DISSERTATION

to obtain the title of

Doctor of Technical Sciences

of

Graz University of Technology

Defended by

Gerhard NEUMANN

On Movement Skill Learning and Movement Representations for Robotics

Thesis Advisor:
O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang MAASS

defended on 23.04.2012

Jury:

Advisor:	O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang MAASS	-	TU Graz
Reviewer:	Univ.-Prof. Dr. Jan PETERS	-	TU Darmstadt
Dean of Studies:	Dipl. Ing., Dr. techn., Univ. Doz. Denis HELIC,	-	TU Graz

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, November 2011 ....

(signature)

Abstract

Modern robots are equipped with sophistacted compliant actuators, allowing the robot to perform highly dynamic complex movement skills, like different forms of locomotion, jumping or even playing tennis. However, classical control and engineering techniques typically fail for such complex tasks. A more promising perspective is to let the robot learn the movement skill by trial and error, which is also the main topic of this thesis. I will discuss 3 different topics that are strongly connected to motor skill learning and present new techniques for each of these fields which can be seen as a step towards learning rich and complex motor skills.

In the first part of the thesis I will examine reinforcement learning in continuous state and action spaces as foundation for motor skill learning. Here, I present two new methods, learning with adaptive state graphs and reinforcement learning by advantage weighted regression. Both methods can easily deal with continuous action spaces which are often considered as problematic for standard reinforcement learning methods.

In the second part, I discuss different types of movement representations. Movement representations are parametric descriptions of a movement plan, and therefore, provide a lower dimensional representation of the resulting trajectories. Choosing a compact movement representation which contains task relevant features can considerable facilitate learning of movement skills. Besides elaborating existing methods, I will introduce 3 new representations. I will introduce the kinematic synergy approach which provides a low dimensional representation of a high dimensional action space. Then I will present the motion template framework, which is the first movement representation which can be sequenced in time by the use of reinforcement learning. The last representation which I introduce is called *Planning Movement Primitive*. This representation employs planning already at the level of the movement representation and therefore allows the use of abstract goals or features of the trajectory as parameter representation, which allows fast learning of complex movement skills.

Finally, in the last part of the thesis, I will address the policy search problem, i.e. given a representation of the movement, how can we find a valid parameter setting by reinforcement learning? Here, I present a new method based on variational inference which generalizes policy search to different initial situations of the robot.

Zusammenfassung

Moderne Roboter sind heutzutage mit nachgiebigen Motoren ausgestattet welche die Ausführung von komplexen dynamischen Bewegungen erlauben, wie zum Beispiel ein zweibeiniger Gang, Hüpfen oder sogar Tennis spielen. Klassische Control-Algorithmen schlagen aber für solche komplexen Aufgaben meist fehl. Ein vielversprechenderer Ansatz ist es hingegen wenn der Roboter mittels Trial-and-Error Lernens die Bewegungen selbst erlernt. In dieser Dissertation werde ich 3 sehr wichtige Themenbereiche des Bewegungslernens diskutieren und neue Methoden präsentieren welche als Schritt in Richtung selbstständiges Lernen von komplexen Bewegungsabläufen gesehen werden können.

Im ersten Teil der Dissertation werde ich Reinforcement Learning in kontinuierlichen Zustands und Aktionsräumen als Grundlage des Bewegungslernens untersuchen. Hierzu werde ich zwei neue Methoden einführen, Lernen mit Adaptiven-Zustands Graphen und Reinforcement Learning by Advantage Weighted Regression. Beide Methoden können einfach mit kontinuierlichen Aktionsräumen umgehen welche für viele Standard Reinforcement Learning Methoden problematisch sind.

Im zweiten Teil dieser Dissertation werde ich verschiedene Bewegungsrepräsentationen disktutieren. Eine Bewegungsrepräsentation ist eine parametrische Beschreibung eines Bewegungsplannes, und beschreibt daher eine Trajecktorie mit typischerweise wenigen Parametern. Die richtige Wahl der Repräsentation kann das Erlernen einer Bewegung erheblich vereinfachen. Hier werde ich zunächst vorhandene Modelle diskutieren und dannach 3 neue Repräsentationen einführen. Der erste Ansatz, kinematische Synergies, dient dazu die Dimensionalität des Aktionsraumes eines Roboters zu verringern und dadurch das Kontroll-Problem erheblich zu vereinfachen. Als nächstes stelle ich den Motion Template Ansatz vor. Dieser Ansatz ist der 1. Ansatz der dazu verwendet werden kann um zu lernen verschiedene Bewegungen hintereinander auszuführen. Die letzte Bewegungsrepräsentation die ich vorstellen werde verwendet Plannungs-Algorithmen um die Bewegung zu generieren. Dies bringt den Vorteil dass man abstrakte Features oder Ziele der Bewegung direkt als Parameter der Repräsentation verwenden kann, welches des Erlernen einer Bewegung erheblich vereinfachen kann.

Der letzte Teil dieser Thesis beschäftigt sich mit Policy Search, also den erlernen eines geigneten Parameter-Vektors wenn man eine gegebene Bewegungsrepräsentation verwendet. Hier werde ich einen neuen Ansatz vorstellen welcher auf Variational Inferenz basiert und die Suche nach Parametern unter verschiedenen Anfangszuständen des Roboters ermöglicht.

Acknowledgements

I would like to thank my advisor Wolfgang Maass for his guidance and inspiring ideas, for the opportunity to participate in interesting research and supporting me during my excessive research.

I am also very grateful for the support of my second (inofficial) supervisor Jan Peters, who became a friend and always helped my out if there were a research-specific or even more private problems. Many thanks also go to Marc Toussaint and Auke Ijspert for their collaboration and guidance.

My gratitude also goes to all my current and former colleagues during my time at the Institute for Theoretical Computer Science, especially to Elmar Rückert, Stefan Häusler, Johannes Bill, Michael Pfeiffer, Helmut Hauser, Stefan Klampfl and Daniela Potzinger, who also became friends outside working life. It was really fun and enjoyable to work with such nice (and sometimes crazy) colleagues. Here, special thanks also goes to our system administrator Oliver Friedl, who had to suffer my permanent unintended attacks on the stability of our computer system.

I also want to acknowledge the financial support from Graz University of Technology, the Austrian Science Fund FWF, and various research programs of the European Union, including SECO, and AMARSi.

Last but not least, my deepest gratitude goes to my family and my friends. To my dad Hilmar, for his support and valueable advice, to my brother Stefan and my two best friends Peter and Hannes for their friendship, time, and encouragments. I also want to thank my former girl-friend Ulrike who attended me for a long and sometimes difficult time during my studies.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Modern robots are equipped with sophistacted compliant actuators, allowing the robot to perform highly dynamic complex movement skills, like different forms of locomotion, jumping or even playing tennis. Classical control and engineering techniques typically fail for such complex movements. A more promising perspective is to let the robot learn the movement skill by trial and error. Learning of such complex movement skills is still one of the major challenges in robotics research and the main topic of this thesis.

1.1 Reinforcement Learning

Movement skill learning can easily be formulated as reinforcement learning (RL, (Sutton, 1996)) problem - the robot autonomously tries different movements and gets evaluative feedback in form of reward. The agent has to adapt its movement such that the reward is maximized.

1.1.1 Markov Decision Processes

The RL framework can be nicely described by Markov decision processes (MDPs). A MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, P, r, p_0, \gamma \rangle$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space of the agent. The transition probabilities are given by $P(s_{t+1}|s_t, a_t)$ for $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$ and the reward of performing action a_t in state s_t is given by $r(s_t, a_t)$. The initial state distribution is given by $p_0(s)$ and γ denotes the discount factor. The underlying principle of MDPs is the Markov assumption, i.e. given the the current state s_t the transition model and the reward model is independent of past states and actions.

The task of a reinforcement learning agent is to find a policy $\pi(a_t|s_t)$ which minimizes the expected future discounted reward

$$R^\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 \sim p_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t)\right],$$

where the initial state s_0 is sampled according to p_0, actions a_t according to the policy π and the state transition according to our transition model. For expectations, I will always use this notation - i.e. after the condition operator '|', it is always given according to which random variables the expectation is calculated.

In robotics, we typically deal with continuous state and action spaces, thus I will always write states $\mathbf{s}$ and actions $\mathbf{a}$ in vector notation.

1.2 Movement Skill Learning

The field of movement skill learning has made considerable progress in recent years which is documented by several success stories. In (Peters and Schaal, 2006), a robot learned to swing a baseball pat in order to hit a ball. In (Kober and Peters, 2010), the game 'ball in the cup' was learned. Here, the learning performance reported by the authors was even comparable to a human child. Other impressive results include playing table tennis (Mülling et al., 2010), performing a jumping movement with a dog-like robot (Theodorou et al., 2010a) and running with a cheetah-like (simulated) robot (Wawrzynski, 2009).

