Learning to Sequence Movement Primitives for Rhythmic Tasks

Bachelor-Thesis von Christoph Matthias Johannes Mayer aus Frankfurt am Main

Tag der Einreichung:

1. Gutachten:
2. Gutachten:
3. Gutachten:

TECHNISCHE UNIVERSITÄT DARMSTADT

Learning to Sequence Movement Primitives for Rhythmic Tasks

Vorgelegte Bachelor-Thesis von Christoph Matthias Johannes Mayer aus Frankfurt am Main

1. Gutachten:
2. Gutachten:
3. Gutachten:

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-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.

Darmstadt, den 1. Oktober 2015

(Christoph Matthias Johannes Mayer)

Abstract

The state of the art methods to learn trajectories in robotics are so called *Movement Primitives*. *Movement Primitives* allow the generation of versatile trajectories using only a low dimensional parameter vector. This in turn allows us to generalize motor skills.
In this thesis we are going to compare learning *Dynamic Movement Primitives* to learning *Rhythmic Dynamic Movement Primitives* in the context of rhythmic tasks where rhythm of the *Dynamic Movement Primitives* is achieved by *Sequencing*.
We are doing this exemplary for the task of bouncing balls on a racket in simulation, but the found results should be applicable for most rhythmic tasks, as long as they are decomposable into several sequences.
The found results indicate that the learning process of sequencing *Dynamic Movement Primitives* is slower than that of *Rhythmic Dynamic Movement Primitives*, but it is able to cope with harder situations in the end as it adapts for example to small errors in the execution of the trajectories.

Zusammenfassung

Um in der Robotik Trajektorien effektiv lernen zu können, ist der Stand der Technik sogenannte *Movement Primitives* zu verwenden, da diese mittels weniger Parameter die effiziente Generierung und Veralgemeinerung von Bewegungen ermöglichen.
In dieser Thesis vergleichen wir *Dynamic Movement Primitives* mit *Rhythmic Dynamic Movement Primitives* darin wie gut sie sich für das Erlenen rhythmischer Aufgaben eignen, wobei der Rhythmus für *Dynamic Movement Primitives* durch *Sequencing* erreicht wird.
Wir Untersuchen dies Beispielhaft für die Aufgabe des Bälle auf einem Tischtennisschläger wiederholt aufspringen lassens und das auch nur in der Simulation. Dennoch sollten die Ergebnisse auf die meisten rhythmischen Aufgaben übertragbar sein, sofern sie sich sinnvoll in einzelne Sequenzen unterteilen lässt.
Die gefundenen Ergebnisse deuten darauf hin, dass es wesentlich länger dauert die Hintereinanderausführung von *Dynamic Movement Primitives*„ statt der einfachen Ausführung von *Rhythmic Dynamic Movement Primitives*, zu erlernen. Der Vorteil besteht allerdings darin, dass sie am Ende mit schwierigeren Situationen, wie zum Beispiel kleinen Fehlern in den Bewegungen der Gelenke, umgehen können, da sie sich darauf anpassen können.

Contents

Figures and Tables

List of Figures

1 Introduction

Humans perform rhythmic movements on an everyday basis.
To move from one place to another we will usually repeat the same movement, stepping forward with one foot and then the other, over and over again. For larger distances we could ride a bike, where our legs again perform a rhythmic movement. Further tasks we could perform are juggling with an arbitrary number of balls or the task of bouncing a ball on a racket that we may know from training in one of the many racket sports. Another field that comes to mind is music. For example most percussion or string instruments are played by repeating a similar movement with a fixed rhythm.
It is evident that many tasks are of rhythmic nature and for that reason it should be an aim of research to enable robots to perform such tasks. In fact performing rhythmic tasks has been researched for several decades and robots are already able to perform several rhythmic tasks, for example robot drumming [1], although this task could be solely learned from demonstration. Nonetheless there are still several challenges, especially for more complex rhythmic tasks.
One challenge of current approaches for performing rhythmic tasks lies in the fact that for many of them the upcoming periods are dependent on the resulting states of previous periods. For playing an instrument this is irrelevant, as the next note is independent of how well the previous ones were played and only dependent on the song.
For other tasks it is impossible to underestimate the importance of this dependency. An example for this would be robot walking where the robot has to balance while walking. If the balance is not taken into account slight errors could add up and lead the robot to fall over.
This was a problem for example for Nakanishi et al. [2] who tried to learn the human walk using *Rhythmic Dynamic Movement Primitives*. Another task for which it is crucial to take into account that future periods depend highly on the execution of previous ones is the task of bouncing balls. Minuscule errors in the robots movement, for numerical or mechanical reasons, or errors that are induced by the environment, like wisps of wind, could result in the ball hitting the racket at the wrong point of time, where the racket has the wrong angle, which in turn results in the ball falling down in one of the later periods if these changes are not taken into account.
This thesis aims at solving the aforementioned challenge that later periods depend on previous ones by comparing two approaches. To teach the robot how to solve a rhythmic task we use Reinforcement Learning, as it is necessary that robots learn not only from demonstration, but also by trial and error in order to acquire new skills. For the first approach each episode of the task is predetermined and not changed from that point of time onwards. For the second approach each episode is broken up into its periods and we directly take the resulting state after performing one period into account in order to create an adapted movement for the next period.
To compare both approaches we test them for the task of bouncing balls, as this task is simple enough for easy comparison, while it still exhibits the problem that minuscule errors add up until an unwanted result, in this case that the ball falls down, is reached.

1.1 Related Work

One particular class of *Reinforcement Learning* (RL) is *Policy Search*. Its main difference to other RL classes is that it directly optimizes the policy without learning a value function or model first. This is an advantage for RL in robotics, as it allows a more efficient usage of samples.
For this reason this thesis uses *Relative Entropy Policy Search* (REPS). This is a *Policy Search* algorithm by Peters et al. [3] and uses the *Relative Entropy* to bound the information loss between the old and the new policy.
In this thesis we consider the original *Infinite Horizon* version [3] by Peters et al. as well as the easier *Episode-based* version [4] by Daniel et al. which is a simplification of the hierarchical Episode-based version of the same paper. This hierarchical REPS version (HIREPS) could be used for finding multiple solutions for our rhythmic task, although our focus is to solve the problem that periodic movements may need updates of the performed movement during an ongoing episode.
The *Policy Search* algorithm by Peters et al. [3] and its derivatives use the *Relative Entropy* to bound the information loss between policies.
Other *Policy Search* algorithms can be divided into several sub categories. There are for example *Expectation Maximization* algorithms like PoWER which was formalized by Kober et al. [5] or Optimal Control Algorithms like PI^2 by Theodorou et al. [6] which is used for learning with parametrized policies. We use REPS, as it combines advantages of advantages of natural policy gradient methods and EM methods.
In order to simplify the learning process *Movement Primitives* which are the state-of-the-art method for generating trajectories used. This simplifies the learning process, as they allow the representation of versatile solutions with few

parameters. *Movement Primitives* are not only used in robotics, instead research by Bizzi et al. [7] suggests, that humans too use them in order to create complex movements.
The behaviour of sequencing these primitives can also be derived from neuroscience and there have been several approaches to learn how to sequence them. For example Kallmann et al. [8] concern themselves with planning how to sequence different *Movement Primitives*. Daniel et al. [9] in comparison learn to sequence the *Movement Primitives* using the hierarchical formulation of REPS, where the algorithm learns to sequence multiple *Movement Primitives* simultaniously to the improvement of the separate *Movement Primitives*. The main difference to the approach this thesis chooses is that instead of sequencing different *Movement Primitives* we only have one *Movement Primitive* that is repeated several times, each time adapted to the current state. we learn how to sequence a *Movement Primitive* with a state dependent modification of itself.
In this paper we use *Dynamic Movement Primitives* [10] for both, rhythmic and discrete, movements. In practice DMPs are an often used type of *Movement Primitives*, although there are several other types of trajectory generators.
For example *Probabilistic Movement Primitives* which have been formalized by Paraschos et al. [11] are able to generate both rhythmic and stroke based movements, but need several demonstrations, as they incorporate the demonstrations variance. One of its advantages in comparison to DMPs is that a meaningful combination of several different trajectories is possible.
For the task of robot drumming Ijspeer et al. [12] used Rhythmic DMPs, but only learned from demonstration. Kotosaka et al. [13] used Neural Oscillators and their goal was to adapt to changing rhythms that was given by a metronome.
Different to pure rhythmic tasks considered in this thesis the task of robot drumming may not only consist of rhythmic movements, but also of discrete ones for more complex songs. Dégallier et al. [1] showed how to sequence these different parts. The difference to the sequencing approach is that in our case no mixing between different types of *Movement Primitives*occurs.
Several works have applied *Movement Primitives* to the task of walking. For example Nakanishi et al. [2] used *Rhythmic Dynamic Movement Primitives* with an additional frequency adaption, although the approach was unable to do on-line balance compensation. Theoretically our second approach that breaks the task down in multiple steps should be able to solve this problem while our first approach is similar to theirs, as both use *Rhythmic Dynamic Movement Primitives* for whole episodes. Another approach at performing the humanoid walk in robotics using *Dynamic Movement Primitives* has been proposed by Rakovic et al. [14]. This approach enables on-line changes to the *Movement Primitive* and sequences them, although they do not learn the Primitive, but instead use an adaptive regulator to react to changes in the robots balance.

