb{R}^{d_a}$, a transition probability distribution $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. A fixed finite horizon $T \in \mathbb{N}$ is the number of time steps until a motion is completed and the state is reset to an initial state. The goal of reinforcement learning is the optimization of a policy $\pi(\mathbf{a}|\mathbf{s})$ that maximizes an expected cumulative reward $J(\pi) = \mathbb{E}_{\mathbf{a}_t \sim \pi_t(\mathbf{a}_t|\mathbf{s}_t)}[\sum_{t=1}^{T} \mathcal{R}_t(\mathbf{a}_t, \mathbf{s}_t)]$ over all time steps $t \in 1, \ldots, T$. In robotic tasks, $\mathbf{s}$ could, for example, be the current joint angles and angular velocities, as well as external states of the environment. A typical example for the actions $\mathbf{a}$ are motor torques.

An essential challenge is that each evaluation of a policy requires the execution of a motion on the physical robot. This typically takes at least several seconds, including the time for the task setup. Thus, each policy evaluation is extremely expensive. Many reinforcement learning algorithms derive a policy from a learned Q-function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ that indicates the quality of an action in a given state and can be used to derive a policy [9]. In robotics, however, it is often impractical to explore the entire state-action space to learn a global Q-function due to the high sample-cost and a large search space with continuous states and actions.

2.1.1 Policy Search

Policy search [4], the direct, local optimization of the policy π, is an established approach in robotics. One reason for the success of policy search methods is the ability to handle continuous and high-dimensional state-action spaces. Another reason is that policy search algorithms have shown to be particularly useful in combination with low-dimensional trajectory parameterizations [11]. These are often structured in such a way that only a subset of trajectory parameters has to be generalized by the policy which further reduces the search space. Due to their explicit formulation, initial trajectories and policies can easily be learned from demonstrations via supervised learning methods. In typical applications (including this thesis), policy search algorithms learn a policy that maps context features $\boldsymbol{c}$ of the state to trajectory parameters $\boldsymbol{\theta}$. Because the parameters for an entire trajectory are determined at the beginning of a movement, the horizon $T = 1$ is only a single step. For the sake of distinction from arbitrary states $\mathbf{s}$ and actions $\mathbf{a}$, the symbols $\boldsymbol{c}$ and $\boldsymbol{\theta}$ will be used instead. A context $\boldsymbol{c}$ may contain features of an observed system state or a definition of the desired behavior. The distribution of this context is generally considered unknown.

The idea of policy search is to iteratively improve an initial policy by alternating between a *policy evaluation* step and a *policy update* step. In the evaluation step, exploratory samples are generated based on the current policy. Using these samples, a policy update is computed such that its reward for the set of samples is improved. To compute such an update, the policy requires some variance to explore the solution space. Therefore it is common to use a *stochastic* policy [4]. A common choice for such a policy is a multivariate Gaussian distribution around a linear function (linear Gaussian policy) $\pi(\boldsymbol{\theta}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\boldsymbol{c} + \mathbf{b}, \boldsymbol{\Sigma})$ with a linear gain $\boldsymbol{K} \in \mathbb{R}^{d_{\boldsymbol{\theta}} \times d_{\boldsymbol{c}}}$, a bias vector $\mathbf{b} \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d_{\boldsymbol{\theta}} \times d_{\boldsymbol{\theta}}}$. The goal of policy search is to learn a contextual stochastic policy $\pi(\boldsymbol{\theta}|\boldsymbol{c})$ directly while treating the problem as a black box.

There are two variants of policy search: model-based and model-free [4]. The idea of the model-based approach is to decrease the number of required real-world sample trajectories by learning a model of the task that allows to generate artificial samples. Subsequently, the artificial samples from the model (e.g., a simulation) are used to optimize the policy off-line. While this approach can lead to significant increases in sample-efficiency, the learned models are often inaccurate which can result in poor real-world performance [4]. The model-free approach optimizes the policy directly on real-world samples. While the sample-efficiency is much lower, direct optimization is not subject to approximation errors of a model. In reality, the effort of learning accurate models is often outweighed by the simplicity of sample generation. This thesis focuses on model-free policy search.

A variety of direct policy search algorithms have been suggested [4]. One example of such an algorithm is relative entropy policy search (REPS) [18]. It is a model-free algorithm that introduces an information-theoretic bound to the optimization of the expected reward to limit the loss of information when updating the policy.

Building on REPS, Kupcsik *et al.* [13] introduced GPREPS, a model-based algorithm that utilizes a Gaussian process reward model in order to improve data efficiency. In each iteration, after sampling a number of trajectories, the context distribution and a Gaussian process reward model is estimated. Then, the learned models are used to generate a high number of artificial samples to perform the REPS policy update.

2.1.2 Contextual MORE

Model-based relative entropy stochastic search (MORE) [1] improves upon REPS by learning a local quadratic model of the reward function, enabling a closed solution for the policy update. While the original version presented by Abdolmaleki *et al.* [1] is not contextual, MORE was extended to the contextual case by Akrour *et al.* for their MOTO-algorithm [2] (see Section 2.1.3). That contextual variant of this state-of-the-art algorithm was chosen to be the foundation for this thesis. However, the principles of replanning policies could be applied to other policy search algorithms as well.

The goal in policy search is to learn a stochastic policy $\pi(\boldsymbol{\theta}|\boldsymbol{c})$ that maximizes the expected reward $J(\pi) = \mathbb{E}_{\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta}|\boldsymbol{c})}[\mathcal{R}(\boldsymbol{\theta}, \boldsymbol{c})]$. The reward function $\mathcal{R}(\boldsymbol{\theta}, \boldsymbol{c}) \mapsto \mathbb{R}$ maps parameters, chosen by the policy depending on the context, to a real number that represents the quality of that choice. It is defined carefully as a goal definition of the problem at hand. If the parameters define a motion trajectory, $\mathcal{R}(\boldsymbol{\theta}, \boldsymbol{c})$ scores the quality of the generated trajectory.

In the policy evaluation step, a fixed number of contexts and resulting parameters are sampled from the current policy $\pi^i(\boldsymbol{\theta}|\boldsymbol{c})$, executed on the task and scored by the reward function $\mathcal{R}(\boldsymbol{\theta}, \boldsymbol{c})$. Then, in the policy update step, the set of context and parameter samples and their respective rewards is then used to improve the expected reward of the policy distribution, shifting it towards more successful parameters samples.

The updated, improved policy $\pi^{i+1}(\boldsymbol{\theta}|\boldsymbol{c})$ is obtained by solving the optimization problem given by

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \iint \mu^i(\boldsymbol{c})\pi(\boldsymbol{\theta}|\boldsymbol{c})\mathcal{R}(\boldsymbol{\theta},\boldsymbol{c})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{c}, \\ \text{subject to} \quad & \mathbb{E}_{\boldsymbol{c}\sim\mu(\boldsymbol{c})}\left[\mathrm{KL}(\pi(\boldsymbol{\theta}|\boldsymbol{c}) \parallel \pi^i(\boldsymbol{\theta}|\boldsymbol{c}))\right] \leq \epsilon, \\ & \mathbb{E}_{\boldsymbol{c}\sim\mu(\boldsymbol{c})}\left[\mathcal{H}\left(\pi(\boldsymbol{\theta}|\boldsymbol{c})\right)\right] \geq \beta. \end{aligned}$$

with $\mu^i(\boldsymbol{c})$ being the current empirical state distribution that was obtained from the evaluated samples. The optimization determines the policy that maximizes the expected reward while bounding the policy changes. The Kullback-Leibler (KL) divergence of the policy update is bounded to limit the rate of convergence. The KL-divergence $\mathrm{KL}(p \parallel q) = \int p(x)\log\frac{p(x)}{q(x)}$ is a measure for the divergence of the probability distribution p from q. Analogous to gradient ascent optimization, convergence can only happen if the step size is sufficiently small. Furthermore, placing a lower bound on the entropy of the policy update limits the reduction of the policy's covariance, which is necessary to force more exploration and can avoid premature convergence to local maxima. The entropy of a distribution p is given by $\mathcal{H}(p) = -\int p(x)\log p(x)$.