However, despite of these success stories, learning, reusing and combining a rich set of complex movement skills is still one of the major challenges in robotics research. The probably largest problem with robot learning are the high-dimensional continuous state and action spaces. Our robot has typically many degrees of freedom (DoF), ranging from 7 for an anthropomorphic robot arm up to 30 for humanoid robots. If we also include the dynamic state of the robot, and thus the joint velocities, we quickly reach 15 to 60 state variables. This is out of the scope for most RL methods. In addition, the continuous control vector is also high-dimensional (one control variable for each DoF), which is beyond the scope of many RL algorithms.

Movement skill learning algorithms can be coarsely divided into value-based and policy search learning algorithms. Value-based algorithms estimate a the value function $V(\mathbf{s})$. The value function tells us the expected future reward if the agent follows a certain policy. Value-based approaches are in theory very efficient. The value function can be used to evaluate every intermediate action of a trajectory, i.e. we know which actions are responsible for the good or bad evaluation of a trajectory? This is often called the temporal credit assignment problem. However, value-functions are usually difficult to estimate in high dimensional continuous state and action spaces. For this reason most of the more recent movement skill learning algorithms try to avoid a direct representation of the value function. For a more detailed description on value-based methods we refer to Chapter 2.

Policy search algorithms on the other hand rely on a parametric representation of the policy and directly try to optimize the policy parameters without explicitly estimating a value function. Hence, in difference to value based algorithms, we can not directly evaluate single actions, but we can only evaluate the costs of the whole trajectory (by performing whole rollouts on the real system). However, since learning a value function is problematic for high dimensional continuous state spaces, more impressive results could be achieved by policy search methods, and therefore, recent research on policy search algorithms was intensified. We will discuss policy search methods in more detail in Chapter 9.

The performance of policy search methods strongly depends on the used movement representation. Choosing an adequate movement representation can increase learning speed of such methods considerably. The most common method is to use a local movement representation. Local representations directly describe the shape of the specific movement trajectory. Therefore, they can only be applied for using the same starting condition in each episode - different starting positions would result in different trajectories which often requires relearning. The setup with the single starting state is often also referred to as the *episodic reinforcement learning*. While this restricts the representational power of the policy, it also considerably simplifies the learning task. Many learning tasks have only become feasible by the episodic task assumption. Most of the success stories in robotic motor skill learning use a local representation, while global representations, i.e. a representation which can be used for any state, are more difficult to learn and therefore less commonly used. We will discuss different local movement representations in more detail in Chapter 5.

1.3 Structure of this Thesis

This thesis is divided into 3 parts. For each part I first introduce relevant concepts in the introduction chapter, the subsequent chapters are always based on published or almost submitted papers where I significantly contributed as first or second author. The first part of the thesis discusses value-based methods for learning in continuous state and action spaces. After the introduction I will present two new value-based approaches. The first method, discussed in Chapter 3, is a graph-based method. It represents the continuous state space by a discrete set of nodes in a graph. The graph is built from experience and grows during learning. The benefit of the graph-based approach is that we can use local controllers, which are employed to navigate between nodes, as form of prior knowledge. The local controllers also provide an efficient treatment of continuous actions. The second value-based method presented in this thesis is based on weighted regression. We use a weighted regression to simplify the max-operator which usually has to be performed in the action space. This operator is hard to perform for continuous actions. We prove that an advantage-weighted regression can be used to replace the max-operator, resulting in a more efficient value-based algorithm suitable for continuous action spaces.

The second part of the thesis discusses movement representations. Here, I will present 3 new methods. In Chapter 6 I introduce a new representation which we denoted as *kinematic synergies*. It provides a lower dimensional manifold of the high-dimensional action space of a (in this case humanoid) robot. The use of synergies significantly simplifies the control of the robot. We applied our approach to balancing the humanoid robot HOAP-2. In Chapter 7 I introduce the motion template representation, which is the first movement representation for which a reinforcement learning algorithm can be used to combine the templates sequentially in time. We applied the motion template approach for complex 2-link pendulum swing-up and balancing tasks. In the last chapter of this part of the thesis I present a primitive which uses inherent probabilistic planning to generate the movement. The inherent planner allows to use abstract features or goals of the movement as parameters. As we will show this representation can simplify the learning problem in comparison to the commonly used approaches considerably.

The last part of the thesis is devoted to policy search algorithms. Here, I will present a new approach in Chapter 10 which is based on variational inference and can generalize policy search to multiple initial situations simultaneously.

Part I

Reinforcement Learning with Continuous State and Action Spaces

Chapter 2

Introduction

In this part of the thesis I will discuss value-based reinforcement learning methods which are also suited for continous action spaces. After giving a short introduction into relevant concepts I will present two new value-based approaches which were published in (Neumann et al., 2007) (Chapter 3) and (Neumann and Peters, 2009) (Chapter 4).

2.1 Value-based Methods

Value-based methods (Bertsekas and Tsitsiklis, 1998) estimate the beliefed accumulated future reward for each state $\mathbf{s}$ if following a policy π, i.e

$$V^\pi(\mathbf{s}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t), \mathbf{s}_{t+1} \sim P(\cdot|\mathbf{s}_t, \mathbf{a}_t)\right],$$

where $r_t = r(\mathbf{s}_t, \mathbf{u}_t)$ is the reward for each time step and γ is the discount factor. $V^\pi(\mathbf{s})$ is also called the value function of policy π. The value function can also be written in its recursive form, i.e.

$$V^\pi(\mathbf{s}) = \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + \gamma V^\pi(\mathbf{s}')\big|\mathbf{a} \sim \pi(\cdot|\mathbf{s}), \mathbf{s}' \sim P(\cdot|\mathbf{s}, \mathbf{a})\right].$$

The optimal value function is defined as

$$V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s}) = \max_{\mathbf{a}} \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + \gamma V^*(\mathbf{s}')\big|\mathbf{s}' \sim P(\cdot|\mathbf{s}, \mathbf{a})\right],$$

which is also known as the *Bellman-Optimality* principle. The V-function evaluates exclusively states and is therefore not directly applicable for decision making. For decision making we need a function which evaluates state-action pairs. This function is usually denoted as the Q-function. The Q-function $Q^\pi(\mathbf{s}, \mathbf{a})$ of policy π is defined as the accumulated reward if we take action $\mathbf{a}$ in the first step and subsequently again follow policy π

$$Q^\pi(\mathbf{s}, \mathbf{a}) \quad = \quad \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + \gamma Q^\pi(\mathbf{s}', \mathbf{a}')\big|\mathbf{s}' \sim P(\cdot|\mathbf{s}, \mathbf{a}), \mathbf{a}' \sim \pi(\cdot|\mathbf{s}')\right]$$

The optimal Q-function is defined as

$$Q^*(\mathbf{s}, \mathbf{a}) \quad = \quad \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}', \mathbf{a}')\big|\mathbf{s}' \sim P(\cdot|\mathbf{s}, \mathbf{a})\right]$$

Note that the V-function can be easily evaluated if the Q-function is known

$$V^\pi(\mathbf{s}) = \mathbb{E}\left[Q^\pi(\mathbf{s}, \mathbf{a})\big|\mathbf{a} \sim \pi(\cdot|\mathbf{s})\right], \qquad V^*(\mathbf{s}) = \max_{\mathbf{a}} Q^*(\mathbf{s}, \mathbf{a}) \tag{2.1}$$

There are many ways to represent the value function. In the most simple setup of a discrete state space we can use a tabular representation. For continuous state spaces we have to rely on parametric or non-parametric function approximators. For a more detailed discussion on different function approximator schemes please consult Section 2.2. Before coming to this section we will briefly review existing methods to learn the value function.

The V-function (or Q-function) is typically estimated from experience, i.e. by the use of the data points $\langle \mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1} \rangle$. Different methods can be applied in this context, here, we will briefly discuss Temporal Difference (TD) Learning, Batch-Mode RL and model-based RL.

2.1.1 Temporal Difference Learning

Temporal Difference (TD) methods incrementally estimate the V- or Q-function from samples. The gathered experience of a single step $\langle \mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1} \rangle$ is used to to calculate the temporal difference error, which is defined as the 1-step prediction error of the V-function

$$\delta_t = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t).$$

For tabular representations of the V-function the TD-error can straightforwardly be used to update $V(\mathbf{s}_{t+1})$, i.e.

$$V(\mathbf{s}_t) = V(\mathbf{s}_t) + \alpha\delta_t.$$

In the case of parametric function approximators, we have to rely on gradient-based methods (Bertsekas and Tsitsiklis, 1998; Sutton, 1996; Baird, 1995), however, these methods are either only proofed to converge for special cases like linear function approximators (Bertsekas and Tsitsiklis, 1998) or are known to have a very slow convergence rate (Baird, 1995). The data point $\langle \mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1} \rangle$ is typically only used once to update the V or Q-function, subsequently the data point is dismissed.