2 Background Theory

In order to train the robot to perform its rhythmic task we need several known techniques which we will quickly introduce. In this thesis, we are going to learn the trajectories of the joints of the robot. To be able to easily generate good trajectories we use *Dynamic Movement Primitives*, which are parametrized trajectory generators.
As the original formulation of *Dynamic Movement Primitives* is unable to generate periodic movements we are using a variation called *Rhythmic Dynamic Movement Primitives*.
In order to provide a good initial solution to shorten the learning process we apply *Imitation Learning* to imitate the trajectory of an appropriate initial hitting movement.
To actually learn a good policy we are using two variants of a *Policy Search* algorithm.

2.1 Dynamic Movement Primitives

An often used technique for generating trajectories in robotics are *Dynamic Movement Primitives* (DMPs) [15]. *Movement Primitives* in general are compact representations of movements. DMPs are formalized as stable non-linear attractor systems and their key advantages are that they are able to represent many versatile solutions while being easy to adapt to local changes in the required trajectory. This adaptability makes them suitable for usage in changing stochastic environments. In principle DMPs are based on linear spring-damper systems. The differential equation that creates the trajectory consists of a spring part that attracts the joint to the goal position and the damper part that lessens the acceleration depending on the current speed and allows actually reaching the goal.
Ordinary spring-damper systems are only controlled by the natural frequency of the spring and the damping coefficient of the damper. To create versatile solutions a so called forcing function f_ω is added to the system. To additionally change the speed at which we approach the goal a temporal scaling variable τ is introduced and make the execution dependent on a monotonously decreasing dynamical system. If we put all this information together we get

$$\ddot{y} = \tau^2 \alpha_y(\beta_y(g-y)-\dot{y})/\tau + f_\omega(z), \tag{2.1}$$
$$\dot{z} = -\tau\alpha_z z. \tag{2.2}$$

In this equation α_y and β_y together are the spring and damper constants. The goal position g is the unique point attractor of the corresponding spring-damper system and determines the end position that is reached for $t \to \infty$. The phase variable z declines exponentially from a starting value of 1 and is scaled by τ and α_z. To use DMPs for N *Degrees of Freedom* (DoF) just as many spring damper systems have to be used. Each of those systems has its own forcing function

$$f_n(z) = \frac{\sum_{i=1}^{K} \Phi_{i,n}(z)\omega_{i,n}}{\sum_{i=1}^{K} \Phi_{i,n}(z)} z, \tag{2.3}$$
$$\Phi_{i,n}(z) = \exp\left(-\frac{1}{2\sigma_{i,n}^2}(z-c_{i,n})^2\right), \tag{2.4}$$

where $\Phi_{i,n}$ are the different basis functions and $c_{i,n}$ are their respective centres. The influence of the basis functions is increased or lessened by the $\omega_i i,n$ which are the weights we are able to learn. The parameters usually learned are $\boldsymbol{\Theta} = \{\boldsymbol{\omega}, \tau, g\}$
In order to dictate the precision of the control over the generated trajectory we are able to freely choose the dimensionality K of $\boldsymbol{\omega}$. As the forcing function is the weighted mean of the basis functions times z, $\lim_{t\to\infty} z = 0$ and $\lim_{z\to 0} \Phi_{i,n}(z) = \exp\left(\frac{-c_{i,n}}{2\sigma_{i,n}^2}\right)$ converges, the forcing function vanishes for $t \to \infty$. Hence the DMP equals a spring-damper system for $t \to \infty$, which makes the system stable.

2.2 Rhythmic Dynamic Movement Primitives

As shown in the previous section DMPs converge for $t \to \infty$. Thus, they are unable to represent rhythmic trajectories for a longer period of time.

The first problem is obviously that z goes to zero and thus, the influence of the forcing function becomes negligible. The next problem is that for the current formulation the basis functions have one center each where their influence is greatest and decreases monotonously with the distance from the center. This means that to model rhythmic movements with these basis functions an infinite number of basis functions would be required. The last issue is, that a point attractor is not suitable for rhythmic movements, as the joints shouldn't be attracted to a single point. Instead the goal is dependent on the current point of time.
To solve these problems, we use a linear function for y, periodic basis functions and a limit cycle oscillator instead of a point attractor [16]. These properties yield

$$\dot{z} = \tau^{-1}, \tag{2.5}$$
$$\Phi_i(z) = \exp\left(h\cos(z-c)+1\right), \tag{2.6}$$
$$\tau\dot{r} = \alpha_r(A-r). \tag{2.7}$$

A is the desired amplitude and r the amplitude of the oscillator. These modifications allow us to create a trajectory with a fixed period.

2.3 Imitation Learning

In robotics initialization of the robot is crucial. As each parameter extends the space of possible solutions a good initialization may reduce the learning time significantly. Furthermore finding a first meaningful solution by sampling may be hard. For the task of bouncing balls it is important to hit the ball in the first place, as not hitting the ball renders it impossible to assign a meaningful reward to different trajectories. To initialize the policy not only do we set the goal position manually, but additionally we try to imitate a given trajectory with our parameters Θ.
For DMPs this is fairly easy, as we are able to learn the weights with *Linear Ridge Regression*. To use Linear Ridge Regression we first have to calculate the forcing function for each time step.

$$f_t = \ddot{q}_t/\tau^2 - \alpha(\beta(g-q_t) - \dot{q}_t/\tau \tag{2.8}$$

The weights to follow the desired trajectory q_t,

$$\boldsymbol{\omega} = (\Psi^T\Psi + \sigma^2 I)^{-1}\Psi^T\boldsymbol{f}, \tag{2.9}$$

are calculated using the matrix Ψ which contains the basis functions of the data points.

2.4 Policy Search

To improve the initial policy that was found using *Imitation Learning* we apply a variant of *Reinforcement Learning* that is called *Policy Search*. *Policy Searchs* main advantage lies in the fact that we do not need to estimate a value function, which for most tasks is infeasible as many samples are needed, or learn a model for the robot which is only possible for moderately non-linear tasks. Instead for this technique we generate samples from the current probabilistic policy π and assess their quality in order to update the policy in the next step, which we repeat until we reach a sufficiently good policy.
The evaluation of the samples differs for the *Episode-based* and *Step-based* cases. An episode is one execution of the complete task, while the episode may be decomposable into different steps. For this reason we learn how to perform the complete execution of the task in the *Episode-based* case. The *Step-based* case differs from the *Episode-based* one in that we learn the actions to transition from one state to another in the episode.
For the Episode-based version we directly evaluate the returns, such that our resulting data-set is

$$\mathscr{D}_{\text{episode}} = \{\boldsymbol{\Theta}_i, R_i\}_{i=1\ldots N} \text{with} \tag{2.10}$$
$$R_i = \sum_{t=1}^{T} r_{t,i}, \tag{2.11}$$

where $\boldsymbol{\Theta}_i$ is the parameter vector of our i^{th} sample, R is the return that accumulates the reward, $r_{t,i}$ is the reward for the t^{th} step of the i^{th} episode and T is the total number of steps. This means that we explore in parameter space at each episode.