This optimization problem can be solved using the method of Lagrange multipliers, yielding a closed solution for the policy update which is given by

$$\pi^{i+1}(\boldsymbol{\theta}|\boldsymbol{c}) \propto \pi^i(\boldsymbol{\theta}|\boldsymbol{c})^{\eta/(\eta+\omega)} \exp\left(\frac{\mathcal{R}(\boldsymbol{\theta},\boldsymbol{c})}{\eta+\omega}\right), \tag{1}$$

where η and ω are Lagrange multipliers. In order to be able to obtain a closed solution for the corresponding dual function, a quadratic reward model is learned from the samples. The reward model is defined as

$$\mathcal{R}(\boldsymbol{\theta},\boldsymbol{c}) \approx \begin{pmatrix}\boldsymbol{\theta}\\ \boldsymbol{c}\end{pmatrix}^T \underbrace{\begin{pmatrix}\boldsymbol{R}_{\theta\theta} & \boldsymbol{R}_{\theta c}/2 \\ \boldsymbol{R}_{\theta c}{}^T/2 & \boldsymbol{R}_{cc}\end{pmatrix}}_{\text{symmetric}} \begin{pmatrix}\boldsymbol{\theta}\\ \boldsymbol{c}\end{pmatrix} + \begin{pmatrix}\boldsymbol{\theta}\\ \boldsymbol{c}\end{pmatrix}^T \begin{pmatrix}\mathbf{r}_\theta\\ \mathbf{r}_c\end{pmatrix} + r_0 \tag{2}$$

and can be learned from the samples with standard supervised learning methods like ridge regression. Akrour *et al.* [2] showed that with this quadratic model and assuming a linear Gaussian policy $\pi(\boldsymbol{\theta}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\boldsymbol{c} + \mathbf{b}, \boldsymbol{\Sigma})$, the policy update (Eq. 1) simplifies to

$$\pi^{i+1}(\boldsymbol{\theta}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{\theta}| \underbrace{\boldsymbol{F}\boldsymbol{L}}_{\boldsymbol{K}^{(i+1)}}\ \boldsymbol{c} + \underbrace{\boldsymbol{F}\mathbf{f}}_{\mathbf{b}^{(i+1)}}, \underbrace{\boldsymbol{F}(\eta+\omega)}_{\boldsymbol{\Sigma}^{i+1}})$$

with $\boldsymbol{F} = (\eta\boldsymbol{\Sigma}^{-1} - 2\boldsymbol{R}_{\theta\theta})^{-1}$, $\boldsymbol{L} = \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{K} + \boldsymbol{R}_{\theta c}$ and $\mathbf{f} = \eta\boldsymbol{\Sigma}^{-1}\mathbf{b} + \mathbf{r}_\theta$.

The key difference between MORE, REPS and GPREPS is the optimization of the dual function. While REPS and GPREPS optimize the dual function based on their real and artificial samples respectively, the quadratic reward model allows for a closed analytical solution.

2.1.3 MOTO

The aforementioned policy search algorithms learn a policy that determines trajectory parameters $\boldsymbol{\theta}$ from an initial context $\boldsymbol{c}$. But some problems in robotics cannot be solved by such a static trajectory and require continuous reaction to the state. MOTO [2] is an efficient method to learn such reactive policies for MDPs with direct policy search. In contrast to the above policy search approaches, the goal is not to learn a policy that determines trajectory parameters $\boldsymbol{\theta}$ but to learn a policy that controls a robot directly via actions $\mathbf{a}$ (i.e., joint torques). The algorithm learns a set of time-dependent linear subpolicies

$\pi_1(\mathbf{a}|\mathbf{s}), \pi_2(\mathbf{a}|\mathbf{s}), \ldots, \pi_T(\mathbf{a}|\mathbf{s})$ that directly control a robot at a series of time steps $1, 2, \ldots, T$. Building on the foundations of contextual MORE, a local quadratic approximation the time-dependent Q-function is learned and used to update the individual subpolicies.

Learning a quadratic approximation of a Q-function in this setting implies a large number of parameters that have to be learned because the number of parameters grows quadratically with the sum of actions (e.g., joint torques) and contexts (e.g., full robot state and external context) and linearly with the number of time steps. Moreover, to generate complex trajectories with such direct control policies, a sub-policy must be learned for many time steps. In their experiments, Akrour *et al.* used up to 50 sub-policies per second. Overall, this results in a significant number of parameters that have to be learned, implying a challenging sample complexity. To alleviate this issue, a lot of work went into sample reuse via the backpropagation of the Q-functions.

The work of Akrour *et al.* (MOTO) [2] has some similarities to the approach that is presented in this thesis (Section 3). An important difference to the approach that is presented here is that Akrour *et al.* use a time-dependent policy for direct robot control, while the present approach uses such a policy for the planning and adaption of a low-dimensional parameterized trajectory. This not only reduces the number of learned parameters but also exploits advantageous properties of the trajectory representation, such as guaranteed smoothness and shape-preserving generalization to a variety of goal positions. Furthermore, the use of a parameterized trajectory enables complex motions with much fewer subpolicies and therefore fewer parameters. Another important difference is that MOTO relies on a quadratic model of the Q-function that is estimated for each controlled time-step, while replanning policies utilize a single joint reward model for all subpolicies. This is because the approach that is proposed in thesis assumes a single reward for the whole trajectory, while MOTO assumes per-step rewards.

2.2 Online Adaptation of Motion Trajectories

Many real-world robotic tasks suffer from partial observability or perturbations after the initial measurement of the context. One method to learn a policy that is able to adapt to such imperfect information is to learn a control policy like MOTO does. However, the number of parameters that have to be learned is a determining factor for the sample-efficiency of a learning task. For this reason, prestructured, low-dimensional trajectory representations are often preferred.

2.2.1 Optimal Stopping

One approach to improve the performance under uncertain or changing contexts is to delay the planning of a trajectory until enough information is collected or the trajectory has to be started because of timing constraints. This approach was discussed by Wang *et al.* [23] using the example of robot table tennis. In this case, the optimal time must be determined to stop waiting and start the trajectory under partial observability. However, this fails in cases where the context measurements are not accurate enough at the time a trajectory's execution must be started because of timing constraints.

2.2.2 Dynamical Trajectory Representations

Defining trajectories rigidly, e.g., based on splines [14] has the disadvantage that trajectories are precomputed and cannot react online to perturbations. Encoding trajectories in terms of control policies [1, 17] assumes that the desired control policy follows a single trajectory. This approach has difficulties with large perturbations of the trajectory that are caused by their strong dependence on time, leading to potentially dangerous behavior [20]. Therefore, many state-of-the-art approaches to adapt online to changing system states use dynamical trajectory representations that are formed by attractor landscapes [7, 8, 10, 12, 15, 16, 20]. A dynamical trajectory is not defined by a fixed path but by a forcing field.

The formulation as a differential equation system generally produce trajectories that are well-suited for robots. Any joint trajectory that should be followed by a robot must be continuous. This is a core strength of dynamical trajectory representations which are typically modeled to emulate spring-damper systems and therefore react to forces with inertia. Such behavior naturally results in smooth trajectories. Jerky movements have the potential to cause wear and damage to physical robots. To avoid this, trajectories must be smooth which means that the joint velocities must not contain discontinuities. This property is satisfied as well by dynamical trajectory representations. Since dynamical trajectories are typically not precomputed but computed during the execution, it is possible to react to perturbations online [7, 10, 16].

2.2.3 Dynamical Movement Primitives

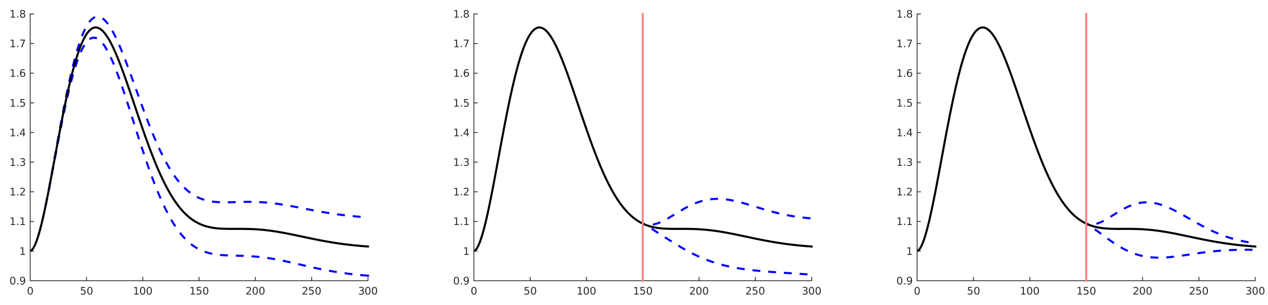

Figure 1: DMP Adaptation & Replanning

Left: DMPs can be generalised to new goal positions while preserving the general shape of the motion.

Center: If the goal position is replanned during the execution (step 150), DMPs still produce a smooth trajectory.

Right: The weights of the forcing function that defines the trajectory shape can be smoothly replanned as well.

A movement primitive is a low-dimensional parameterized representation of a single movement. Several varieties of *dynamical movement primitives* (DMPs) have been introduced [7, 8, 19]. This thesis uses the point-to-point DMP formulation of Hoffmann *et al.* [7]. These DMPs are described by a dynamical system that is affected by a forcing function and always converges to a specified goal position. They are formulated such that the speed, start position, goal position and shape of a trajectory are parameterized independently. DMPs are often used for robotic tasks in conjunction with direct policy search [1, 5, 6, 11].

A DMP consists of a canonical system

$$\tau \dot{s} = -\alpha s\,,$$

that determines the phase of the motion, where α is a constant and τ is a variable that controls the speed of the movement while preserving its shape. The trajectory is produced by the differential equation

$$\ddot{\boldsymbol{x}} = \boldsymbol{C}(\boldsymbol{g} - \boldsymbol{x}) - \boldsymbol{D}\dot{\boldsymbol{x}} - \boldsymbol{C}(\boldsymbol{g} - \boldsymbol{x}_0)s + \boldsymbol{C}\boldsymbol{f}(s)\,, \tag{3}$$

where $\boldsymbol{x}$ is the vector of joint positions, $\boldsymbol{C}$ is a spring constant matrix, $\boldsymbol{D}$ is a damping constant matrix, $\boldsymbol{g}$ is the goal position and $\boldsymbol{x}_0$ is the start position. The forcing function

$$\boldsymbol{f}(s) = \frac{\sum_i \psi_i(s) w_i}{\sum_i \psi_i(s)} s\,,$$

with Gaussian basis functions $\psi_i(s) = \exp(-h_i(s - c_i)^2)$, defines the shape of the trajectory via the weights w_i. Because $s \to 0$ for $t \to \infty$, the influence of the forcing function decreases with time and the trajectory converges to the goal position $\boldsymbol{g}$

Because the dynamical system of a DMP behaves like a spring-mass-damper system, it fulfills the crucial requirement of smooth adaptability to parameter changes. By changing the goal position $\boldsymbol{g}$, the trajectory generated by a DMP during the motion can be generalized to a variety of goal positions while retaining the overall shape of the motion. Figure 1 illustrates how replanning the goal position $\boldsymbol{g}$ or forcing function weights w_i of a DMP still produces smooth trajectories with a similar trajectory shape. These joint trajectories are typically computed online during their execution and followed by a PD-controller.