2.1.2 Batch-Mode Reinforcement Learning

Batch-Mode RL methods use the whole history of the agent to update the V- or Q-function which allows a more efficient data usage than for standard TD methods.

The first application of Batch-Mode RL was a method called 'Experience Replay' (Lin, 1992) (EP). EP is basically just an extension of TD-learning. After each time step, K imaginary time steps out of the history of the agent are shown to the TD-learning algorithm and used to update the Q or V-function. While there has been a very recent and impressive extension of this approach for using actor critic algorithms with neural networks (Wawrzynski, 2009), EP is limited to the function approximator techniques which can be used in the online setup, which excludes regression trees (Ernst et al., 2005) or Gaussian Processes (Deisenroth et al., 2009).

More recent work in batch mode RL has concentrated on Fitted Q-iteration (FQI) (Ernst et al., 2003). FQI iteratively approximates the Q-function by using the whole batch of experienced data points $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i >\}_{1 \le i \le N}$. For each data-point we calculate the target Q-value $\tilde{Q}(i)$ by using the old estimate of the Q-function at the successor states, i.e.

$$\tilde{Q}_{l+1}(i) \quad = \quad r_i + \gamma V_l(\mathbf{s}_i') = r_i + \gamma \max_{\mathbf{a}'} Q_l(\mathbf{s}_i', \mathbf{a}') \tag{2.2}$$

Learning the Q-function $Q_{l+1}(\mathbf{s}, \mathbf{a})$ then defines a new regression problem. As this regression problem is formulated with the whole batch of data, also batch-model supervised regression methods can be used. The whole process has to be repeated for L times in order to calculate the optimal Q-function for the next L steps (thus, L typically needs to be quite high). FQI can be used with all types of function approximators, very good results have been shown with regression trees and neural networks. In Chapter 4 we present a new FQI method which uses a weighted regression to approximate the max operator over the action space in Equation 2.2. This allows an efficient treatment of continuous action spaces.

2.1.3 Model-Based Techniques

The model-based variant of FQI is fitted V-iteration (Boyan and Moore, 1995) (FVI). In FVI we iteratively fit the optimal V-function instead of the Q-function. However, in order to do so, we have to know the transition model $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ and the reward model $r(\mathbf{s}, \mathbf{a})$ of the MDP. Both models can again be learned from data or might already be given as prior knowledge.

$$\tilde{V}_{k+1}(\mathbf{s}_i) \quad = \quad \max_{\mathbf{a}} r(\mathbf{s}_i, \mathbf{a}) + \gamma \int_{\mathbf{s}'} P(\mathbf{s}'|\mathbf{s}_i, \mathbf{a}) V_k(\mathbf{s}') d\mathbf{s}_i. \tag{2.3}$$

Usually the integral over $\mathbf{s}'$ in continuous state spaces is very hard to perform. This has limited the use of fitted V-iteration. However, recent work with Gaussian Processes (Deisenroth et al., 2009) could circumvent this problem. Due to the Gaussian transition probabilities the integral can be solved analytically. A similar, slightly simpler kernel based approach was used in (Jong and Stone, 2007) to estimate the transition probabilities. The transition model was estimated by a simple linear averager using a Gaussian similarity kernel. Subsequently the V-function can be calculated by the use of Prioritized Sweeping (PS, (Moore and Atkeson, 1993)), which is much more efficient than fitted V-iteration. However, this method can only be applied to simple, linear function approximators like linear averagers. This has so far limited this approach to very simple applications.

In Chapter 3, we present a new model-based method which uses a graph-based representation. Here, the state space is represented by a discrete set of nodes of a graph which is built adaptively from experience. The graph is called an 'Adaptive-State Graph' (Neumann et al., 2007). Nodes between the graph can be reached by the use of local controllers, which are assumed to be part of the prior knowledge. Because we can always use the local controller to navigate directly to the nodes in the graph the V-function only needs to be represented at these discrete set of nodes. Furthermore, simple planning methods such as value iteration can be applied to the discrete graph in order to calculate the V-function.

2.2 Continuous State Spaces : V-Function Approximation

In continuous state spaces we have to rely on function approximation techniques to estimate the V or Q-function. Many types of approximators can be applied in this context, including linear function approximators (Sutton, 1996; Timmer and Riedmiller, 2007), neural networks (Riedmiller, 2005), regression trees (Ernst et al., 2005), local regression techniques (Neumann and Peters, 2009) and Gaussian Processes (Deisenroth et al., 2009). All these methods can be easily applied to batch-model RL, however, as some approximators inherently use batch updates (such as regression trees or Gaussian Processes), TD-learning methods have the restriction that not all available function approximators can be used.

2.2.1 Linear Function Approximators

The most rigorous convergence proofs exists for linear function approximators. Linear function approximators typically use D (non-linear) features $\Phi_i(\mathbf{s})$ which are linearly combined to approximate the V-function

$$V(\mathbf{s};\mathbf{w}) = \sum_{i=1}^{D} \Phi_i(\mathbf{s}) w_i = \mathbf{\Phi}(\mathbf{s})^T \mathbf{w}.$$

The features $\Phi_i(\mathbf{s})$ have to be predefined by the user. Finding a good feature representation is non-trivial and considered to be one of the biggest problems when using linear function approximators.

Due to the linear representation the V-function (or Q-function) for a given policy can be easily calculated in batch-mode by employing least-square solution techniques, resulting into the Least-Square Temporal Difference (LS-TD) (Boyan, 1999) algorithm. This algorithm can only be used for policy evaluation, i.e. estimating the value function of a given policy. It's variant Least-Square Policy Iteration (LS-PI) (Lagoudakis and Parr, 2003) can also be used for finding the optimal policy. A popular method to define the features is to use grid-based RBF-networks or tile-codings (Sutton and Barto, 1998). However, this usually fails for high dimensional state spaces because these methods suffer from the curse of dimensionality, i.e. the number of features scales exponentially with the number of dimensions.

A potential approach to avoid the problem of defining meaningful features by hand has been proposed in (Kolter and Ng, 2009a). Here, an huge amount of randomly defined features can be used. In order to avoid overfitting, a L1-norm regularization term has been used. The resulting algorithm is called Lasso-TD. Still the application of this method has so far been limited to rather simple tasks such as the pendulum swing-up task (Kolter and Ng, 2009a).

2.2.2 Non-Linear Methods

Non-linear methods are usually more flexible than linear function approximators and do not require manual tuning of the feature representation. They are typically difficult to use for TD-learning, however, impressive results could be shown by the use of batch RL methods.

A commonly used non-linear approach are feedforward neural networks (NNs). In (Wawrzynski, 2009), the locomotion of a planar simulated cheetah robot is learned by using neural networks and Experience Replay. NNs have also been applied successfully with FQI (Riedmiller, 2005). Here, the main work has been done in the context of Robocup. This method have been applied for learning to dribble with a wheeled robot (Riedmiller et al., 2009), learning to control a omni-directional drive (Riedmiller et al., 2009) or learning to control a slot car-racer (Kietzmann and Riedmiller, 2009).

Another popular non-linear approximation technique which has been used for FQI are regression trees, i.e. Extremely-Randomized Trees (ExtRa Trees). In (Ernst et al., 2005), the ExtRa-Trees have been applied to many standard optimal control tasks for RL showing that it outperforms online RL. The tree-based approach has also been applied to simulated HIV (Ernst et al., 2006) and epilepsy treatment (Guez et al., 2008) tasks. Both methods, FQI with neural network or regression trees, have been used as baseline methods in my paper (Neumann and Peters, 2009) where I introduced a new batch-mode RL algorithm based on advantage weighted regression.

In (Deisenroth et al., 2009), Gaussian Processes (GPs) have been used to approximate the V-function as well as the transition model. As GPs show very good generalization properties, this is one of the most efficient value-based methods seen so far.

2.3 Continuous Action Spaces : The greedy operator

Another subtle point when using RL for robotics are the continuous action spaces. From the definition of the optimal V-function we can see that we have to perform the greedy operator $\max_{\mathbf{a}}$ over the whole action space. The standard approach for is to use a discretized set of actions, however, this becomes very inefficient if we deal with high dimensional action spaces or we need to approximate the policy very accurately.