For the *Step-based* version we have a *Markov Decision Process*. This means that the agent is initially in a state $\boldsymbol{s} \in \mathbb{S}$ and by performing an action $\boldsymbol{a} \in \mathbb{A}$ transitions to a state $\boldsymbol{s}' \in \mathbb{S}$, $\mathbb{S}$ and $\mathbb{A}$ are the state and action space respectively. The resulting data-set is

$$\mathscr{D}_{\text{step}} = \{\boldsymbol{s}_{t,i}, \boldsymbol{a}_{t,i}, Q_{t,i}\}_{i=1...N, t=1...T} \text{with} \tag{2.12}$$

$$Q_{t,i} = \sum_{h=t}^{T} r_{h,i}, \tag{2.13}$$

where $Q_{t,i}$ is the reward to come. In this case we explore the action space at each step. The action is chosen by a stochastic policy $a_t\ \pi(\cdot|s;\Theta)$
Policy Search methods optimize the expected reward usually only locally, such that a global maximum may not be found. The policy update itself is dependent on the used algorithm. The difficulty lies in finding the right tradeoff between exploration and exploitation. Exploration is important in order to find even better solutions in future iterations of the search, but exploiting the current results is our goal to maximize the expected reward.
For *Probabilistic Policy Search* methods which are used in this thesis the policy update can be a weighted maximum likelihood estimate of the parameters. For Gaussian policies $\pi(\boldsymbol{\Theta}; \boldsymbol{\omega}) = \mathscr{N}(\boldsymbol{\Theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ this is

$$\boldsymbol{\mu} = \frac{\sum_i w_i \boldsymbol{\Theta}_i}{\sum_i w_i}, \tag{2.14}$$

$$\boldsymbol{\Sigma} = \frac{\sum_i w_i (\boldsymbol{\Theta}_i - \boldsymbol{\mu})(\boldsymbol{\Theta}_i - \boldsymbol{\mu})^T}{\sum_i w_i}, \tag{2.15}$$

$$\tag{2.16}$$

where w_i is the i^{th} weight.
For this thesis variants of the REPS algorithm are used which has an information-theoretic approach to find the right tradeoff which means that it limits the loss of information of each update.

3 Policy Search for Rhythmic Motions

In this thesis we compare two approaches for rhythmic tasks. The first approach is *Episode-based* and similar to other approaches for rhythmic tasks. The second is the novel approach. It is *Step-based* and achieves learning rhythmic movements by sequencing non-rhythmic DMPs.

3.1 The Episode-based Approach

For this approach we use the *Episode-based* formulation of *Relative Entropy Policy Search* (REPS) in order to learn the policy of the given task, where the policy controls the RDMPs. At first we retrieve an initial policy via *Imitation Learning*. As we only imitate a single trajectory we are only able to estimate the mean of the parameter vector $\boldsymbol{\Theta}$, but not its variance, so we still have to initialize the covariance matrix with reasonable entries.
The *Episode-based* formulation aims at maximizing the average return $J_\omega = \int_{\boldsymbol{\Theta}} \pi_\omega(\boldsymbol{\Theta}) R(\boldsymbol{\Theta}) d\boldsymbol{\Theta}$ of a whole episode with $\omega = \{\boldsymbol{\mu}, \Sigma\}$. $R(\boldsymbol{\Theta})$ is the return of an episode for given parameters $\boldsymbol{\Theta}$. The $\boldsymbol{\Theta}$ values are sampled from the distribution $\pi(\boldsymbol{\Theta}|\omega)$.
To bound the deviation of the new policy from the old policy, REPS uses the *Kullback-Leibler* (KL) divergence $\mathrm{KL}(p||q) = \int_{\mathbb{R}} p(x) \log \frac{p(x)}{q(x)} \mathrm{dx}$, also known as *Relative Entropy* which is a non-symmetric distance measure of probability distributions. The KL divergence leads us to the resulting bounded optimization problem:

$$\max_{\pi} \int \pi(\boldsymbol{\Theta}) R(\boldsymbol{\Theta}) d\boldsymbol{\Theta}, \tag{3.1}$$

$$s.t.\ \epsilon \geq \int \pi(\boldsymbol{\Theta}) \log \frac{\pi(\boldsymbol{\Theta})}{q(\boldsymbol{\Theta})} d\boldsymbol{\Theta}, \tag{3.2}$$

$$1 = \int \pi(\boldsymbol{\Theta}) d\boldsymbol{\Theta}. \tag{3.3}$$

Independent of the policy we are able to calculate a closed form solution efficiently using Lagrangian multipliers. The algorithm for the update of the *Episode-based* version is:
For this approach we are using RDMPs and in this context they can be called a deterministic lower-level policy that

Algorithm (Episode-based REPS):	
Input:	KL-bound ϵ, data set $\mathcal{D}_{ep} = \{\boldsymbol{s}_i, \boldsymbol{\Theta}_i, R_i\}_{i=1,\dots,N}$
Optimize dual-function:	$[\eta] = \mathrm{argmin}_{\eta'}\, g(\eta')$. s.t. $\eta \geq 0$
	$g(\eta) = \eta\epsilon + \eta \log(\sum_{i=1}^{N} \frac{1}{N} \exp(\frac{R_i}{\eta}))$
Obtain parametric policy:	$\pi_\omega(\boldsymbol{\Theta}|\boldsymbol{s})$ by weighted ML estimate
	$\omega_{new} = \mathrm{argmax}_\omega \sum_{i=1}^{N} \exp(\frac{R_i}{\eta}) \log \pi_\omega(\boldsymbol{\Theta}_i|\boldsymbol{s}_i)$

generates the action we perform in this episode. Our parameter vector $\boldsymbol{\Theta}$ controls the RDMPs and for that reason the policy that generates our $\boldsymbol{\Theta}$ samples is called an upper-level policy. It is possible to divide $\boldsymbol{\Theta}$ into 3 parts. The first part is the vector of the weights for our RDMPs. Its size is the Degrees of Freedom times the number of basis functions per DoF. With these parameters we are able to control the form of our trajectory. The second part is the τ of the RDMPs and exists in order to control the period length of the trajectories. As it makes sense for all Movement Primitives to have the same period length this is a scalar value. The third part is also a scalar value and $\in \mathbb{N}$ as it is measured in time steps. This part is the offset of the RDMP, that is needed for many rhythmic tasks, although not for all. It could be necessary if the first period has to be different from the remaining ones. For example for the task of bouncing balls it is necessary if we want to be able to learn to hit the balls up to different heights, as the height directly determines the period length.

3.2 The Step-based Approach

This approach uses the *Step-based* formulation of REPS in combination with normal DMPs to create rhythmic movements. It uses the fact that every complex movement is still composable using multiple simpler Movement Primitives. As the learned movement is rhythmic all of its periods are similar and we only have to learn one state dependent DMP.
For this we again imitate a single trajectory and have to come up with a reasonable covariance matrix. Then we use the more complex but original formulation of the REPS algorithm.
It is not *Episode-based*, but *Step-based* [15] and uses the *Infinite Horizon formulation*. This means that it maximizes the average reward of its steps. In this case the steps are not the several time steps, but the individual periods that make up the whole rhythmic movement, although the periods itself consist of multiple sub steps. The average reward is given as $J_{\pi,\mu^\pi} = \mathbb{E}[r(\boldsymbol{s},\boldsymbol{a})] = \int\int \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})r(\boldsymbol{s},\boldsymbol{a})d\boldsymbol{s}\, d\boldsymbol{a}$.
$\mu^\pi(\boldsymbol{s})$ is the stationary state distribution and represents the probability of visiting $\boldsymbol{s}$ when following π. Adding the constraint for the stationary state distribution the resulting optimization problem is given by

$$\max_{\pi,\mu^\pi} \int\int \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})r(\boldsymbol{s},\boldsymbol{a})d\boldsymbol{s}\, d\boldsymbol{u}, \tag{3.4}$$

$$s.t.\ \epsilon \geq \sum_{\boldsymbol{s},\boldsymbol{a}} \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \log \frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})} d\boldsymbol{s}\, d\boldsymbol{a}, \tag{3.5}$$

$$\int \mu^\pi(\boldsymbol{s}')\phi(\boldsymbol{s}')d\boldsymbol{s}' = \int\int\int \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})\phi(\boldsymbol{s}')\mathrm{d}\boldsymbol{s}, \mathrm{d}\boldsymbol{a}\mathrm{d}\boldsymbol{s}', \tag{3.6}$$

$$1 = \int\int \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathrm{d}\boldsymbol{s}, \mathrm{d}\boldsymbol{a}, \tag{3.7}$$