2.2.4 Online Adaptation of Dynamical Trajectories

The online adaptability of dynamical trajectory representations serves as the basis for many approaches to adapt trajectories to external states. To avoid obstacles, Hoffmann *et al.* [7] added a repelling forcing term to Eq. 3. By coupling these obstacle avoidance terms and the goal position to a vision system that tracks the obstacles, a motion could be adapted to changes in the goal position while avoiding the obstacles.

Ude *et al.* [22] demonstrated how DMPs can be adapted online by continuously updating the goal position of a motion using an active vision system. With this strategy, they realized a system that is accurate enough to pick up objects that are handed to the robot by a person.

A more advanced approach that was presented by Kober *et al.* [12] is the use of reinforcement learning to learn dynamic motor primitives that are coupled to an external state. For this, the forcing function of a DMP is extended to be dependent on the position and velocity of an external state. First, a baseline is established without perturbations, then coupling factors are learned for the perturbed case. Their results show that dynamical systems are able to effectively adapt online to state perturbations.

Similarly, Pastor *et al.* [16] demonstrated how DMPs can be used to adapt a grasping motion online. A feedback term is added to a DMP in order to match the forces that act on the hand during grasping to force trajectories of previously recorded successful grasping motions.

Khansari-Zadeh and Billard [10] introduced the concept of autonomous dynamic systems which is a more general formulation of dynamical trajectories that models the dynamical system with Gaussian mixture models and has the ability to adapt to perturbations instantly. A stability constraint is used to learn stable dynamical systems from demonstrations.

3 Replanning Policies

The idea of replanning policies is to replan parameters of dynamical trajectories to react to changes in the context. The motivation behind this is to exploit the flexibility of such trajectory representations to achieve adaptive behavior with a low number of parameters that have to be learned. State-of-the-art applications of contextual policy search often assume that the context is always completely specified ahead of the motion execution, allowing to precompute a trajectory. The present thesis weakens this assumption.

Both issues of perturbations in the state and uncertain state measurements are addressed by replanning trajectories after their execution has started. Consequently, trajectories can be corrected online. Replanning steps are introduced where the initial trajectory can be adjusted to adapt to changes in the context. The present thesis proposes a framework of replanning policies that can be expressed as sets of subpolicies that are sampled at the different (re)planning times. This is possible under weak assumptions and enables them to be used straightforwardly with episodic policy search. This thesis also presents a variant of replanning policies that have the same number of parameters as a non-replanning policy for the same task. The experiments (Section 5) demonstrate that replanning significantly improves performance over non-replanning policies on tasks with noisy state measurements or perturbations in the state. Under the assumption of a stationary context distribution, the proposed approach is able to learn replanning policies with the same number of parameters as without replanning.

Contextual learning often assumes a Gaussian policy $\pi(\boldsymbol{\theta}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\boldsymbol{c} + \mathbf{b}, \boldsymbol{\Sigma})$ that is linear in context features. Commonly, such a policy is used to precompute a trajectory. In case there are multiple planning steps $k \in 1, \ldots, \rho$ such a policy can be divided into one independent linear Gaussian subpolicy $\pi_k(\boldsymbol{\theta}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}_k\boldsymbol{c} + \mathbf{b}_k, \boldsymbol{\Sigma}_k)$ for each planning step. These subpolicies can be evaluated successively with the current context at their respective planning step. While the initial planning step must remain at the start of a trajectory, the following *replanning* steps can be spaced arbitrarily along the trajectory's time frame. This allows the adaption of trajectory parameters to a changing context.

If the Markov property holds, the subpolicies can be assumed to be independent because of causality. The Markov property can be achieved by either including the robot state in the context or by including the entire history of contexts and parameters of previous planning steps. However, the simplifying assumption is made that the subpolicies are always independent. Thus, replanning policies can be described by a single multivariate Gaussian distribution

$$\pi(\boldsymbol{\theta}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\boldsymbol{c} + \mathbf{b}, \boldsymbol{\Sigma}) \tag{4}$$

that is linear in context features with stacked vectors for the output parameters $\boldsymbol{\theta} = \left(\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T, \cdots, \boldsymbol{\theta}_\rho^T\right)^T$, context $\boldsymbol{c} = \left(\boldsymbol{c}_1^T, \boldsymbol{c}_2^T, \cdots, \boldsymbol{c}_\rho^T\right)^T$, and bias $\mathbf{b} = \left(\mathbf{b}_1^T, \mathbf{b}_2^T, \cdots, \mathbf{b}_\rho^T\right)^T$ along with block matrices for the gain and covariance

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{K}_1 & & \cdots & 0 \\ & \boldsymbol{K}_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & \boldsymbol{K}_\rho \end{pmatrix} \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_1 & & \cdots & 0 \\ & \boldsymbol{\Sigma}_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & \boldsymbol{\Sigma}_\rho \end{pmatrix}.$$

The number of parameters of the replanning policy $\pi(\boldsymbol{\theta}|\boldsymbol{c})$ grows linearly with the number of planning steps ρ. Retaining the block shape during the policy update requires further constraints on the reward model in Eq. 2. It is necessary to assume that $\boldsymbol{R}_{\theta\theta}$, $\boldsymbol{R}_{\theta c}$ and $\boldsymbol{R}_{cc}$ are block-diagonal as well. With this constraint, the policy update of MORE trivially preserves the independence of the policies by preserving the block-diagonality of $\boldsymbol{K}^{(i+1)}$ and $\boldsymbol{\Sigma}^{(i+1)}$. This is imperative to preserving the independence of the subpolicies.

The most powerful replanning policy model is a set of distinct subpolicies for every possible measurement step. Yet, increasing the number of parameters increases the search space and will slow down the learning process. In the following, two approaches for limiting the number of parameters are presented. The first approach uses the same subpolicy for all planning steps, allowing the number of replanning steps to be independent of the number of parameters. The second approach uses one subpolicy per planning step but limits the number of replanning steps.

3.1 Stationary Replanning Policy

If replanning has to occur constantly during a trajectory, the number of parameters for independent subpolicies becomes very large. In this case, a desirable assumption is $\hat{\pi}(\boldsymbol{\theta}|\boldsymbol{c}) := \pi_1(\boldsymbol{\theta}|\boldsymbol{c}) = \pi_2(\boldsymbol{\theta}|\boldsymbol{c}) = \cdots = \pi_\rho(\boldsymbol{\theta}|\boldsymbol{c})$, keeping the same number of parameters as the equivalent non-replanning policy. This means that the same policy is evaluated multiple times during a trajectory. However, this requires that the context of all planning steps is stationary. This requirement arises from the fact that a linear model is used for generalization. For example, if the context of a table tennis setup is defined as the current ball position, it violates this condition. In some cases, this limitation might be overcome by using a model of the state progression and defining the context as features of the model (e.g., a prediction of the state at a fixed time in the future). However, this is only feasible if (1) a sufficiently good model of the system state is available, and (2) unpredictable system state perturbations (external or by the robot itself) can be ruled out until after the last replanning step. This thesis denotes such a policy with equal subpolicies a *stationary replanning policy*.

An equivalent form of the reward model in Eq. 2 exists that uses a lower triangular matrix instead of a symmetric matrix. It is given by

$$\mathcal{R}(\boldsymbol{\theta},\boldsymbol{c}) \approx \begin{pmatrix}\boldsymbol{\theta}\\ \boldsymbol{c}\end{pmatrix}^T \underbrace{\begin{pmatrix}\boldsymbol{R}_{\theta\theta} & \boldsymbol{0}\\ \boldsymbol{R}_{\theta c}{}^T & \boldsymbol{R}_{cc}\end{pmatrix}}_{\text{lower triangular}} \begin{pmatrix}\boldsymbol{\theta}\\ \boldsymbol{c}\end{pmatrix} + \begin{pmatrix}\boldsymbol{\theta}\\ \boldsymbol{c}\end{pmatrix}^T \begin{pmatrix}\mathbf{r}_\theta\\ \mathbf{r}_c\end{pmatrix} + r_0\,, \tag{5}$$

where $\boldsymbol{R}_{\theta\theta}$ and $\boldsymbol{R}_{cc}$ are lower triangular matrices. The reward model for the policy update can be learned from a set of policy samples using linear ridge-regression, given by

$$\begin{aligned}\boldsymbol{w} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{Y}\,, \qquad \text{with}\\ \boldsymbol{Y} &= \left(\mathcal{R}^{(1)}, \mathcal{R}^{(2)}, ..., \mathcal{R}^{(n)}\right)^T\\ \boldsymbol{\Phi} &= \left(\phi(\boldsymbol{\theta}^{(1)},\boldsymbol{c}^{(1)}), \phi(\boldsymbol{\theta}^{(2)},\boldsymbol{c}^{(2)}), ..., \phi(\boldsymbol{\theta}^{(n)},\boldsymbol{c}^{(n)})\right)^T\,,\end{aligned}$$

where n is the number of samples, $\mathcal{R}^{(i)}$, $\boldsymbol{c}^{(i)}$ and $\boldsymbol{\theta}^{(i)}$, are the reward, context and corresponding parameters of sample i respectively and λ is a regularization constant. The regression solves for the vector $\boldsymbol{w}$ that contains all parameters of the reward model in Eq. 5. $\phi(\boldsymbol{\theta},\boldsymbol{c})$ are the features of the policy sample $(\boldsymbol{\theta},\boldsymbol{c})$.