In this part of the thesis I will present 2 methods which can solve the $\max_{\mathbf{a}}$-operator efficiently and hence are well suited for continuous action spaces. In Chapter 3 I introduce a graph-based RL method. Instead of using continuous actions the agent can choose the next node he wants to reach within the graph. The decision of the agent is therefore discrete, and the max-operator is again easy to solve. The navigation to the desired node is then done by a local controller, which inherently uses continuous valued actions. As the graph is adapted to the current task, the nodes of the graph, and hence the discrete actions of the agent, are always located in areas relevant for the task.

In Chapter 4 I present a novel method to approximate the $\max_{\mathbf{a}}$-operation. Here, we have shown that by the use of a soft-greedy action selection mechanism the $\max_{\mathbf{a}}$ operator can be efficiently approximated by an advantage-weighted regression (AWR). The AWR can be performed very efficiently and therefore considerably simplifies the use of value-based RL methods for continuous actions.

Chapter 3

Reinforcement Learning with Local Controllers and Adaptive State Graphs

Contents

In this chapter we present a new reinforcement learning approach to finding optimal solutions for continuous control problems in unknown environments with arbitrary reward functions. We assume that the local system dynamics in such problems can be efficiently approximated with simpler models, still it is a hard task to design globally optimal trajectories. The presented algorithm uses directed exploration to build an adaptive state graph of sample points within the continuous state space. Global solution trajectories are formed by combining local controllers that connect nodes of the graph. A new generalization technique exploits the connectivity of the state graph to predict rewards of unexplored edges. We demonstrate our approach on complex movement planning tasks with continuous states and actions in continuous time.

3.1 Introduction

Finding near-optimal solutions for continuous control problems is of great importance for many research fields. Many of these problems involve difficult reward or cost functions, which are usually not exactly known in advance. In the weighted region problem (Mitchell and Papadimitriou, 1991) in path-planning, for example, we need to find the shortest path to a goal state through regions of varying movement costs. This task is challenging already, but becomes much harder if the agent neither has a map of the environment nor knows the exact costs associated with the regions. In robotics arbitrary reward functions can be used e.g. to enforce obstacle avoidance or stable and energy-efficient movements. Most existing approaches to these problems require either complete knowledge of the underlying system, or are restricted to simple reward functions. In this chapter we present an approach to this problem that utilizes minimal prior knowledge and deals with arbitrary reward functions, in order to efficiently learn high quality control strategies for continuous problems in continuous time.

Reinforcement learning (RL) (Sutton and Barto, 1998) is an attractive framework for the addressed problems. It can learn optimal policies through interaction with an unknown environment. In continuous environments, the most common approach is to use parametric approximations to the value functions. However, several authors have reported problems concerning the learning speed, quality and robustness of the solutions (Baird, 1995; Boyan and Moore, 1995). Our proposed method transforms the continuous problem into a discrete Markov decision process (MDP) on a finite set of sample states, using simple local controllers to navigate between them. Such hierarchical decompositions of the policy are known to speed up the search for optimal solutions (Sutton et al., 1999). Local controllers for small regions of the state space are often easily available, and can be seen as minimal prior information about the task's underlying system dynamics. Local controllers do not assume complete knowledge of the environment (e.g. location of obstacles), and are therefore not sufficient to find globally optimal solutions.

The idea of using local controllers has been applied very successfully in sampling-based planning methods (Kavraki et al., 1996; Kuffner and LaValle, 2000). These methods build a graph consisting of random sample points and connect them with local controllers. A global solution is constructed by combining the paths of several local controllers to a path that leads to the goal. The two most prominent approaches of this style are rapidly exploring random trees (RRTs) (Kuffner and LaValle, 2000) and probabilistic roadmaps (Kavraki et al., 1996), which were developed for kinematic path planning in Euclidean configuration spaces. Planning techniques are very efficient, but their application is limited to completely known environments.

Our proposed algorithm combines the advantages of RL and local planning to efficiently learn high quality policies in initially unknown continuous environments with arbitrary reward functions. The algorithm explores the state space and builds an *adaptive state graph* of sample points that are connected by local controllers. We developed an online approach to building this graph, which immediately incorporates feedback from the environment, like reward signals or unexpected transitions. We present exploration heuristics to initially cover the state space sparsely, but still sufficiently to find ways to a goal state. Later the graph is refined in critical regions. A novel generalization scheme predicts rewards for unexplored edges, to avoid unnecessary exploration and find better solutions faster. The adaptive state graph transforms the continuous control problem into a discrete MDP, for which the optimal policy can be calculated with exact planning algorithms like dynamic programming. This results in more accurate policies and reduced running time in comparison to function approximation techniques. Our algorithm naturally deals with continuous actions and continuous time steps, which leads to smoother and more natural trajectories (Doya et al., 2000).

The idea of combining local controllers with RL has been studied in the past: The Parti-game algorithm (Moore and Atkeson, 1995) divides a continuous state space into cells of varying size and uses local controllers to navigate between the cells. Parti-game in its original formulation cannot maximize arbitrary reward functions, but is restricted to finding paths to a goal state through regions of homogeneous reward. The Parti-game idea was extended to value function approximation for general continuous control problems in (Munos and Moore, 2002). In contrast to our method they assume knowledge of the whole environment and do not make use of local controllers. (Guestrin and Ormoneit, 2001) have used combinations of local controllers for static path planning tasks in stochastic environments. Their graph is built from uniform samples over the whole state space, rejecting those that result in collisions. They also assume that a detailed simulation of the environment is available to estimate the costs and success probabilities of every transition.

The main motivation for the design of a new algorithm is that none of these approaches can handle unknown and arbitrary reward functions at the same time. Remaining alternatives are standard function approximation and model-based RL techniques like Prioritized Sweeping (Moore and Atkeson, 1993). Model-based algorithms learn reward and transition models, which are used for offline updates of the value function. In this paper we demonstrate on various problems that our algorithm achieves faster convergence and finds more accurate solution trajectories than widely used RL techniques.

In the next section we introduce the basic setup of our algorithm. Section 3.3 shows how the adaptive state graph is constructed and refined, making use of the reward prediction scheme introduced in Section 3.4. In Section 3.5 we evaluate our algorithm on various static and dynamic path finding tasks and a planar 3-link arm reaching task, before concluding in Section 3.6.

3.2 Graph Based RL

We consider episodic, deterministic control tasks in continuous space and time, in which the agent's goal is to move from an arbitrary starting state to a fixed goal state with maximal reward. In the beginning the agent only knows the locations of the start and goal state, and can use local controllers to navigate to a desired target state in its neighborhood. We will first define the mathematical notation for this problem and then sketch the various steps of the algorithm for finding good solution trajectories.

3.2.1 Mathematical Problem Formulation

Let $\mathcal{X}$ define the state space of all possible inputs $x \in \mathcal{X}$ to a controller. We require $\mathcal{X}$ to be a metric space with given metric $D : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$. Control outputs $u \in \mathcal{U}$ change the current state x according to the system dynamics $\dot{x} = f(x, u)$. In this paper we assume that only an approximate local model $\hat{f}(x, u)$ is known, which does not capture possible nonlinearities due to obstacles. The objective is to find a control policy $\mu : \mathcal{X} \to \mathcal{U}$ for the actual system dynamics $f(x, u)$ that returns for every state x a control output $u = \mu(x)$ such that the agent moves from a starting state $x^S \in \mathcal{X}$ to a goal state $x^G \in \mathcal{X}$ with maximum reward.

Our algorithm builds an adaptive state graph $\mathcal{G} = \langle V, E \rangle$, where the nodes in $V = \{x_1, \ldots, x_N\} \subset \mathcal{X}$ form a finite subset of sample points from $\mathcal{X}$. We start with $V_0 = \{x^S, x^G\}, E_0 = \emptyset$ and let the graph grow in subsequent exploration phases. The edges in $E \subseteq V \times V$ correspond to connections between points in V that can be achieved by *local controllers*. The local controller $a(e)$ for an edge $e = (x_i, x_j)$ tries to steer the system from x_i to x_j, assuming that the system dynamics along the path correspond to $\hat{f}(x, u)$. The adaptive state graph contains only those edges that can be traversed by a local controller, but the combination of multiple edges yields globally valid trajectories.

For an edge e we define $t(e)$ as the time needed for the complete transition and $r(e)$ as the total reward obtained on the edge. In our approach we separate the total reward into two components: $r(e) = r^{\text{goal}}(e) + r^{\text{trans}}(e)$. The *goal* reward r^{goal} is given upon reaching the goal state. We assume here that the location of the goal, and thereby the goal reward, is known in advance. The *transition* reward r^{trans} captures all other rewards that result from interactions with the environment, such as time- and action-dependent transition costs, punishments for collisions, and negative or positive location-dependent rewards.