where $\phi(\boldsymbol{s})$ is the state dependent feature which is not needed by the method which uses the Episode-based formulation, but could be added to it too.
As before the constraint optimization problem can be solved with Lagrangian multipliers. The final algorithms is given by

```
Algorithm (Step-based REPS):
Input:                          KL-bound $\epsilon$, data set $\mathcal{D} = \{\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i\}_{i=1,...,N}$
for $i = 1, ..., N$ do
    State-action visits:        $n(\boldsymbol{s}_i, \boldsymbol{a}_i) = \sum_j I_{ij}$.
    Summed reward:              $\tilde{r}(\boldsymbol{s}_i, \boldsymbol{a}_i) = \sum_j I_{ij} r_i$.
    Summed features:            $\delta\tilde{\phi}(\boldsymbol{s}_i, \boldsymbol{a}_i) = \sum_j I_{ij}(\phi(\boldsymbol{s}'_i) - \phi(\boldsymbol{s}_i))$.
end for ($I_{ij}$ is 1 if $\boldsymbol{s}_i = \boldsymbol{s}_j$ and $\boldsymbol{a}_i = \boldsymbol{a}_j$, 0 elsewhere)
Compute sample bellman error    $\delta_{\nu,i} = \frac{\tilde{r}(\boldsymbol{s}_i,\boldsymbol{a}_i) + \nu^T \delta\tilde{\phi}(\boldsymbol{s}_i,\boldsymbol{a}_i)}{n(\boldsymbol{s}_i,\boldsymbol{a}_i)}$.
Optimize dual-function:         $[\eta, \nu] = \mathrm{argmin}_{\eta',\nu'}\, g(\eta', \nu')$. s.t. $\eta \geq 0$
                                $g(\eta, \nu) = \eta\epsilon + \eta \log\left(\frac{1}{N}\sum_{i=1}^N \exp\left(\delta_{\nu,i}\right)\right)$.
Obtain parametric policy:       $\pi_\omega(\boldsymbol{a}|\boldsymbol{s})$ by weighted ML estimate
                                $\boldsymbol{\Theta}_{k+1} = \mathrm{argmax}_{\boldsymbol{\Theta}} \sum_{i=1}^N \exp(\frac{\delta_{\nu,i}}{\eta}) \log \pi_{\boldsymbol{\Theta}}(\boldsymbol{a}_i|\boldsymbol{s}_i)$.
```

The parameters we have to learn for this approach are just the weight vector with size DoF times number of basis functions and the again scalar value τ that we use in order to speed up or slow down the execution of the movement primitives. What makes this harder than the learning process of the *Episode-based* approach is that we have to take the state $\boldsymbol{s}$ into account.

4 Experiments

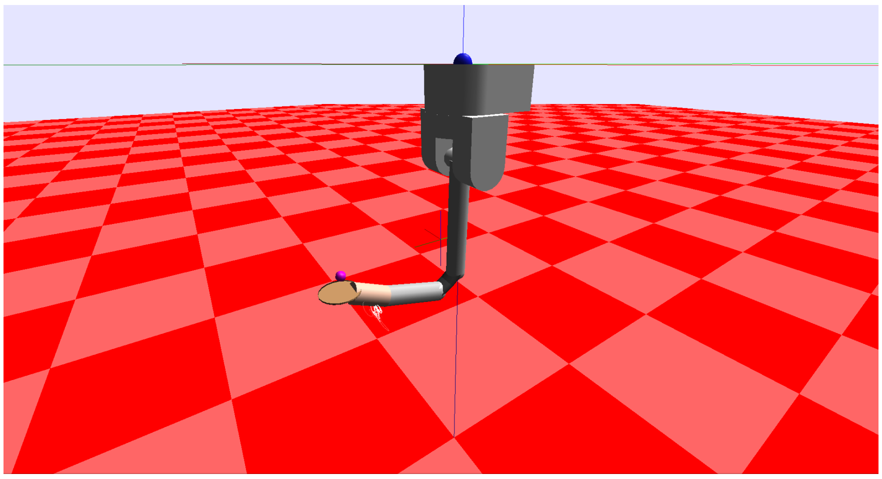
Figure 4.1: The 7 DoF robotic arm performing the task of bouncing balls.

In this chapter we are going to compare the *Episode-based* and the *Step-based* method for the task of bouncing balls on a racket. In this problem statement we simulate a 7 DoF robotic arm with a ping pong racket as an end-effector. For this we use the *Simulation Laboratory* (SL) software package [17]. SL allows the creation of complex rigid-body dynamics simulations while staying close to the actual robot.

The ball is always dropped from the initial position $\begin{pmatrix} 0 & -0.545 & -0.7 \end{pmatrix}^T$ which is chosen in such a way that the ball is exactly 0.2 over the center of the end-effector and the initial joint-angles are 0 except the elbow joint, which is bent in a right angle and the racket is rotated in such a way that its hitting plane is parallel to the x-y plane.

The task is to hit the ball in such a way that its peak is always at the same point and the ball should never fall down. Figure 4.1 shows the simulation at one point of time to make it clear what this task looks like.

The learning process in general is dependent on many parameters. For some of them, like the total amount of samples, it is obvious that higher values lead to a higher return on average in the end and for others there may exist a perfect value independent of the remaining parameters. Most of them probably belong to the third type, which means that their optimal value depends on the remaining parameters.

To optimize all parameters simultaneously is nearly impossible, as the optimization process would depend on random variables and it is likely that the function $f : \boldsymbol{\Theta} \Rightarrow \mathbb{J}_\omega$ that maps the parameter space to the corresponding average return contains local maxima.

This section nonetheless compares different values for some of the parameters in order to enhance the learning process for both methods to compare them afterwards.

For each value a parameter takes in one of the experiments 5 trials have been executed and the figures contain the trials mean and variance.

As numerical errors and non-deterministic behaviour in SL showed that it was impossible to learn the task of bouncing balls for all 3 dimensions for the *Episode-based* version we restrict ourselves to only the movement in z-direction and try to optimize the reached height.

4.1 Experiments of the Episode-Based Approach

In this section we evaluate several parameters of the *Episode-based* version which uses RDMPs to create complete Episodes at once and each episode consists of 5000 time steps in total.

To enable a better comparison of the effects of changing the different parameters, all parameters are fixed to the same value in all experiments except the parameter that is currently examined. The relevant default values of the parameters are: KL boundary $\epsilon = 1$, 10 iterations, 50 samples per iteration, discount factor $\gamma = 1$ and we use only the current iterations samples, old ones are discarded.

The return for each episode is calculated after the execution of the trajectory has been finished and it rewards the number or peaks while it punishes the distance of the ball at each peak from the initial position, i.e

$$r_i = \left\| \begin{pmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{pmatrix} \boldsymbol{q}_i - \begin{pmatrix} 0 \\ -0.545 \\ -0.7 \end{pmatrix} \right\|_1 , \tag{4.1}$$

$$R = \sum_i \left(k_4 + \gamma^i r \right), \tag{4.2}$$

where the r_i is the reward for the i^{th} peak, q_i is its position and γ is the discount factor. k_1, k_2 and k_3 control the punishment for each axis and k_4 determines how much finding a peak is rewarded. For this task the k_i are chosen in such a way that the maximum reward possible is 5.

4.1.1 Changing ϵ

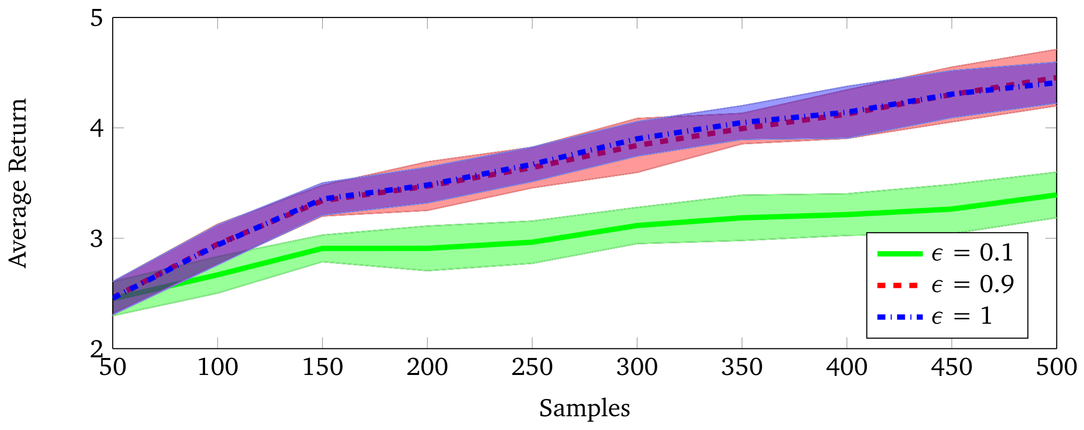

Figure 4.2: This figure shows the comparison of different boundaries ϵ for the KL divergence. Although all possible values from 0.1 to 1 with a step size of 0.1 have been tested only the best result and the two boundaries are shown.