In the case of a stationary policy, solving the reward model for all model parameters yields the parameter vector

$$\boldsymbol{w} = \left(r_0, \boldsymbol{w}_{\text{linear}}, \boldsymbol{w}_{\text{quadratic}}\right)^T$$

with the parameters for the linear part $\boldsymbol{w}_{\text{linear}} = (\hat{\mathbf{r}}_\theta{}^T, \hat{\mathbf{r}}_c{}^T)^T$ and the parameters for the quadratic part $\boldsymbol{w}_{\text{quadratic}} = (\text{vech}(\hat{\boldsymbol{R}}_{\theta\theta})^T, \text{vec}(\hat{\boldsymbol{R}}_{\theta c})^T, \text{vech}(\hat{\boldsymbol{R}}_{cc})^T)^T$, where $\text{vec}(\cdot)$ denotes the vectorization function, reordering all elements of a matrix into a column vector. Because $\boldsymbol{R}_{\theta\theta}$ and $\boldsymbol{R}_{cc}$ are triangular, the half-vectorization function vech$(\cdot)$ is also utilized. It reorders all elements of a lower triangular matrix into a column vector.

The corresponding features for the regression are defined as

$$\phi(\boldsymbol{\theta}, \boldsymbol{c}) = \left(1, \phi_{\text{linear}}, \phi_{\text{quadratic}}\right)^T$$

with linear features $\phi_{\text{linear}} = \left(\sum_{i=1}^{\rho} \boldsymbol{\theta}_i^T, \sum_{i=1}^{\rho} \boldsymbol{c}_i^T\right)^T$ and quadratic features $\phi_{\text{quadratic}} = \left(\text{vech}(\sum_{i=1}^{\rho} \boldsymbol{\theta}_i \cdot \boldsymbol{\theta}_i^T)^T, \text{vec}(\sum_{i=1}^{\rho} \boldsymbol{\theta}_i \cdot \boldsymbol{c}_i^T)^T, \text{vech}(\sum_{i=1}^{\rho} \boldsymbol{c}_i \cdot \boldsymbol{c}_i^T)^T \right)^T$.

3.2 Non-Stationary Replanning Policy

The second, more general variant of replanning policies can utilize differently distributed contexts for different planning steps, exploiting the independence of the subpolicies. This implies that even the dimensions of the context vectors $\boldsymbol{c}_1, \cdots, \boldsymbol{c}_\rho$ may differ between planning steps. Because of its capability to handle non-stationary contexts, this policy variant is denoted as *non-stationary*. An example of such a task is robot table tennis, where the ball state distribution is different at different points in time due to the progressing motion of the ball. As a result of non-stationarity, the number of parameters for the regression increases significantly. The vector of linear parameters for this case is given by $\boldsymbol{w}_{\text{linear}} = \left(\mathbf{r}_{\theta 1}^T, \cdots, \mathbf{r}_{\theta\rho}^T, \mathbf{r}_{c1}^T, \cdots, \mathbf{r}_{c\rho}^T\right)^T$. Likewise, the vector of quadratic parameters $\boldsymbol{w}_{\text{quadratic}} = \left(\text{vech}(\boldsymbol{R}_{\theta\theta 1})^T, \cdots, \text{vech}(\boldsymbol{R}_{\theta\theta\rho})^T, \text{vec}(\boldsymbol{R}_{\theta c1})^T, \cdots, \text{vec}(\boldsymbol{R}_{\theta c\rho})^T, \text{vech}(\boldsymbol{R}_{cc1})^T, \cdots, \text{vech}(\boldsymbol{R}_{cc\rho})^T \right)^T$ contains all elements of the block-diagonal matrices $\boldsymbol{R}_{\theta\theta}$, $\boldsymbol{R}_{\theta c}$ and $\boldsymbol{R}_{cc}$.

4 Implementation

This chapter explains how replanning policies were applied in the following experiments.

4.1 Replanning Control

Replanning policies rely on the online adaption of trajectories. For this reason, the underlying trajectory representation must be adaptable during its execution. All experiments in this thesis (Section 5) rely on dynamic movement primitives (DMPs), as presented by Hoffmann *et al.* [7]. The DMP trajectories are followed by a PD-controller. Experiments for the ball catching task in Section 5.4 have shown that summing the parameters of all previous subpolicies instead of replacing them can be used as a regularization for difficult problems. Algorithm 1 shows how robots can be controlled with replanning policies that adapt parameters of a DMP.

Algorithm 1 Online Trajectory Replanning

```
DMP.initialize()
for $s \leftarrow 1...s_{\max}$ do
  {(re-)plan if $s$ is a planning step:}
  if $s = s_k \in 1...s_\rho$ then
    measure $\boldsymbol{c}_k$
    if replanning by parameter replacement then
      $\boldsymbol{\theta}_k \leftarrow \pi_k(\boldsymbol{\theta}_k|\boldsymbol{c}_k)$
    else {replanning by parameter addition}
      $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} + \pi_k(\boldsymbol{\theta}'_k|\boldsymbol{c}_k)$
    end if
    DMP.parameters $\leftarrow \boldsymbol{\theta}_k$
  end if
  {advance the DMP and control the robot:}
  DMP.computeNextStep()
  $y \leftarrow$ DMP.currentJointState()
  PD-Controller.execute($y$)
end for
```

4.2 Initialization

All policies are initialized from an initial DMP which is either a hand-defined trajectory (hole reaching) or learned from a physical demonstration (table tennis and ball catching). After obtaining an initial DMP, the policies are initialized such that the mean of the initial policy produces the initial trajectory. In the case of replanning by parameter replacement, the bias vectors $\mathbf{b}_k$ of all subpolicies were initialized with their respective parameters of the initial DMP. All gains $\boldsymbol{K}_k$ are initialized with zero-matrices and all covariances $\boldsymbol{\Sigma}_k$ are set to an identity matrix that is scaled to achieve the desired level of initial exploration. In the case of replanning by parameter addition, the biases $\mathbf{b}_2, \ldots, \mathbf{b}_\rho$ of all planning steps but the first are initialized with zero vectors.

5 Experiments

This section presents experiments that were conducted to demonstrate the usefulness of replanning policies. In two variants of a hole reaching task, the performance of stationary and non-stationary policies is analyzed in settings that are specially constructed to be challenging because of perturbations (Section 5.1) and partial observability (Section 5.2). Furthermore, the more realistic tasks of simulated robot table tennis (Section 5.3) and ball catching with a real robot (Section 5.4) are evaluated.

5.1 Planar Hole Reaching with State Perturbations

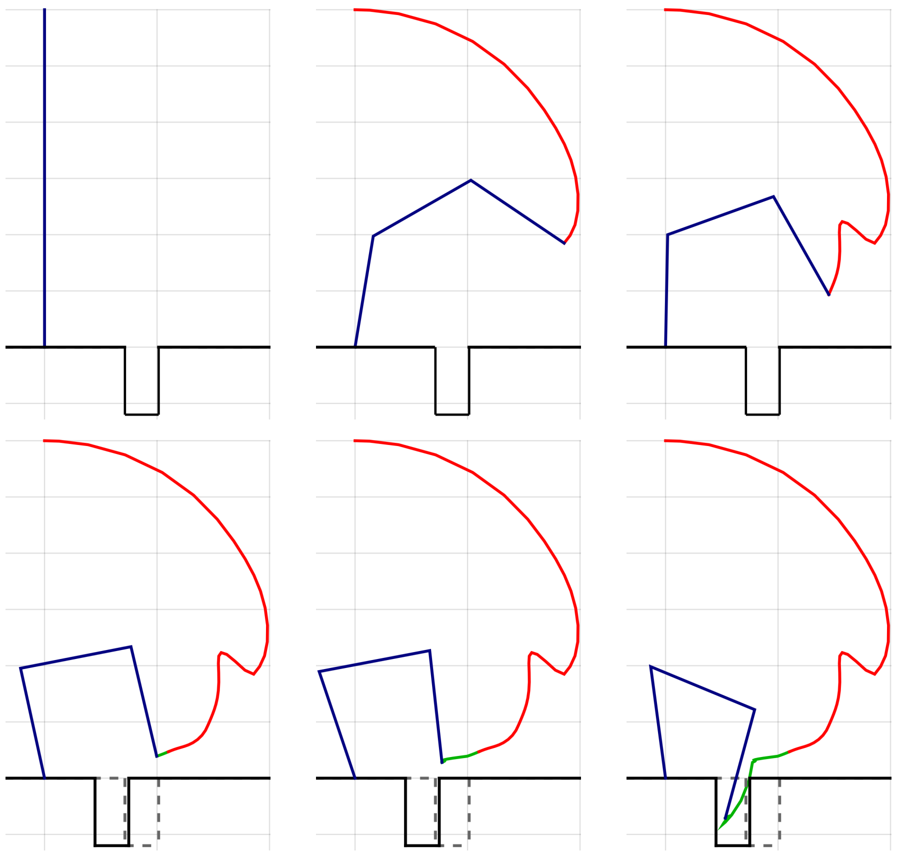

Figure 2: Hole Reaching Task

The hole reaching task setup, showing the original hole position in gray (dashed) and the new hole position in black. An end-effector trajectory that was replanned by a non-stationary policy is shown in red before the hole position change and in green after the change.