For a given graph $\mathcal{G}$ we have to solve the discrete problem of finding a policy $\pi : V \to E$ that selects at every node $x_i \in V$ an outgoing edge $\pi(x_i) = e_{ij} = (x_i, x_j)$ and moves to the successor node x_j. The objective is to find a policy π which produces a sequence of edges $\langle e_0 = (x^S, x_1), \ldots, e_i = \pi(x_i) = (x_i, x_{i+1}), \ldots, e_n = (x_n, x^G) \rangle$ that starts in x^S and ends in x^G, such that the (possibly discounted) sum of rewards $R^\pi := \sum_{i=0}^{n-1} \gamma^{i-1} r(e_i)$ is maximized.

For this task we use value iteration (Sutton and Barto, 1998), which is a dynamic programming approach to finding optimal value functions in discrete MDPs with known reward and transition functions. The advantage of value iteration is that we can propagate new reward information quickly throughout the whole graph. This may be used to update the values of all nodes whenever the agent receives new information about the graph, e.g. when new nodes and edges are inserted. Value iteration is guaranteed to converge to an optimal policy (Puterman, 1994), based on the knowledge contained in the adaptive state graph.

3.2.2 Sketch of the Algorithm

The agent interacts with the environment by using local controllers to move between states that are contained as nodes in the adaptive state graph. Initially the graph is empty, except for the start and goal state. New nodes and edges are created in the *initial exploration phase* by simulating the approximate model $\hat{f}(x, u)$ from the current node to generate potential successor states. The exploration heuristics directs the agent towards the goal, and the policy chooses between following an edge that is already in the graph, or exploring a new successor state. In the latter case the state is added as a new node into the graph. Whenever a new node is inserted into the graph we also add all possible edges to neighboring nodes that can be achieved by local controllers. We use a reward prediction technique to estimate the rewards for those edges. These predictions are later improved by actually experienced rewards from similar edges. We re-plan the policy with value iteration whenever new nodes and edges become available, or an unpredicted reward is obtained during a transition.

When the agent actually reaches the goal state we reduce exploration and the algorithm enters the *graph refinement* phase. In this phase the currently best path is optimized by adding new nodes to the graph and improving predictions for the edge rewards.

The quality of the resulting policy depends on the available edges and nodes of the graph, but also on the quality of the local controllers. We assume here that local controllers can compute near-optimal solutions to connect two states in the absence of unforeseen events. We restrict ourselves here to rather simple system dynamics, for which controllers are easily available. In Euclidean spaces we typically try to connect two states with a straight line. Extending the approach to non-linear dynamics or even learning the local controllers for more complex dynamical systems is part of future work.

3.3 Building the Adaptive State Graph

Previous approaches for sampling-based planning, e.g. (Guestrin and Ormoneit, 2001; Kavraki et al., 1996), have used uniform random sampling of nodes over the whole state space. This requires a large number of nodes, of which many will lie in irrelevant or even unreachable regions of the state space. On the other hand, a high density of nodes in critical regions is needed for fine-tuning of trajectories. The presented algorithm iteratively builds a graph by adding states that are visited during two phases of online exploration: In the *initial exploration* phase we use heuristic exploration scores to direct the search towards the goal state. During *graph refinement* new nodes are added along successful trajectories to optimize solutions found so far.

3.3.1 Initial Exploration Phase

Initially the agent needs to search for a path to the goal state, thereby expanding the adaptive state graph into previously unknown regions. For every state x_i that the agent visits we create a set of potential *successor states* $\tilde{x}_i^j$. This is done by simulating the dynamical system $\hat{f}(x, u)$ with different control laws for a specific amount of time. The control actions and execution times can either be fixed in advance, or randomly picked from a distribution over control actions and times. To ensure exploration into unvisited areas we immediately reject successor states that are closer than some threshold $\theta_{\min}^{exp}$ to existing nodes in the graph.

Our algorithm directs exploration towards the goal, but at the same time abandons paths with high negative rewards, which are unlikely to be included in optimal paths from start to goal. We therefore store a global queue Q of the most promising successor states $\tilde{x}$, ranked by an *exploration score* $\sigma_{\text{exp}}(\tilde{x})$. This score is equivalent to the estimated sum of rewards for a path that first goes from the starting state x^S to $\tilde{x}$ and then follows the direct path to the goal state x^G. The reward to reach

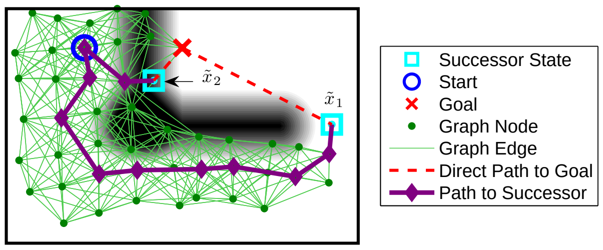


Figure 3.1: Illustration of the exploration process. Exploration is rather continued at successor $\tilde{x}_1$ than at $\tilde{x}_2$, because the reward to reach $\tilde{x}_2$ is strongly negative.

$\tilde{x}$ from x_S can be easily calculated from the current graph. The reward for the direct path to the goal state from $\tilde{x}$ is estimated by the reward of a simulated local controller, ignoring any obstacles on the path. Since we always assume that the goal is reached, we also add the goal reward r^{goal}. Tuning this parameter either enforces stronger exploration if r^{goal} is large, or narrows the search space if r^{goal} is small. Figure 3.1 illustrates exploration scores in a puddle world task (see Section 3.5.1), where shaded regions indicate negative rewards. In this example $\tilde{x}_1$ has a higher exploration score than $\tilde{x}_2$, because reaching $\tilde{x}_2$ requires traversing a region of large negative reward.

The k highest scored successor states in the queue Q are candidates for exploration. Before running the value iteration we insert these k successors as terminal nodes into the graph, and add virtual *exploration edges* from the nodes from which they were created. The rewards of these edges are the estimated rewards-to-goal. The policy computed by value iteration may then either choose an exploration edge, thereby adding a new node to the adaptive state graph, or move to an already visited node. The latter indicates that exploring from other nodes seems more promising than continuing the exploration at the current node.

Whenever a new node is inserted into the graph we also add new edges by simulating local controllers to all neighboring nodes under the local dynamics $\hat{f}(x, u)$. The local controllers additionally yield predictions for the transition reward of these edges, representing the estimated transition costs in the absence of unforeseen events, such as obstacles. In Section 3.4 we describe a technique to use information from the graph to get more accurate reward predictions. The true transition reward for an edge is not known until it is actually traversed for the first time. We then either replace the predictions by the true values, or delete the edge if we discover that the local controller cannot complete the connection (e.g. because of an obstacle in between the nodes). New reward information is also used to update predictions for unvisited edges.

When the goal state is actually reached, the policy may still continue to visit successor nodes in Q, if their exploration scores are higher than the sum of rewards on the currently best path from the start to the goal. The initial exploration phase stops when no such successors remain. If the exploration phase does not find a solution trajectory within a given time, we use a finer resolution of nodes. This can be done by using smaller time steps for the creation of successor nodes, or by decreasing $\theta_{\text{min}}^{\text{exp}}$, thereby allowing successor nodes to lie closer to already visited nodes.

3.3.2 Graph Refinement Phase

Graph refinement starts after the initial exploration phase, and optimizes the graph to find better trajectories. The basic idea is to find bottlenecks on the best trajectory found so far, i.e. nodes where the number of outgoing edges is small. For some nodes a low outdegree is sensible, e.g. because they are located at narrow passages. For other nodes this may just reflect a lack of alternatives, and so generating new successor nodes and outgoing edges may improve the current policy.

The main component of graph refinement is an offline process, in which we stochastically select nodes for optimization and add new successor nodes. For every node x_i in the graph we compute an *optimization score* $\sigma_{opt}(x_i)$, which is the sum of rewards on the best path in the graph from the start to the goal via x_i. The probability of selecting x_i for optimization is proportional to $\sigma_{\text{opt}}(x_i)$, and indirectly proportional to the number of outgoing edges in x_i. This gives higher probability to nodes on good solution trajectories and nodes with small outdegrees. Let x^* be the node selected for optimization, chosen according to the described probability distribution. We then create a new successor node by simulating the local system dynamics $\hat{f}(x, u)$ from x^*, using a random variation of the control law of the optimal outgoing edge from x^*. New edges and predictions for the rewards are generated as in the initial exploration phase.

The insertion of new nodes is typically performed after a fixed number of episodes, in which we collect online experience. This is done by following an ε-greedy policy that explores new edges and nodes, and uses the gathered reward information to update predictions for unseen edges.