The main advantage of REPS in comparison to other *Policy Search* algorithms is that the change from one policy to the next is bounded by the KL-divergence, such that a good trade-off between exploration and exploitation can be found.
For this reason the ϵ that bounds the KL-divergence is an obvious target for optimization. Usual values for ϵ are in the range $[0.1; 1]$, so this was the range for sampling.
In figure 4.2 we see the learning process of for $\epsilon = 0.9$ for which the best solution was found. The other two graphs show boundaries 0.1 and 1 respectively.

4.1.2 Changing Discount Factor

Many reward functions attach greater importance to earlier steps, as the subsequent steps depend on the previous ones. For this reason we evaluate how a discount factor γ that reduces the influence of the n^{th} peak by γ^n improves the learning process.
As we lessen the reward per peak it is clear that the returns for the different values is incomparable. For this reason the plot depicted in figure 4.3 shows the returns of the version with discount factor 1 independent of the actual discount factor of this trial.
Although we subtracted the influence of the discount factor for this evaluation the result shows that having no discount worked best. A possible explanation would be that a nice peak at a later point indicates that the previous ones were good too, but one that our reward function thinks of as good in the beginning may still lead to the catastrophic addition of errors.

4.1.3 Reusing Samples

In REPS each sample is generally only used once. After the sample has been used it is thrown away and its information is lost. As sampling is the most time consuming part of Policy Search for most applications, it could be a good idea to keep

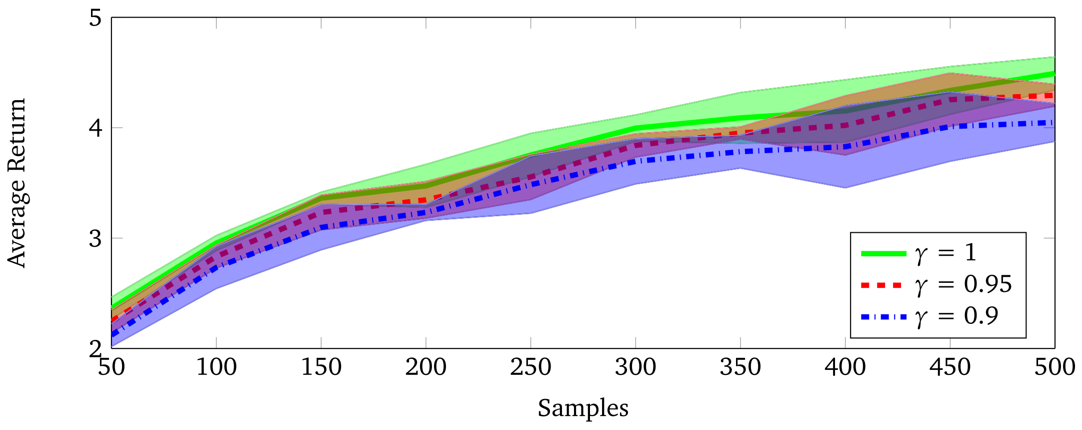

Figure 4.3: This figure shows the comparison of different discount factors γ. The average return in this case is calculated with a discount factor of 1 for all values of γ.

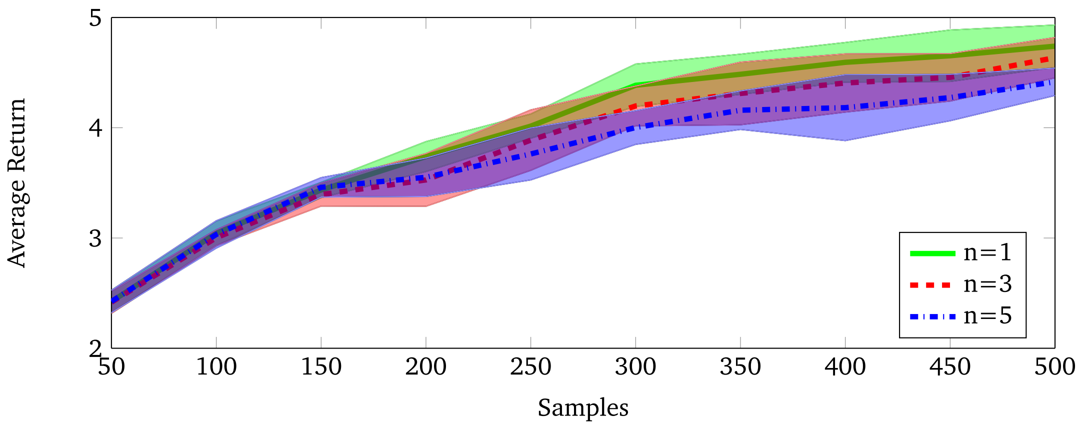

Figure 4.4: This figure shows the comparison of the amount of reused samples. In this context a value of 5 means that additionally to the current iterations samples the last 4 iterations samples are also used.

the information. Therefore we reuse old samples directly in REPS, but as we get samples in regions where our policy has a low density and using more samples is still more complex we do not want to reuse too many samples.
In figure 4.4 we see that this has a purely detrimental effect on this task, so we should not reuse old samples.

4.1.4 Changing Samples per Iteration

As mentioned for reusing samples the process of sampling is the most time consuming part of the REPS algorithm. For that reason we want to use our samples as efficient as possible.
For this experiment we assume that for the total cost of learning, independent of its unit, everything but sampling is negligible. This means that if we leave the total number of samples constant and only change the number of samples per iteration we have another meaningful parameter to observe.
For this experiment we are not only going to observe the change for our default ϵ value, but also for $\epsilon = 0.5$. We do this as ϵ could be highly correlated to the samples per iteration, as more samples per iteration means that we need a greater change in comparison in order to end up with the same policy after the learning process finished.
The results that can be seen in figure 4.5 and 4.6 show that although the graphs for $\epsilon = 1$ and $\epsilon = 0.5$ are different the curves are still very similar, such that the suspected high correlation is quite unlikely.

4.2 Experiments of the Step-Based Approach

In this section we are going to evaluate the several parameters of the second method. In this method each sample corresponds to a single hitting movement and an episode ends after 25 hits or if the ball falls down. Rewards are assigned to each sample and the learned policy depends on the parameters.

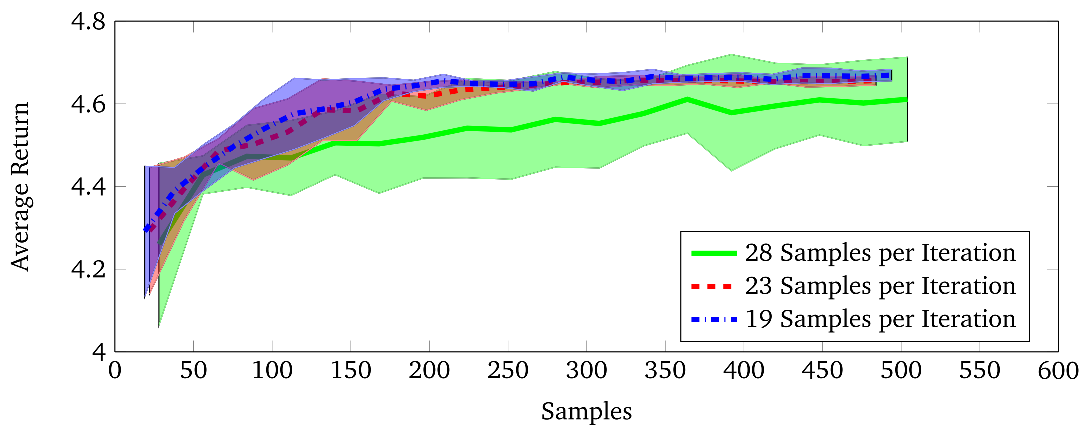

Figure 4.5: This figure shows the comparison of the samples per iteration while the the total amount of samples is fixed with $\epsilon = 1$.