The first experiment is a planar hole reaching task with perturbations in the state. A simulated robot arm is tasked with reaching into a hole in the ground at a randomized distance in front of it. At a random time during the motion, the hole position changes a small random amount. This is an example where replanning is necessary to adapt the motion to a changing context.

As pictured in Figure 2, a 3-link robot arm was simulated. It follows the trajectory of a DMP with three basis functions per joint that defines the trajectory for each joint angle, starting from an upright position. The task is to move the end-effector to the bottom of the hole while avoiding contact with the floor and walls (black). Therefore, shortly before the end of the simulation, a reward is given proportionally to the negative squared distance of the end-effector to the center of the bottom of the hole. Meanwhile,

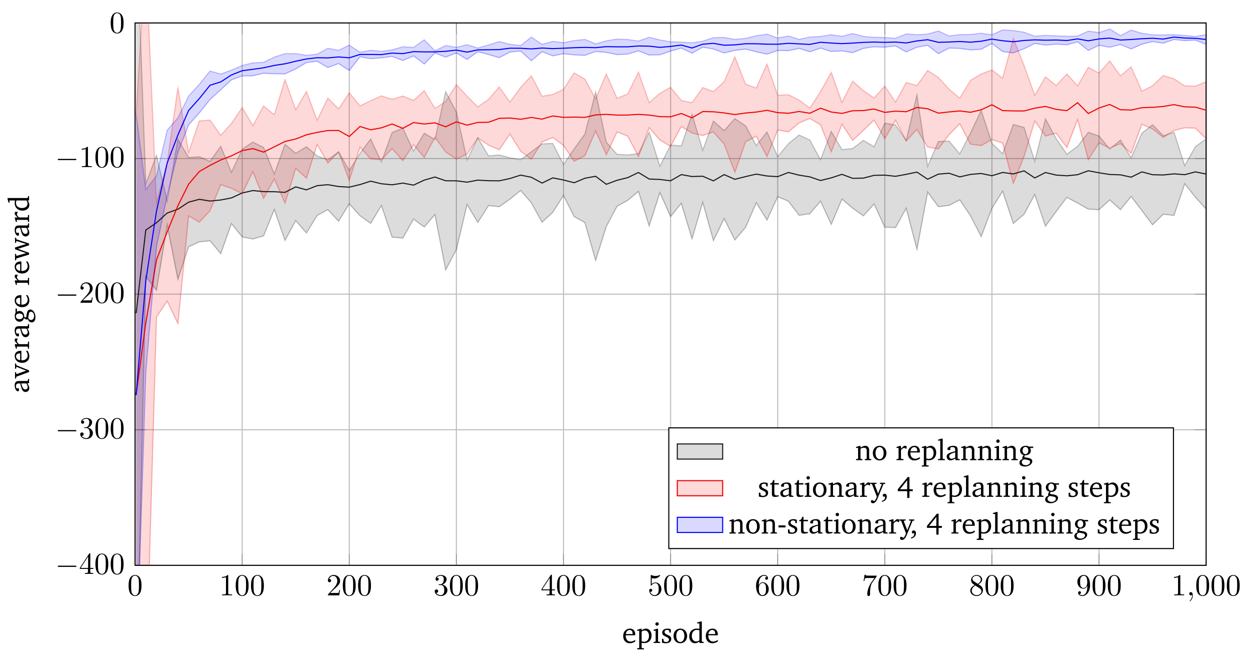

Figure 3: Hole Reaching Task with State Perturbations
Average rewards and variance per episode for non-replanning, stationary replanning and non-stationary replanning policy.

# replanning steps	1	4	9
stationary	-70.0	**-64.3**	-82.8
non-stationary	-44.4	**-12.2**	-50.0
no replanning		-111.4	

Table 1: Average Final Rewards
Average final rewards after 1000 episodes for different numbers of replanning steps for the hole reaching task with state perturbations.

every step of contact with the floor or hole walls is punished. Additionally, the problem is regularized by punishing joint accelerations. After a random time which is drawn from a uniform distribution, the hole position changes by a small random distance. In Figure 2 the old hole position is shown as a gray dashed line while the updated hole position is shown in black. The context is defined as a set of three radial basis function features of the hole position, while the output parameters of the policy are the weights of the DMP forcing function and the DMP goal position (in total 12 parameters per independent planning step).

For the experiments, 10 trials of 1000 episodes were evaluated. In each episode, 50 samples were generated while keeping the last 1000 for the policy update.

No Replanning

The black curve in Figure 3 shows the performance of the hole reaching task without replanning. Unable to adapt to a change in the hole position, it fails to reach adequate performance.

Replanning with a Stationary Policy

Although the context distribution is not exactly stationary, it is very close to stationary and the simplifying assumption of stationarity can be made. For the experiments, the replanning steps were spaced uniformly within the simulation interval. The red graph shows that replanning with a stationary policy that has an equal number of parameters (12) as the non-replanning policy improves the performance radically. It has 4 replanning steps. The average rewards are affected by a similar level of noise as the non-replanning version. This is because the summation of slightly differently distributed noisy contexts increases the total noise in the reward model estimation.

Replanning with a Non-Stationary Policy

The best performance was reached with a non-stationary replanning policy (blue). This boost in performance over the stationary policy can be explained by the more powerful model. The policies for the different replanning steps differ greatly, allowing advanced strategies like waiting until the hole change has occurred before moving down into the hole.

The effect of different numbers of replanning steps was also analyzed. As listed in Table 1, more replanning steps are not better by default. In the non-stationary case, increasing the number of planning steps results in a higher number of parameters that have to be learned, rendering the problem more difficult. Similarly, increasing the number of planning steps for the stationary case leads to increased noise for the estimation of the reward model, likewise making the problem more difficult. This demonstrates that the number of planning steps ρ is an important hyperparameter that needs to be chosen carefully.

5.2 Hole Reaching with Partial Observability

An additional experiment with a partially observable variant of the hole reaching task was performed. Instead of changing the hole position, the hole position remains fixed. However, for this experiment, the context features of the hole position are affected by additive uniform noise. The noise level is proportional to the distance between the end-effector and the entry of the hole. The reasoning behind this is to simulate a camera that is mounted to the end-effector.

Again, 10 trials of 1000 episodes were evaluated. In each episode, 50 samples were generated, keeping the last 1000 for the policy update.

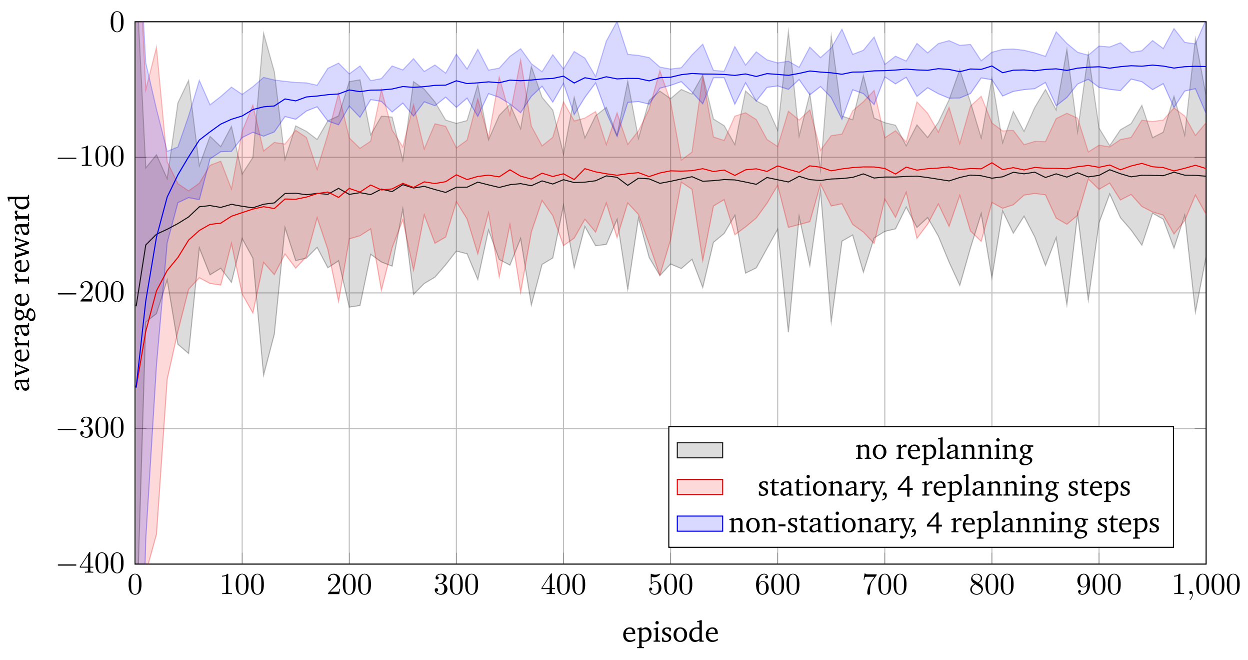

Figure 4: Hole Reaching Task with Partial Observability
Average final rewards after 1000 episodes for different numbers of replanning steps.