3.4 Reward Prediction

Generalization of learned results for unseen states or actions is a well-known concept in reinforcement learning (Sutton and Barto, 1998). In our case we want to predict the transition rewards for unseen edges of the adaptive state graph. This speeds up the learning process and avoids unnecessary exploration of all edges. The general idea is to exploit local similarities, i.e. parallel connections of similar regions of the state space are likely to have similar rewards.

During the agent's exploration, our approach uses new information about the reward of the currently traversed edge $e_{cur} = (x_i, x_j)$ to update the predictions for similar edges. We say that two edges are similar if both their starting and target nodes lie within certain neighborhoods. We call the region $S_{\varepsilon_s}(x_i) = \{x \in \mathcal{X} \mid D(x, x_i) < \varepsilon_s\}$ around the starting point x_i the *starting area*, and the region $T_{\varepsilon_t}(x_j) = \{x \in \mathcal{X} \mid D(x, x_j) < \varepsilon_t\}$ around the target point x_j the *target area*. An edge is similar to e_{cur} if its starting point lies in $S_{\varepsilon_S}(x_i)$ and its target point lies in $T_{\varepsilon_t}(x_j)$. This case is illustrated in Figure 3.2(a).

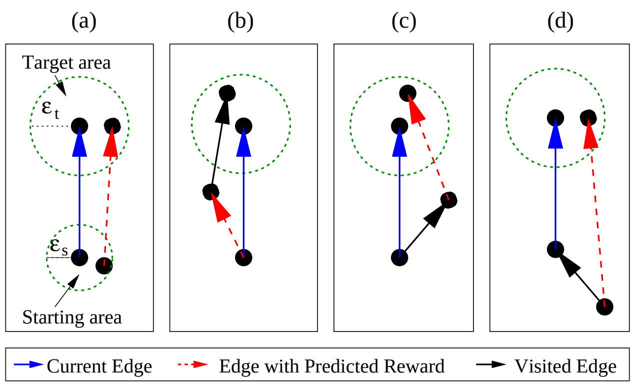


Figure 3.2: Four cases for online local reward prediction. The currently traversed edge is drawn blue, previously visited edges are drawn black, and the prediction is for the unvisited red edge. (a) Edges from starting area to target area; (b) Unvisited edges that are connected to the target area via previously visited edges; (c) Unvisited edges that connect from direct successors of the starting node to the target area; (d) Direct edges from predecessors of the starting node to the target area.

More updates can be performed if we consider indirect connections from $S_{\varepsilon_s}(x_i)$ to $T_{\varepsilon_t}(x_j)$, which use paths of two edges to connect the two regions (see Figure 3.2 (b) and (c)). Paths longer than two edges are not considered, because they may lead through completely different regions, thereby violating our assumption of local similarity of the connections. Let $e_1 = (x_1, x_2), e_2 = (x_2, x_3)$ be a 2-edge indirect connection with $x_1 \in S_{\varepsilon_x}(x_i), x_3 \in T_{\varepsilon_t}(x_j)$. Using the current edge reward $r(e_{cur})$ as an approximation to the total reward of the alternative path, we can assign shares of $r(e_{\mathrm{cur}})$ to unvisited edges, proportional to their durations $t(e_1)$ and $t(e_2)$:

$$\hat{r}(e_1) = \frac{r(e_{\mathrm{cur}}) \cdot t(e_1)}{t(e_1) + t(e_2)} \qquad \hat{r}(e_2) = \frac{r(e_{cur}) \cdot t(e_2)}{t(e_1) + t(e_2)}$$

The current edge $e_{\mathrm{cur}} = (x_i, x_j)$ may also complete a 2-edge path from one of the predecessors of x_i to the target node (see Figure 3.2 (d)). If we know the reward for the predecessor edge $e_p = (x_p, x_i)$, this yields predictions for any direct edges from the predecessor node to the target area. The prediction for unvisited edges is then simply the sum of rewards of the two known edges $r(e_{\mathrm{cur}}) + r(e_p)$.

To ensure that updates are only performed along chains of edges that follow similar trajectories in the state space, we exclude connections that enclose large angles with the currently traversed edge from the prediction. Secondly, we use a similarity measure for weighted updates, giving more weight to predictions that originate from more similar trajectories. A straightforward measure for the similarity of short transitions with nearby starting and target points is the time that is needed for the transition. Local controllers for short connections of similar points will likely follow a similar trajectory if they need the same amount of time. We define the time-similarity weight of two paths with total times t_1 and t_2 as

$$w = \exp\left(-\beta \cdot |\log(t_1) - \log(t_2)|\right)$$

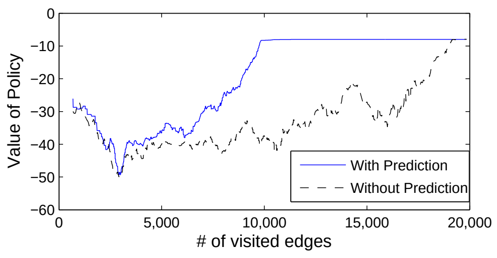


Figure 3.3: Speed-up effect of reward prediction on a static puddle world task with uniformly sampled nodes.

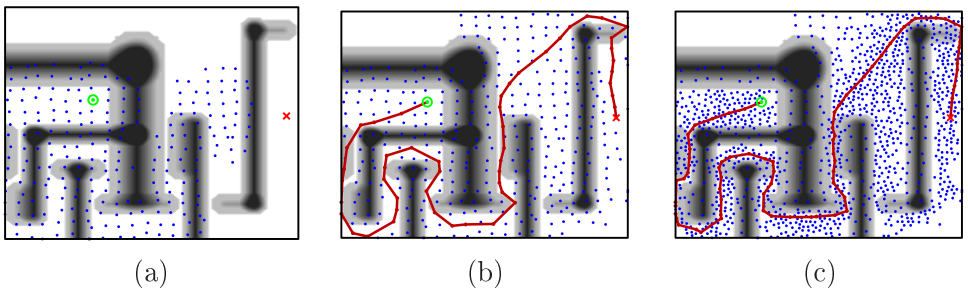


Figure 3.4: Static Puddle World: (a) and (b) shows the graph in the initial exploration phase after 20 and 65 episodes. (c) shows the graph after the refinement phase. The red line indicates the optimal policy to the target.

The absolute logarithm ensures that the weights are proportional to relative, not absolute time differences. For every updated edge e' we store a weight $w_{e'}$ which reflects the confidence of the current estimate. Initial reward estimates $\hat{r}(e')$ come from local controllers, and are assigned small constant initial weights $w_{e'} = w_0 > 0$. Every time an update of an edge is performed we change the reward prediction to the weighted sum of all updates so far, and increase the weight of the edge by the similarity of the alternative route. To preferentially improve reward estimates of low confidence, the exploration scheme may also take the weights into account as an additional factor for action selection.

3.4.1 Predictions for New Edges

Whenever new edges are inserted into the adaptive state graph during the exploration and refinement phases, we basically use the same mechanism as above to estimate their rewards. We search for known 1- or 2-edge paths into the target region of the new edge and perform the updates.

3.4.2 Results of Reward Prediction

To isolate the speed-up effect of reward prediction from the exploration schemes, we learned policies in a static puddle world (see also Section 3.5.1) with 600 uniformly sampled nodes. The rewards of the edges were initialized to estimated time-dependent costs. Figure 3.3 shows that an agent with reward prediction finds the optimal policy after visiting around 50% of the edges that an agent without prediction needs.

3.5 Experiments

In this section we show that our algorithm can solve several continuous control problems that are challenging for standard reinforcement learning techniques. We show that the algorithm requires less actual experience than existing methods and finds more accurate trajectories.

3.5.1 Static Puddle World

The puddle world task is a well-known benchmark for reinforcement learning algorithms in continuous domains. The objective is to navigate from a given starting state to a goal state in a 2-dimensional environment which contains *puddles*, representing regions of negative reward. Every transition inflicts a negative reward proportional to the required time, plus additional penalties for entering a puddle area. The 2-dimensional control action $u = (v_x, v_y)$ corresponds to setting velocities in x and y directions, leading to the simple linear system dynamics $\dot{x} = v_x, \dot{y} = v_y$.

We can safely assume to know this dynamics, but planning a path to the goal state and avoiding the unknown puddles remains a difficult task. Figure 3.4 shows various stages of the exploration process in a maze-like puddle world with multiple puddles. In Figure 3.4(a) it can be observed that the agent directs its initial exploration towards the goal, while avoiding paths through regions of negative reward. Less promising regions like the upper left part are also visited less frequently. After the end of the initial exploration phase (Figure 3.4(b)) the agent knows a coarse path to the goal. A better solution is found after the graph refinement phase, which is illustrated in Figure 3.4(c). The path is almost optimal and avoids all puddles on the way to the goal, even at narrow passages.