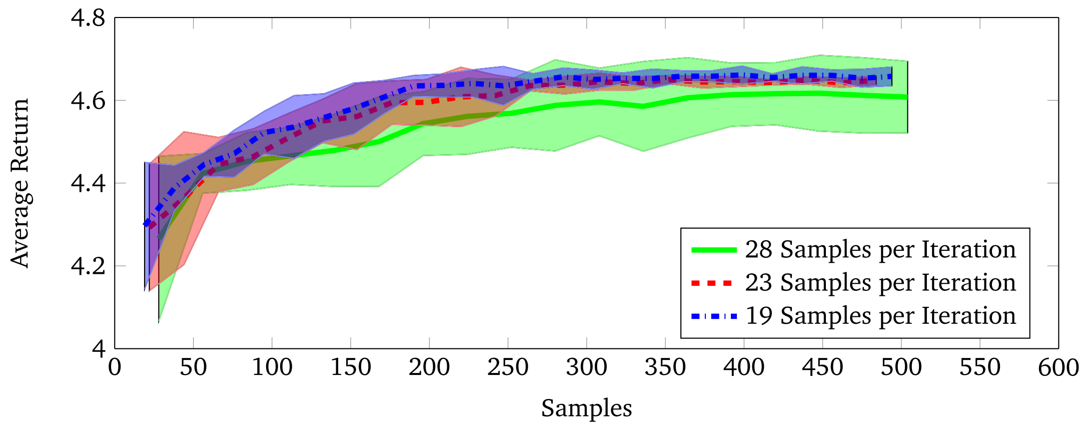

Figure 4.6: This figure shows the comparison of the samples per iteration while the the total amount of samples is fixed with $\epsilon = 0.5$

As the initially assigned parameters were too bad this sections parameters were updated after each experiment to the parameters with the best result.
To avoid repetitions the explanations as to why a certain parameter is important are only in the first methods experiment section.
The return is calculated after each episode and it is the sum of the rewards that have been calculated after each step. The reward for each step just takes in account the final position of the racket and the joints. We reward again if in the end the ball did not fall down and punish the difference of the joint angles to the initial joint angles, the difference of the

ball position to its initial ball position and the difference of the ball position, at the pre-calculated point of time when it is going to have the same height as at its previous hitting position, to the center of the racket at its initial position. I.e.

$$j = \left\| q_j - \left(\; 0, 0, 0, \tfrac{\pi}{2}, 0, 0, -\tfrac{\pi}{2} \; \right)^T \right\|, \tag{4.3}$$

$$g = 0.81, \tag{4.4}$$

$$t = \sqrt{2\frac{q_z - q_{\text{prev,z}}}{g}}, \tag{4.5}$$

$$p = \left\| \left(\begin{pmatrix} q_x \\ q_y \end{pmatrix} + \begin{pmatrix} \dot{q}_x \\ \dot{q}_y \end{pmatrix} \right) - \begin{pmatrix} 0 \\ -0.545 \end{pmatrix} \right\|, \tag{4.6}$$

$$e = \left\| q - \begin{pmatrix} 0 \\ -0.545 \\ -0.7 \end{pmatrix} \right\|, \tag{4.7}$$

$$r = -k_1 j - k_2 p - k_3 e + u k_4, \tag{4.8}$$

where q and q_j are the final ball and racket positions respectively, $q_{\text{prev,z}}$ is this steps initial height, r is the reward, j is the component to punish wrong joint positions, p punishes the distance to the optimal hitting plane and e punishes differences in the end positions. u rewards the hitting movement if the ball is still up in the end. The different $k_i, i \in \{1, ..., 4\}$ are the meta parameters to control the components influence. The distances are Euclidean. In general the maximum reward is 250, as each hit has a maximum reward of 10.

4.2.1 Changing ϵ

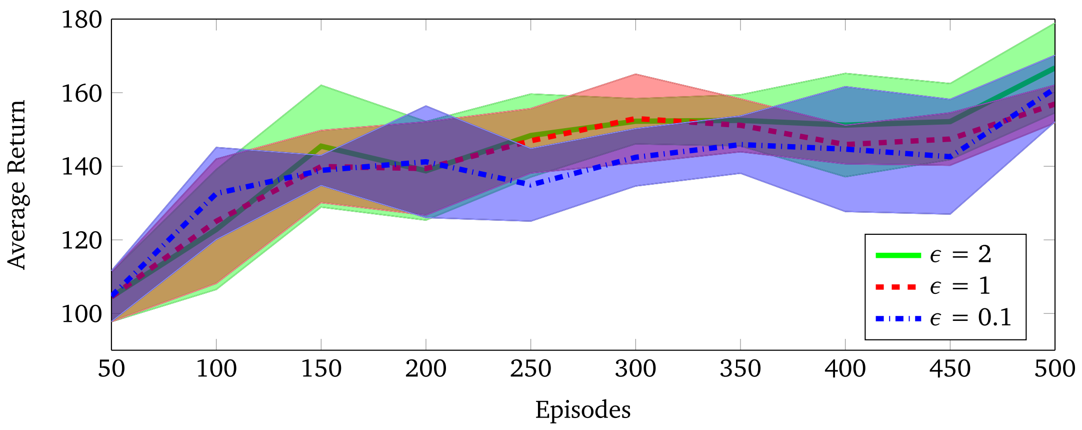

Figure 4.7: Comparison of different boundaries ϵ for the KL divergence. The fixed parameters are 100 iterations, 50 episodes per iteration and the reuse of the previous 9 iterations.

This experiments fixed parameters are: we use the current and the previous 9 iterations samples, we have 10 iterations and 50 episodes per iteration. Figure 4.7 shows that even $\epsilon = 0.01$, although it is far less than the first methods optimal ϵ value, performs nearly as good as $\epsilon = 0.1$ and $\epsilon = 1$ which had no reason the following experiments use $\epsilon = 0.1$.

4.2.2 Reusing Samples

Although the first method indicated that reusing samples is detrimental to the learning process it was unclear if this was a peculiar property of the first method.
As we can see in figure 4.8 it is not peculiar to the method, but to the task, as using only the iterations own samples showed the best results.

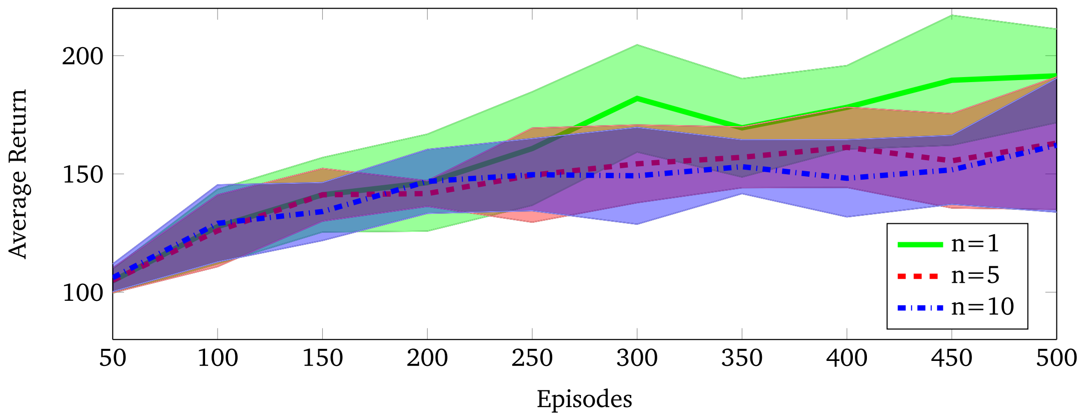

Figure 4.8: Comparison of different amounts of reused samples. The fixed parameters are 10 iterations, 50 episodes per iteration and $\epsilon = 0.1$

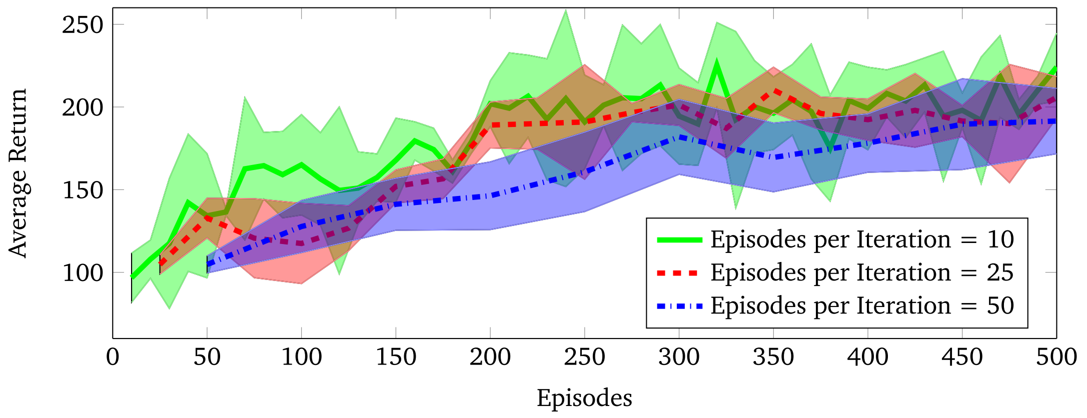

Figure 4.9: Comparison of different episodes per iteration while keeping the total number of episodes constant with $\epsilon = 0.1$.