No Replanning

The black curve in Figure 4 shows the performance without replanning. The noise level at the beginning of the trajectory is significant. Thus, it is unable to locate the hole reliably enough.

Replanning with a Stationary Policy

The performance of a replanning policy with a stationary policy is displayed in red. Compared to the non-replanning policy, the performance cannot be improved. The reason for that is that although the mean of the context features is the same for each replanning step, their variance differs drastically. Thus, the stationarity requirement for the context is violated. The closer the end-effector is to the hole, the less noisy the measurements are. In order to utilize the more accurate measurements better, it is necessary to adapt to this change in accuracy with a different policy, requiring a non-stationary policy.

Replanning with a Non-Stationary Policy

As before, the best performance was achieved with a non-stationary replanning policy (blue). This demonstrates that simple replanning policies can be used to learn problems that suffer from partially observable states with non-trivial noise models.

5.3 Table Tennis

Another task that was evaluated for this thesis is simulated robot table tennis. A ceiling-mounted robot arm is tasked to return incoming balls to a point on the other side of the table. This is a difficult problem in the real world because of imperfect ball tracking and tight timing constraints. For these reasons, it has been used frequently in reinforcement learning research [2, 5, 6, 15, 18, 23]. Ball state measurements tend to increase in accuracy the closer it gets to the hitting moment due to filtering. However, the hitting motion has to be initiated before a perfect estimate of the ball state is available, leading to a decreased performance of trajectories that are planned ahead. These challenges make this table tennis task a good candidate for the application of replanning.

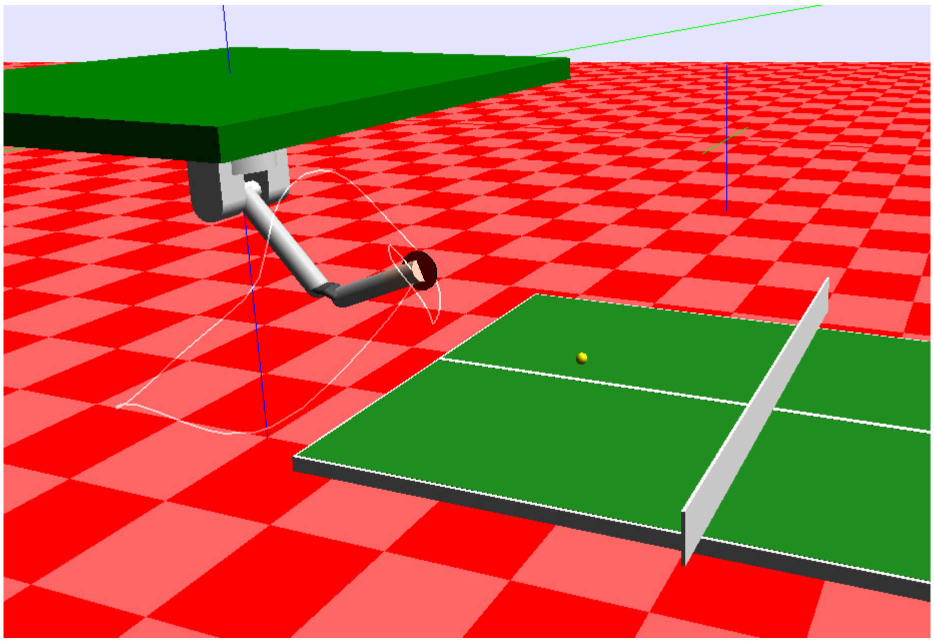

Figure 5: Table Tennis Task

This figure shows the setup of the simulated table tennis task. A table tennis racket is attached to the end-effector of a ceiling-mounted barret robot.

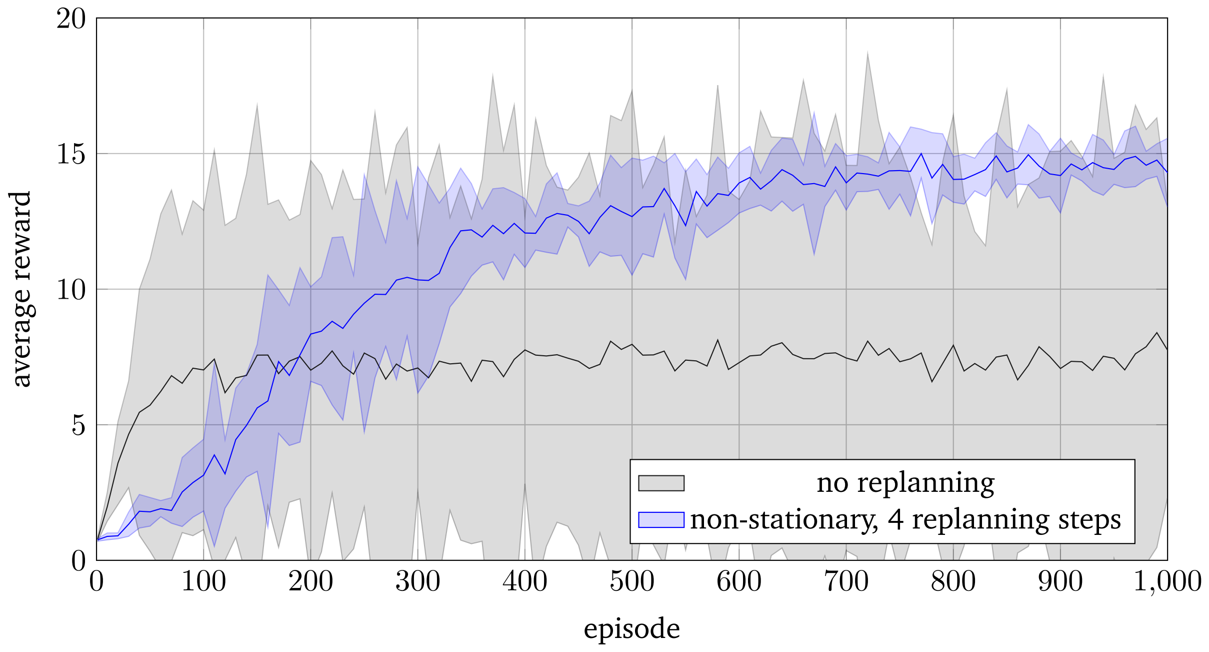

Figure 6: Table Tennis Learning Curves
Comparison of the average reward of the table tennis task and its variance per episode for non-stationary replanning policies and non-replanning policies.

The setup consists of a table tennis table and a ceiling-mounted Barret robot arm which has a table tennis paddle attached to its end-effector (see Figure 5). A simulated ball cannon shoots a table tennis ball in a fixed direction such that it bounces once on the robot's side of the table. However, the trajectories of the ball cannon are affected by noise, similar to a real ball cannon, requiring the adaption of each hitting motion to the given ball context. The robot is tasked to perform a forehand stroke that returns the ball to a specified point on the opposite side of the table. It is controlled by a PD-controller that follows a DMP trajectory which is initialized by imitation-learning. Therefore the policy has to adapt the DMP goal position to the variations in the ball trajectories. The issue of noisy ball measurements is simulated by adding Gaussian noise to the simulated ball position. This ball measurement is then filtered by an extended Kalman filter in order to maximize the performance of policy search without replanning, resulting in the black curve in Figure 6. Despite the filtering, the ball state remains noisy and can never be determined exactly. The uncertainty of the ball state measurement decreases with time, due to the extended Kalman filtering.

A trivial approach to solve this issue would be to wait until the state measurement is accurate enough before determining a trajectory [23]. Unfortunately, this strategy is infeasible because of the robot's physical limitations. Because the robot's acceleration is limited, the trajectory must be initiated before the measurement becomes accurate enough.

For the table tennis experiments, 7 trials of 1000 episodes were simulated for a non-replanning policy and a non-stationary replanning policy. Each episode, 100 samples are generated. For the policy update, the last 500 samples are used.

No Replanning

As shown in Figure 6, learning the table tennis task without replanning only reaches suboptimal performance. The remaining noise in the filtered measurements renders the generated trajectories very inaccurate. Furthermore, the achieved rewards are subject to large variations.

Replanning with a Non-Stationary Policy

In order to improve the performance, replanning with a non-stationary policy was applied to the table tennis task. Because the context is time-dependent, a stationary policy cannot be used for this task. To handle the measurement uncertainty, the trace of the Kalman filter covariance matrix was used as additional context. In this experiment, the average performance of the non-stationary policy significantly improves upon the non-replanning policy. Moreover, the variance of the average rewards is greatly reduced. This demonstrates that indeed, replanning can robustly improve the performance of a task with partial observability.

5.4 Ball Catching

To demonstrate that replanning policies can be useful in complex real-world robotic tasks, an attempt was made to learn a ball-catching task on a real robot. A Barrett WAM robot with seven joints was used as depicted in Figure 7. The robot's task is to catch a ping-pong ball with a cup that is mounted to its end-effector. The ball is thrown by a (single) human experimenter. Incoming balls are tracked by an optical motion capture system. This is a very challenging problem for several reasons.