Standard function approximation techniques like CMACs and RBFs need several thousands of episodes to converge on this task, and are therefore not considered for comparison. Better results were achieved by Prioritized Sweeping (Moore and Atkeson, 1993), a model-based RL algorithm which discretizes the environment and learns the transition and reward model from its experience. In Figure 3.5 we compare the performance of RL with adaptive state graphs to prioritized sweeping with various discretization densities. We evaluate the performance of the agent by measuring the sum of rewards obtained by its greedy policy at different training times. The training time is the total amount of time the agent has interacted with the environment.

Figure 3.5 shows that the graph-based RL algorithm is faster to achieve reasonable performance than prioritized sweeping with coarse discretization. Our algorithm gradually improves its performance in the graph refinement phase, which starts at approximately 700 seconds. After further training time the graph-based approach slightly outperforms the best policy found by prioritized sweeping on a fine 50×50 grid. The refined graph in the end contains about 1200 nodes, which is less than half the number of states used by prioritized sweeping on the fine grid.

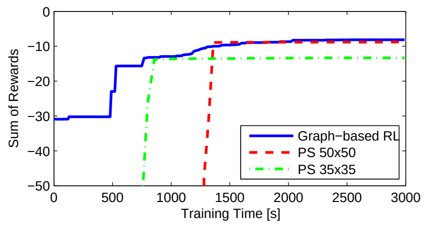


Figure 3.5: Learning performance of RL with adaptive state graphs (max. 1193 nodes) and prioritized sweeping (PS) with different discretization densities on the static puddle world from Figure 3.4. (Average over 5 trials.)

3.5.2 3-Link Arm Reaching Task

In the next task we control a simulated planar 3-link robot arm under static stability constraints in an environment with several obstacles (see Figure 3.6). The links of the robot arm have different weights, and the center of mass (CoM) of the robot needs to be kept inside a finite support polygon. If the CoM leaves a neutral zone of guaranteed stability, the agent receives negative reward that grows quadratically as the CoM approaches the boundary of the support polygon. Under these constraints the trivial solution of rotating the arm around the top left obstacle achieves lower reward than the trajectory that maneuvers the arm through the narrow passage between the obstacles.

The 3-dimensional state space consist of the three joint angles $(\theta_1, \theta_2, \theta_3)$ and the control actions correspond to setting the angular velocities. The approximate model $\hat{f}$ is a simple linear model. The true system dynamics f contains nonlinearities due to obstacles, which are not captured by $\hat{f}$.

The comparison in Figure 3.7 shows that graph-based RL converges much faster to more accurate trajectories than prioritized sweeping with various levels of discretization.

3.5.3 Dynamic Puddle World

We study a dynamic version of the puddle world problem on a simplified environment (see Figure 3.8(a)). The 4-dimensional state space consists of $(x, y, \dot{x}, \dot{y})$, and the control actions correspond to accelerations in x and y direction. The approximate model is still linear but of higher order. In the dynamic case the design of local controllers is more difficult, because positions and velocities are coupled. We first calculate the time required by a bang-bang controller to reach its target for $(x, \dot{x})$ and $(y, \dot{y})$ independently. The controller which reaches its target faster is then slowed down such that all state variables arrive at the target simultaneously.

Figure 3.8(b) shows a comparison of our algorithm to RBF value function approximation. Our approach converges much faster and finds solution trajectories of

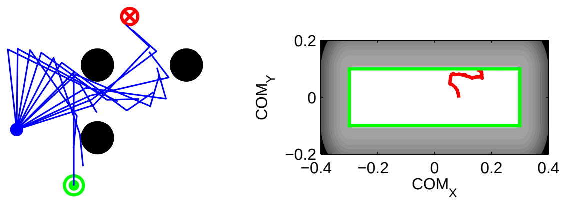


Figure 3.6: Arm reaching task with stability constraints. Left: Solution trajectory found by our algorithm. The agent must reach the goal region (red) from the starting position (green), avoiding the obstacles. Right: The agent receives negative reward if its center of mass (red) leaves the neutral zone (green).

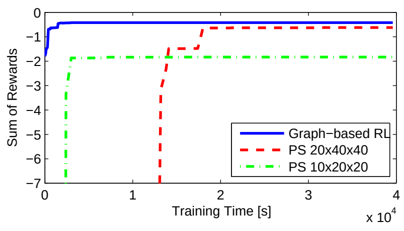


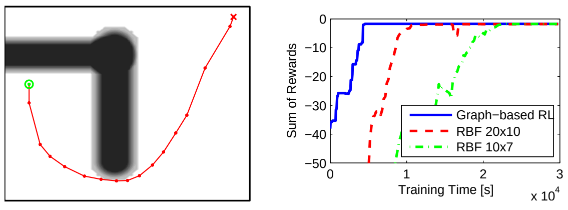
Figure 3.7: Learning performance on the 3-link arm reaching task for RL with adaptive state graphs (max. 2629 nodes) and prioritized sweeping (PS) with different discretization densities. (Average over 5 trials)

similar quality. In contrast to the previous examples, prioritized sweeping did not find satisfactory results in reasonable time. One reason is that on dynamic tasks discretized state signals often violate the Markov property. The other reason is the exponential increase in the number of states with growing dimensionality.

3.6 Conclusion and Future Work

In this paper we introduced a new combination of reinforcement learning and sampling-based planning for control problems with complex reward functions in unknown continuous environments. We use minimal prior knowledge in the form of approximate models and local controllers to increase learning speed and produce continuous control outputs for varying time intervals. Our algorithm builds an adaptive state graph through efficient goal-directed exploration and refines the graph in later stages. A new generalization scheme for reward prediction of unvisited edges increases the performance of the algorithm by avoiding unnecessary exploration. We demonstrated on various movement planning tasks with complex reward functions that RL with adaptive state graphs outperforms standard RL techniques for function approximation.

Future work will extend the approach to non-linear system dynamics and higher dimensional problems. The approach is particularly promising for complicated tasks



(a) (b)

Figure 3.8: (a) Dynamic puddle world environment and solution trajectory. (b) Learning performance on the dynamic puddle world for RL with adaptive state graphs (max. 8106 nodes) and RL with RBF function approximation. (20 resp. 10 RBF centers per position dimension and 10 resp. 7 RBF centers per velocity dimension. Average over 5 trials.)

that can be projected to low dimensional representations, such as balancing humanoid robots using motion primitives (Hauser et al., 2007). Future investigations will also concern strategies to reduce the number of nodes, thereby enabling applications in larger state spaces.

3.7 Acknowledgments

This chapter is based on the paper (Neumann et al., 2007) written by Gerhard Neumann (GN), Michael Pfeiffer (MP) and Wolfgang Maass (WM). GN implemented the graph-based RL algorithm and conducted most of the experiments while MP implemented the reward prediction mechanism. WM significantly improved the paper writting.

Chapter 4

Fitted Q-iteration by Advantage Weighted Regression

Contents

Recently, fitted Q-iteration (FQI) based methods have become more popular due to their increased sample efficiency, a more stable learning process and the higher quality of the resulting policy. However, these methods remain hard to use for continuous action spaces which frequently occur in real-world tasks, e.g., in robotics and other technical applications. The greedy action selection commonly used for the policy improvement step is particularly problematic as it is expensive for continuous actions, can cause an unstable learning process, introduces an optimization bias and results in highly non-smooth policies unsuitable for real-world systems. In this paper, we show that by using a soft-greedy action selection the policy improvement step used in FQI can be simplified to an inexpensive advantage-weighted regression. With this result, we are able to derive a new, computationally efficient FQI algorithm which can even deal with high dimensional action spaces.

4.1 Introduction

Reinforcement Learning (Sutton and Barto, 1998) addresses the problem of how autonomous agents can improve their behavior using their experience. At each time step t the agent can observe its current state $s_t \in \mathcal{X}$ and chooses an appropriate action $a_t \in \mathcal{A}$. Subsequently, the agent gets feedback on the quality of the action, i.e., the reward $r_t = r(s_t, a_t)$, and observes the next state s_{t+1}. The goal of the agent is to maximize the accumulated reward expected in the future. In this paper, we focus on learning policies for continuous, multi-dimensional control problems. Thus the state space $\mathcal{X}$ and action space $\mathcal{A}$ are continuous and multi-dimensional, meaning that discretizations start to become prohibitively expensive.

While discrete-state/action reinforcement learning is a widely studied problem with rigorous convergence proofs, the same does not hold true for continuous states and actions. For continuous state spaces, few convergence guarantees exist and pathological cases of bad performance can be generated easily (Boyan and Moore, 1995). Moreover, many methods cannot be transferred straightforwardly to continuous actions.