4.2.3 Changing Episodes per Iteration

For the *Episode-based* method each sample corresponded to an episode and thus each REPS iteration had a fixed number of samples. In the second approach this is not the case. Now up to 25 samples are contained in a single episode. For this reason Samples per Iteration it is no meaningful measure, but Episodes per Iteration is. Figures 4.9 and 4.10 show that we were finally able to find a solution for $\epsilon = 1$ with 50 iterations and only 10 episodes per iteration.

4.3 Direct Comparison

As for both methods different reward functions were used it is hard to compare both methods just from their rewards. For this reason this section compares the generated ball trajectories for the case where the x and y dimension are fixed and for the case where only the x dimension is still fixed.

4.3.1 Single Dimension

As we can see from the figures 4.11 and 4.12 the *Episode-based* approach is clearly better for the case where all dimensions except the height are fixed. For the *Episode-based* version it makes sense to find such a solution after only a few samples. The *Step-based* approach in comparison is disappointing. After the same amount of samples the solution is much worse as the no uniform height is achieved and the algorithm is still far from converging.

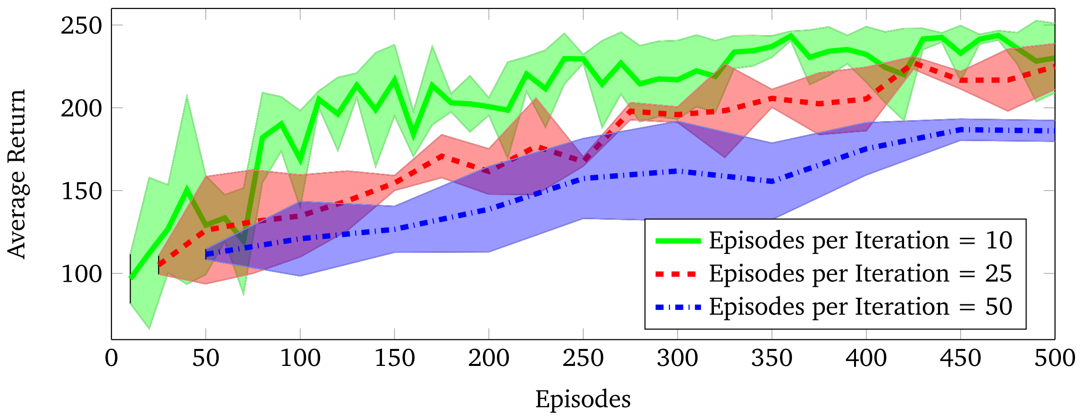

Figure 4.10: Comparison of different episodes per iteration while keeping the total number of episodes constant with $\epsilon = 1$.

4.3.2 Multiple Dimensions

Now that we were able to learn the one dimensional case with the *Step-based* method, in a way that the ball is at least hit every time we can observe how it performs for more dimensions.
In order to make the task not to complex we still hold the x-axis fixed and only enable the ball to change on the y-axis. Learning the movement for the y-axis is hard, as slight movements in the elbow lead to great differences in the balls trajectory. For this method the punishment for the difference of the next hitting position from the perfect position was increased and in order to still obtain positive rewards the reward for hitting the ball was increased to 30. This means that the maximum possible return is 750. We also sampled more episodes, 40 per iteration, as to little episodes led to numerical instabilities.
As this reward is incomparable to all other rewards it is left out. Nonetheless in the 8th iteration of one trial we could observe the first trajectory with 5 hits 4.14 and after 10 iterations 4 successful hits seem to be the average. In this case the *Step-based* method is better, as the *Episode-based* method that we can see in figure 4.13 needed 50 iterations to converge and is still only able to achieve 5 peaks at its best.

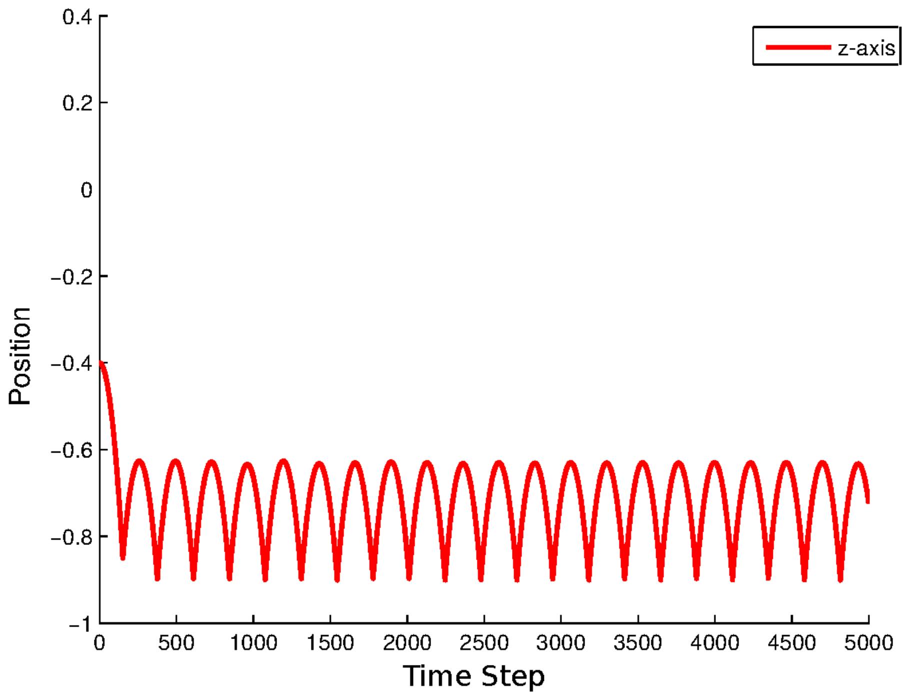

Figure 4.11: This figure shows the the converged solution of the Episode-based approach for the one-dimensional case.

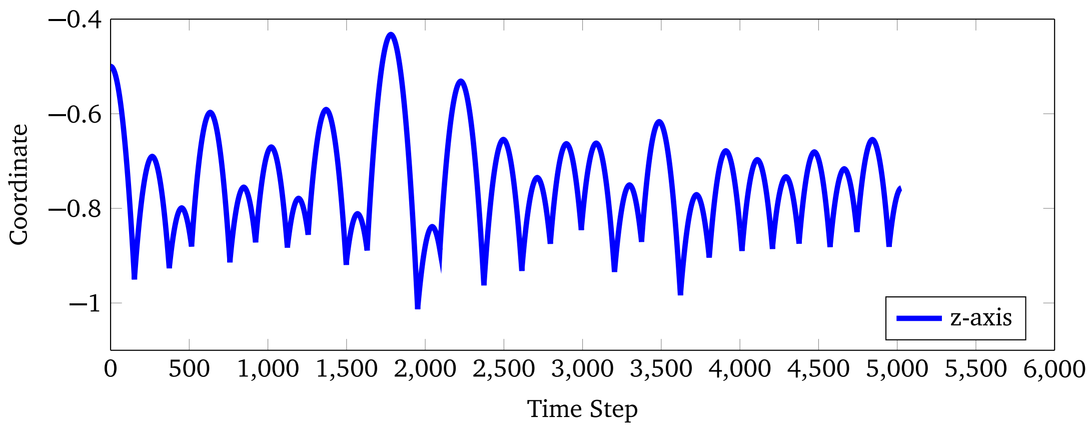

Figure 4.12: This figure shows the the solution found by the Step-based approach for the one-dimensional case before the algorithm converges, as this takes too much time.