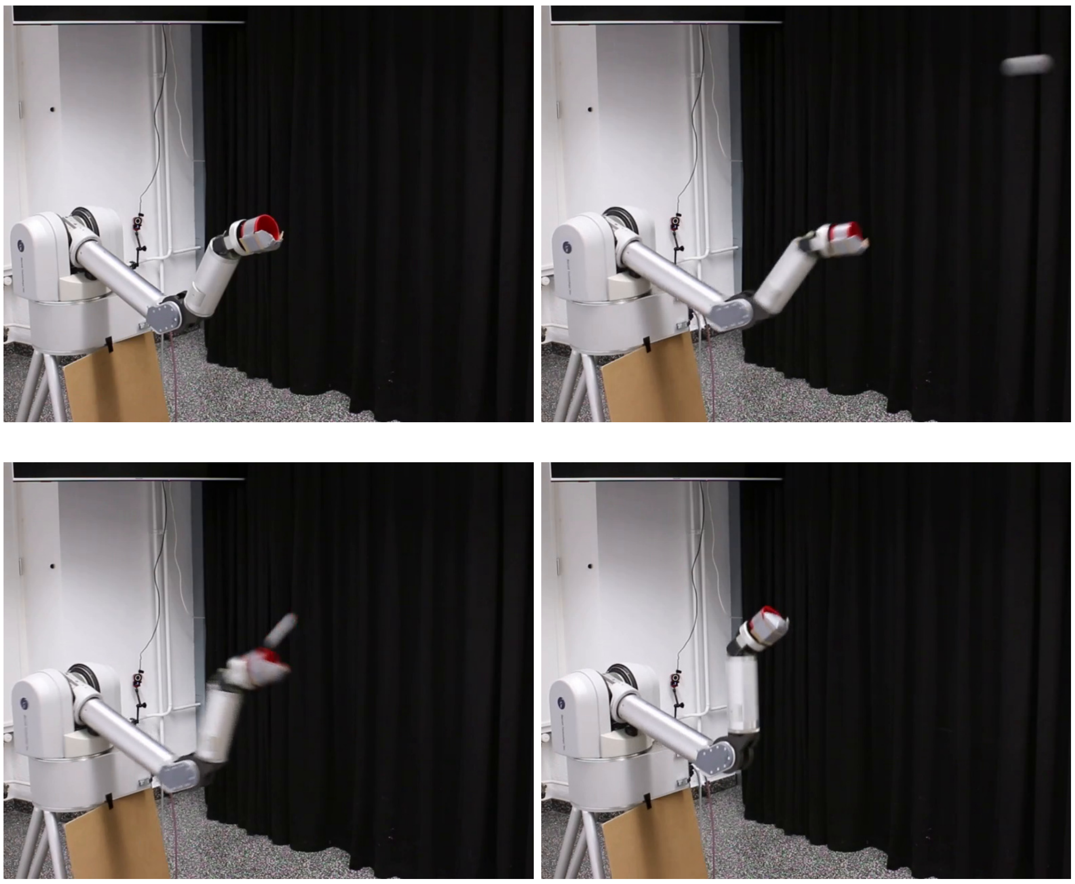

Figure 7: Ball Catching Task

This image sequence shows the setup of the ball catching task. The robot has to catch a ball (grey) in a cup on its end-effector (red). The ball is thrown by a human from ca. 3 meters in front of the robot. The time from the first registration of the incoming ball to the moment of catching is around 0.5s.

The ball is thrown by a person and therefore the robot must generalize to a large variety of ball trajectories. The ball positions at the catching moment are distributed approximately inside a circle

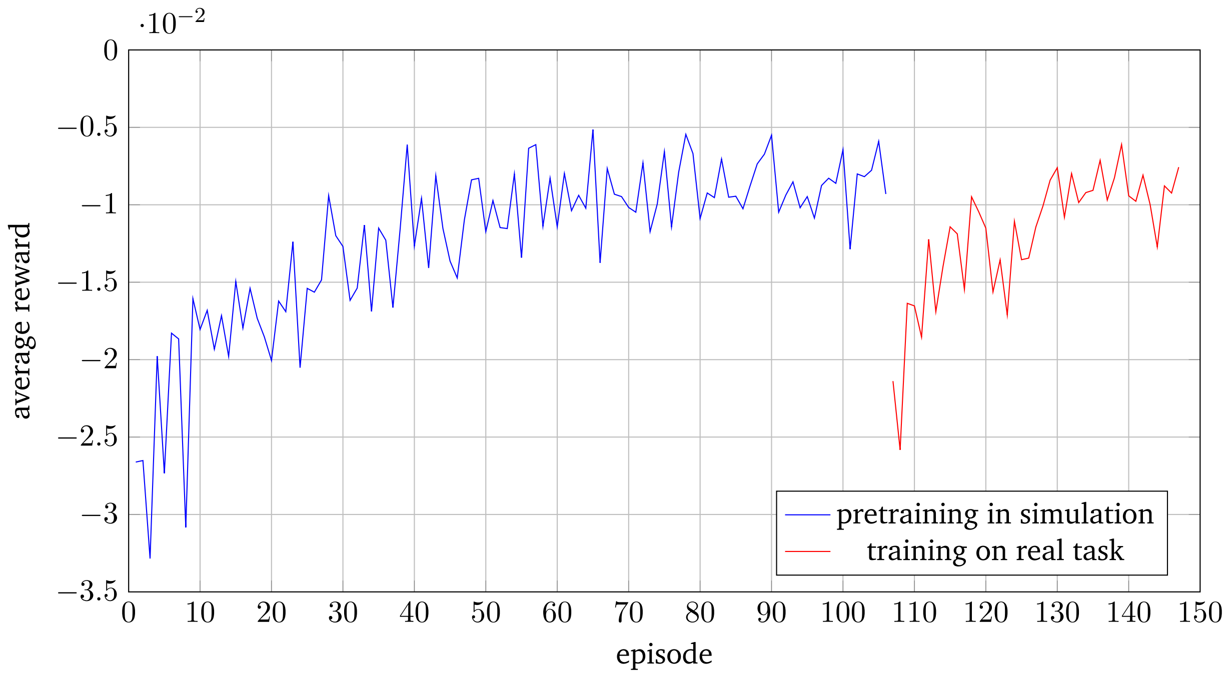

Figure 8: Ball Catching Learning Curves

The blue curve shows the average reward per episode for the policy pretraining in a simulation. The red curve shows the average rewards after transferring the policy that was learned in the simulation to the real robot.

with a diameter of 0.5m. Furthermore, human ball throw trajectory distributions may vary over time. Causes can be for instance improving throwing skills or fluctuating level of focus. The result is a high level of noise in the reward function. Another challenge is that the task is very time-critical. The robot has only approximately 0.5s from the first detection of an incoming ball to perform a catch. This is not only challenging for the robot. Another consequence of the strong time-constraint is that it is extremely difficult for a person to provide good demonstrations. Because of this, learning an initial policy in a supervised fashion is impractical.

A policy was learned that generalizes the goal position of a DMP depending on the ball position at the (re-)planning time. Thus, the number of learned parameters was $\rho \times (d_{\boldsymbol{\theta}} \times d_{\boldsymbol{c}} + d_{\boldsymbol{c}}) = 3 \times (7 \times 3 + 7) = 84$ In order to determine the remaining DMP parameters, a single catching movement was learned from a demonstration. The reward function punishes the squared minimum distance between the ball and the opening of the cup. Additionally, a small static reward is given if the ball entered the cup.

First experiments revealed that the strategy of sweeping through task space at high velocity is a very common local maximum of reward functions for such a ball catching task. However, this strategy is dangerous and may be harmful to the robot and its environment. It is enabled by the possibility to generate large jumps in the trajectory by replanning the goal position to the opposite side of the joint space. To avoid large jumps of the DMP's goal position between the replanning steps, an additive variant of replanning was devised. Instead of replacing the previous goal position every replanning step, the parameters that result from the replanning policies are added to the previous goal position. This modification effectively prevents the policy from developing sweeping behavior.

Furthermore, the joint angles of motions were limited to $\pm 10°$ around the starting position of the trajectory. This effectively restricts the solution space to joint configurations that are close to the default configuration. An added benefit of these limits is the prevention of dangerous movements in the case of occasional glitches in the ball tracking system.

To save time on the real robot system, an initial policy was learned in a simulation first. Using two replanning steps, the simulated robot was trained on a set of 100 recorded ball trajectories. This solution

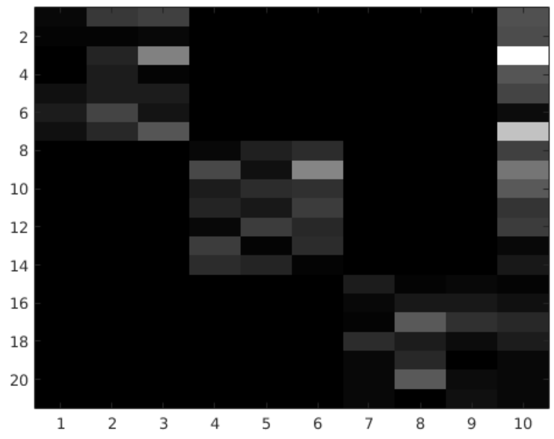

Figure 9: Ball Catching Policy
This image visualizes the gain matrix $\boldsymbol{K}$ and bias $\mathbf{b}$ of the final policy for ball catching (brighter means higher absolute value). It clearly shows that all three planning steps are relevant.

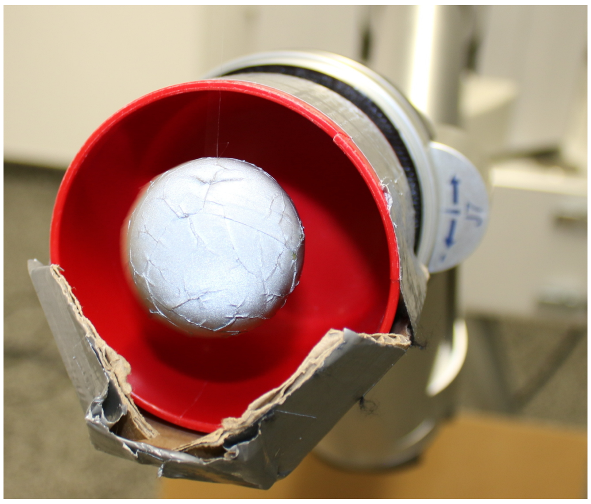

Figure 10: Ball in Relation to Cup
The opening of the cup has roughly twice the radius of the table tennis ball. For this reason, catching the ball requires high accuracy.

was then transferred to the real robot task by initializing the weights $\boldsymbol{K}$ and bias $\mathbf{b}$ of the real-world task's policy from the final policy of the simulated task. Because considerable differences exist between the two tasks, the covariance matrix $\boldsymbol{\Sigma}$ of the policy was initialized with a scaled identity matrix that forces more exploration than the converged policy from the simulated task. In simulation, 106 iterations were conducted and 41 on the real robot. Figure 8 shows the average rewards for each iteration.

The final policy was inspected in order to analyze the effect of each subpolicy. As depicted in Figure 9, each subpolicy has relevant values for gain and bias and contributes to the combined behavior. Furthermore, the subpolicies have specialized to their time steps. For example, the first subpolicy is rather insensitive to variations in the x-position of the ball, which is sensible because the trajectory is started when the ball crosses a plane perpendicular on the x-axis. On the other hand, the third subpolicy is most sensitive to the y-position, which is the horizontal axis perpendicular to the throwing direction.

The resulting policy can reliably touch the ball and catch the ball in roughly 10% of throws although this number is hard to determine exactly since it strongly depends on the throw distribution of an individual person. This low rate of catching can be explained both by the high variance in human throws and the difficulty of the task setup. The opening of the cup is so small in relation to the ball that the accuracy must be within about ± 2cm (see Figure 10). A further complication is caused by the filtering of the motion capture system. If the ball bounces off the rim of the cup, it is rarely observed close to the entry of the cup due to the filtering. If it barely passes the cup, it is often observed closer to the entry of the cup than if it bounces off the rim, creating an unwanted local maximum. Combined with the small opening, this bias on the reward function proved to be hard to combat and resulted in a policy that often touches the ball but has difficulties to catch it.

6 Future Work

While DMPs offer the possibility to be replanned smoothly, the real-world experiments revealed some shortcomings with respect to the safe adaptation of the standard DMP formulation. The possibility to develop complex trajectories by replanning the goal position in earlier steps was exploited by the policy search. While this offers a more powerful model, the low influence of the goal position on the trajectory in earlier stages can cause very exaggerated goal positions in earlier planning steps. Subsequently, the shape of a DMP is much less constrained with replanning than if it is planned ahead, resulting in potentially unsafe exploration. Furthermore, the search space of possible trajectories is greatly increased, causing the learning problem to be more difficult. In the ball catching experiment, placing a regularizing punishment on the reward function has proved to be difficult without introducing too much bias. Such behavior can be prevented by clamping the goal position. However, this implies greatly reducing the influence of a replanning policy on the earlier parts of the trajectory. The alternative method to limit the space of possible trajectories, that was applied to the ball catching task, is to place a tight joint limit on the resulting trajectory. But whenever the trajectory is cut off by the limit, the desired joint velocity for the PD-controller becomes zero, frequently resulting in considerable torque spikes. This jerky behavior can be harmful to real robots and should be avoided.

Future research could go into a DMP formulation that is safer in combination with replanning policies. This could be done by adding a forcing term in the differential equation of the DMP that attracts the output values towards a range that is deemed safe. Such a safe range could, for example, be a predefined constant range or a band around the trajectory of the same DMP without any adaptation. The goal would be to obtain a DMP formulation that can be smoothly replanned while guaranteeing that the resulting trajectory remains close to the initial trajectory.

7 Conclusion

Real-world robotic tasks often suffer from partial observability or perturbations in the environment state, rendering trajectories that are planned ahead of a motion inadequate. To adapt online to changes in the environment state or its measurements, this thesis presented a framework for learning replanning policies with very little modification to standard contextual policy search and discussed under which conditions they can be applied to different scenarios. Experiments were performed on different tasks, comparing the performance of stationary and non-stationary policies with non-replanning policies. The results of the experiments demonstrate that replanning policies outperform non-replanning policies for tasks with partially observable or changing states due to their ability to adapt online. These results indicate that replanning offers great potential to be applied to real-world robotic tasks, as they are often challenging due to partial observability and perturbations.

Bibliography

[1] Abbas Abdolmaleki, Rudolf Lioutikov, Jan Peters, Nuno Lau, Luis Pualo Reis, and Gerhard Neumann. “Model-based relative entropy stochastic search.” In: *Proceedings of Advances in Neural Information Processing Systems [NIPS]*. 2015, pp. 3537–3545.

[2] Riad Akrour, Gerhard Neumann, Hany Abdulsamad, and Abbas Abdolmaleki. “Model-free trajectory optimization for reinforcement learning.” In: *Proceedings of the International Conference on Machine Learning [ICML]*. 2016, pp. 2961–2970.

[3] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. “A survey of robot learning from demonstration.” In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[4] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, *et al.* “A survey on policy search for robotics.” In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142.

[5] Felix End, Riad Akrour, Jan Peters, and Gerhard Neumann. “Layered direct policy search for learning hierarchical skills.” In: *Proceedings of the IEEE International Conference on Robotics and Automation [ICRA]*. 2017, pp. 6442–6448.

[6] Gabriel, Alexander and Akrour, Riad and Peters, Jan and Neumann, Gerhard. “Empowered skills.” In: *Proceedings of the IEEE International Conference on Robotics and Automation [ICRA]*. 2017, pp. 6435–6441.

[7] Heiko Hoffmann, Peter Pastor, Dae-Hyung Park, and Stefan Schaal. “Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance.” In: *IEEE International Conference on Robotics and Automation [ICRA]*. IEEE. 2009, pp. 2587–2592.

[8] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. “Movement imitation with nonlinear dynamical systems in humanoid robots.” In: *Proceedings of the IEEE International Conference on Robotics and Automation [ICRA]*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[9] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. “Reinforcement learning: A survey.” In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[10] S. Mohammad Khansari-Zadeh and Aude Billard. “Learning stable nonlinear dynamical systems with Gaussian mixture models.” In: *Transactions on Robotics* 27.5 (2011), pp. 943–957.

[11] Jens Kober and Jan R Peters. “Policy search for motor primitives in robotics.” In: *Proceeding of Advances in neural information processing systems [NIPS]*. 2009, pp. 849–856.

[12] Jens Kober, Betty Mohler, and Jan Peters. “Learning perceptual coupling for motor primitives.” In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems [IROS]*. IEEE. 2008, pp. 834–839.

[13] Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, Gerhard Neumann, *et al.* “Data-efficient generalization of robot skills with contextual policy search.” In: *Proceedings of the 27th Conference on Artificial Intelligence [AAAI]*. 2013, pp. 1401–1407.

[14] Hiroyuki Miyamoto, Stefan Schaal, Francesca Gandolfo, Hiroaki Gomi, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. “A kendama learning robot based on bidirectional theory.” In: *Neural networks* 9.8 (1996), pp. 1281–1302.

[15] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. “Learning to select and generalize striking movements in robot table tennis.” In: *The International Journal of Robotics Research* 32.3 (2013), pp. 263–279.

[16] Peter Pastor, Ludovic Righetti, Mrinal Kalakrishnan, and Stefan Schaal. “Online movement adaptation based on previous sensor experiences.” In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems [IROS]*. IEEE. 2011, pp. 365–371.

[17] Jan Peters and Stefan Schaal. “Natural actor-critic.” In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190.

[18] Jan Peters, Katharina Mülling, and Yasemin Altun. “Relative Entropy Policy Search.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Atlanta. 2010, pp. 1607–1612.

[19] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics.” In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[20] Stefan Schaal, Peyman Mohajerian, and Auke Ijspeert. “Dynamics systems vs. optimal control—a unifying view.” In: *Progress in brain research* 165 (2007), pp. 425–445.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[22] Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. “Task-specific generalization of discrete and periodic dynamic movement primitives.” In: *Transactions on Robotics* 26.5 (2010), pp. 800–815.

[23] Zhikun Wang, Abdeslam Boularias, Katharina Mülling, Bernhard Schölkopf, and Jan Peters. “Anticipatory action selection for human–robot table tennis.” In: *Artificial Intelligence* 247 (2017), pp. 399–414.