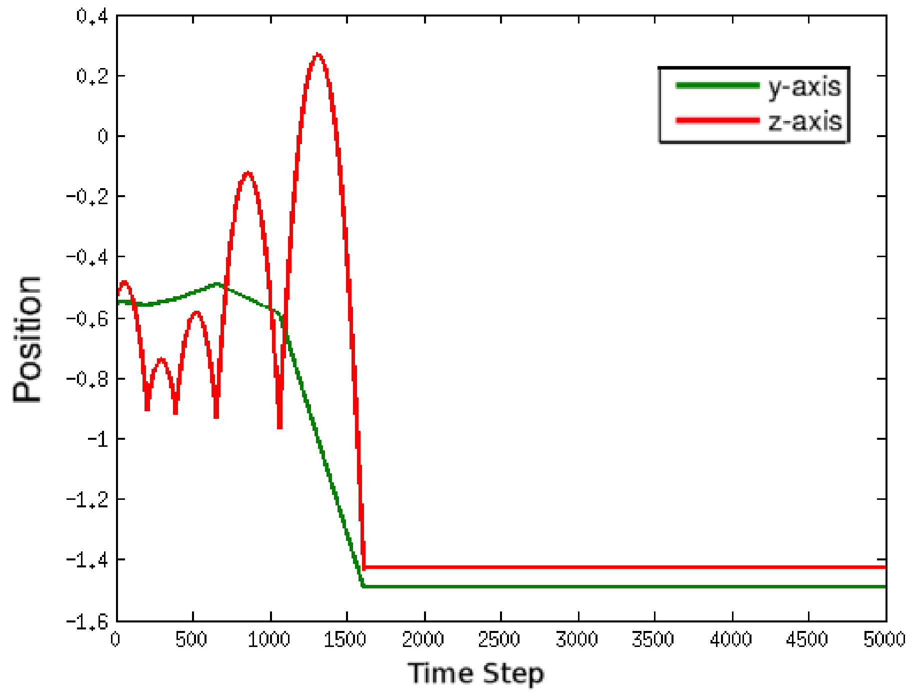

Figure 4.13: This figure shows the the converged solution of the Episode-based approach for the one-dimensional case.

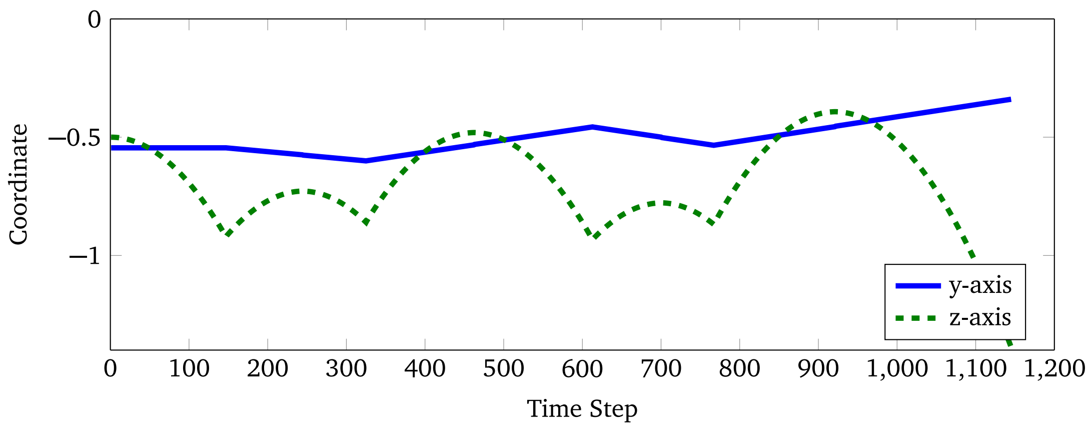

Figure 4.14: Sample ball trajectory after 8 iterations with fixed x-dimension for the Step-based version.

5 Conclusion and Further Work

For the one-dimensional bounceball taks the *Episode-based* method that was discussed in this thesis showed better results after only a few iterations, but it converged too fast to a suboptimal, but still very good policy. For the two-dimensional case it was unable to find a working solution at all.
Although the *Step-based* method needed many more samples to execute the task of bouncing balls for one dimension and the found solution was considerably worse it seems to be the more promising approach. This is because it performed better for the two-dimensional case which exhibits the real challenge this thesis tried to address.
Obvious shortcomings of the *Episode-based* method are that it is inflexible. Numerical errors that arise in the simulation as well as slight errors that could occur in reality due to wisps of wind, the unknown spin of the ball or just imprecise robot movements lead to unstable solutions of this task and after only a few hits the ball falls down even with a converged policy.
State dependent steps, like those in the *Step-based* method, seem to be the best way to cope with these problems. For this reason it would be interesting to advance the second approach further. It could be possible to reduce the number of samples that is required to find an adequate solution and it has yet to be shown where the limits for the second approach lie.
To be able to adapt better to the different states it could be required to use different features, which should be tried too.
Those ideas are all realizable in simulation, but the final goal should be to implement the approach on a real robot. This is especially hard not only as robots are fragile and sampling has to be done in real time. The real problem is that the second method stopped the simulation at each peak to calculate the next trajectory. In reality this is obviously impossible and it has yet to be seen if the calculation of the next step can be executed fast enough, such that the robot is still able to use the given trajectory for hitting the ball.
Possible solutions to this problem would be to just hit the ball higher or to predict the peak position while the ball is still ascending.
Another problem is to find out a close enough approximation of the position and velocity of the ball at its peak, as this information defines the state.
As you can see there are a lot of fascinating challenges and I hope that in the near future I will be able to meet at least a few of these challenges.

Bibliography

[1] S. Degallier, C. P. Santos, L. Righetti, and A. Ijspeert, “Movement generation using dynamical systems: a humanoid robot performing a drumming task,” in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pp. 512–517, IEEE, 2006.

[2] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, “Learning from demonstration and adaptation of biped locomotion,” *Robotics and Autonomous Systems*, vol. 47, no. 2, pp. 79–91, 2004.

[3] J. Peters, K. Mülling, and Y. Altun, “Relative entropy policy search.,” in *AAAI*, 2010.

[4] C. Daniel, G. Neumann, and J. R. Peters, “Hierarchical relative entropy policy search,” in *International Conference on Artificial Intelligence and Statistics*, pp. 273–281, 2012.

[5] J. Kober and J. R. Peters, “Policy search for motor primitives in robotics,” in *Advances in neural information processing systems*, pp. 849–856, 2009.

[6] E. Theodorou, J. Buchli, and S. Schaal, “A generalized path integral control approach to reinforcement learning,” *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[7] E. Bizzi, V. Cheung, A. d’Avella, P. Saltiel, and M. Tresch, “Combining modules for movement,” *Brain research reviews*, vol. 57, no. 1, pp. 125–133, 2008.

[8] M. Kallmann, R. Bargmann, and M. Mataric, “Planning the sequencing of movement primitives,” in *proceedings of the international conference on simulation of adaptive behavior (SAB)*, pp. 193–200, 2004.

[9] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, “Learning sequential motor tasks,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2626–2632, IEEE, 2013.

[10] S. Schaal, S. Kotosaka, and D. Sternad, “Nonlinear dynamical systems as movement primitives,” in *IEEE International Conference on Humanoid Robotics*, pp. 1–11, 2000.

[11] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Probabilistic movement primitives,” in *Advances in Neural Information Processing Systems*, pp. 2616–2624, 2013.

[12] A. J. Ijspeert, J. Nakanishi, and S. Schaal, “Learning rhythmic movements by demonstration using nonlinear oscillators,” in *Proceedings of the IEEE/RSJ int. conference on intelligent robots and systems (IROS2002)*, no. BIOROB-CONF-2002-003, pp. 958–963, 2002.

[13] S. Kotosaka and S. Schaal, “Synchronized robot drumming by neural oscillator,” *JOURNAL-ROBOTICS SOCIETY OF JAPAN*, vol. 19, no. 1, pp. 116–123, 2001.

[14] M. Rakovic, B. Borovac, M. Nikolic, and S. Savic, “Realization of biped walking in unstructured environment using motion primitives,” *Robotics, IEEE Transactions on*, vol. 30, no. 6, pp. 1318–1332, 2014.

[15] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, “A survey on policy search for robotics.,” *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[16] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, “Learning movement primitives,” in *Robotics Research. The Eleventh International Symposium*, pp. 561–572, Springer, 2005.

[17] S. Schaal, “The sl simulation and real-time control software package,” *University of Southern California*, 2009.