Current approaches often circumvent continuous action spaces by focusing on problems where the actor can rely on a discrete set of actions, e.g., when learning a policy for driving a car to a goal in minimum time, an actor only needs three actions: the maximum acceleration when starting, zero acceleration at maximum velocity and maximum throttle down when the goal is sufficiently close for a point landing. While this approach (called bang-bang in traditional control) works for the large class of minimum time control problems, it is also a limited approach as cost functions relevant to the real-world incorporate much more complex constraints, e.g., cost-functions in biological systems often punish the jerkiness of the movement (Viviani and Flash, 1995), the amount of used metabolic energy (Alexander, 1997) or the variance at the end-point (Wolpert, 1998). For physical technical systems, the incorporation of further optimization criteria is of essential importance; just as a minimum time policy is prone to damage the car on the long-run, a similar policy would be highly dangerous for a robot and its environment and the resulting energy-consumption would reduce its autonomy. More complex, action-dependent immediate reward functions require that much larger sets of actions are being employed.

We consider the use of continuous actions for fitted Q-iteration (FQI) based algorithms. FQI is a batch mode reinforcement learning (BMRL) algorithm. The algorithm mantains an estimate of the state-action value function $Q(\mathbf{s}, \mathbf{a})$ and uses the greedy operator $\max_a Q(\mathbf{s}, \mathbf{a})$ on the action space for improving the policy. While this works well for discrete action spaces, the greedy operation is hard to perform for high-dimensional continuous actions. For this reason, the application of fitted Q-iteration based methods is often restricted to low-dimensional action spaces which can be efficiently discretized. In this paper, we show that the use of a stochastic soft-max policy instead of a greedy policy allows us to reduce the policy improvement step used in FQI to a simple advantage-weighted regression. The greedy operation $\max_a Q(\mathbf{s}, \mathbf{a})$ over the actions is replaced by a less harmful greedy operation over the parameter space of the value function. This result allows us to derive a new, computationally efficient algorithm which is based on Locally-Advantage-WEighted Regression (LAWER).

We test our algorithm on three different benchmark tasks, i.e., the pendulum swing-up (Riedmiller, 2005), the acrobot swing-up (Sutton and Barto, 1998) and a dynamic version of the puddle-world (Sutton, 1996) with 2 and 3 dimensions. We show that in spite of the soft-greedy action selection, our algorithm is able to produce high quality policies.

4.2 Fitted Q-Iteration

In fitted Q-iteration (Ernst et al., 2005; Riedmiller, 2005; Antos et al., 2008) (FQI), we assume that all the experience of the agent up to the current time is given in the form $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i >\}_{1 \le i \le N}$. The task of the learning algorithm is to estimate an optimal control policy from this historical data. FQI approximates the state-action value function $Q(\mathbf{s}, \mathbf{a})$ by iteratively using supervised regression techniques. New target values for the regression are generated by

$$\tilde{Q}_{k+1}(i) \quad = \quad r_i + \gamma V_k(\mathbf{s}'_i) = r_i + \gamma \max_{\mathbf{a}'} Q_k(\mathbf{s}'_i, \mathbf{a}'). \tag{4.1}$$

The regression problem for finding the function Q_{k+1} is defined by the list of data-point pairs D_k and the regression procedure Regress

$$D_k(Q_k) = \left\{ \left[(\mathbf{s}_i, \mathbf{a}_i), \tilde{Q}_{k+1}(i) \right]_{1 \le i \le N} \right\}, \quad Q_{k+1} = \mathrm{Regress}(D_k(Q_k)) \tag{4.2}$$

FQI can be viewed as approximate value iteration with state-action value functions (Antos et al., 2008). Previous experiments show that function approximators such as neural networks (Riedmiller, 2005), radial basis function networks (Ernst et al., 2005), CMAC (Timmer and Riedmiller, 2007) and regression trees (Ernst et al., 2005) can be employed in this context. In (Antos et al., 2008), performance bounds for the value function approximation are given for a wide range of function approximators. The performance bounds also hold true for continuous action spaces, but only in the case of an actor-critic variant of FQI. Unfortunately, to our knowledge, no experiments with this variant exist in the literature. Additionally, it is not clear how to apply this actor-critic variant efficiently for nonparametric function approximators.

FQI has proven to outperform classical online RL methods in many applications (Ernst et al., 2005). Nevertheless, FQI relies on the greedy action selection in Equation (4.1). Thus, the algorithm frequently requires a discrete set of actions and generalization to continuous actions is not straightforward. Using the greedy operator for continuous action spaces is a hard problem by itself as the use of expensive optimization methods is needed for high dimensional actions. Moreover the returned values of the greedy operator often result in an optimization bias causing an unstable learning process, including oscillations and divergence (Peters and Schaal, 2007a). For a comparison with our algorithm, we use the Cross-Entropy (CE) optimization method (de Boer et al., 2005) to find the maximum Q-values. In our implementation, we maintain a Gaussian distribution for the belief of the optimal action. We sample n_{CE} actions from this distribution. Then, the best $e_{CE} < n_{CE}$ actions (with the highest Q-values) are used to update the parameters of this distribution. The whole process is repeated for k_{CE} iterations, starting with a uniformly distributed set of sample actions.

FQI is inherently an offline method - given historical data, the algorithm estimates the optimal policy. However, FQI can also be used for online learning. After the FQI algorithm is finished, new episodes can be collected with the currently best inferred policy and the FQI algorithm is restarted.

4.3 Fitted Q-Iteration by Advantage Weighted Regression

A different method for policy updates in continuous action spaces is reinforcement learning by reward-weighted regression (Peters and Schaal, 2007b). As shown by the authors, the action selection problem in the immediate reward RL setting with continuous actions can be formulated as expectation-maximization (EM) based algorithm and, subsequently, reduced to a reward-weighted regression. The weighted regression can be applied with ease to high-dimensional action spaces; no greedy operation in the action space is needed. While we do not directly follow the work in (Peters and Schaal, 2007b), we follow the general idea.

4.3.1 Weighted regression for value estimation

In this section we consider the task of estimating the value function V of a stochastic policy $\pi(\cdot|\mathbf{s})$ when the state-action value function Q is already given. The value function can be calculated by $V(\mathbf{s}) = \int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})Q(\mathbf{s},\mathbf{a})d\mathbf{a}$. Yet, the integral over the action space is hard to perform for continuous actions. However, we will show how we can approximate the value function without the evaluation of this integral. Consider the quadratic error function

$$\begin{aligned}\text{Error}(\hat{V}) &= \int_{\mathbf{s}} \mu(\mathbf{s}) \Big(\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) Q(\mathbf{s},\mathbf{a}) d\mathbf{a} - \hat{V}(\mathbf{s}) \Big)^2 d\mathbf{s} &(4.3)\\ &= \int_{\mathbf{s}} \mu(\mathbf{s}) \left(\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left(Q(\mathbf{s},\mathbf{a}) - \hat{V}(\mathbf{s}) \right) d\mathbf{a} \right)^2 d\mathbf{s}, &(4.4)\end{aligned}$$

which is used to find an approximation $\hat{V}$ of the value function. $\mu(\mathbf{s})$ denotes the state distribution when following policy $\pi(\cdot|\mathbf{a})$. Since the squared function is convex we can use Jensens inequality for probability density functions to derive an upper bound of Equation (4.4)

$$\text{Error}(\hat{V}) \leq \int_{\mathbf{s}} \mu(\mathbf{s}) \int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left(Q(\mathbf{s},\mathbf{a}) - \hat{V}(\mathbf{s}) \right)^2 d\mathbf{a} d\mathbf{s} = \text{Error}_B(\hat{V}). \quad (4.5)$$

The solution $\hat{V}^*$ for minimizing the upper bound $\text{Error}_B(\hat{V})$ is the same as for the original error function $\text{Error}(\hat{V})$.

Proof. To see this, we compute the square and replace the term $\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})Q(\mathbf{s},\mathbf{a})d\mathbf{a}$ by the value function $V(\mathbf{s})$. This is done for the error function $\text{Error}(\hat{V})$ and for the upper bound $\text{Error}_B(\hat{V})$.

$$\begin{aligned}\text{Error}(\hat{V}) &= \int_{\mathbf{s}} \mu(\mathbf{s}) \Big(V(\mathbf{s}) - \hat{V}(\mathbf{s}) \Big)^2 d\mathbf{s} &(4.6)\\ &= \int_{\mathbf{s}} \mu(\mathbf{s}) \left(V(\mathbf{s})^2 - 2V(\mathbf{s})\hat{V}(\mathbf{s}) + \hat{V}(\mathbf{s})^2 \right) d\mathbf{s} &(4.7)\end{aligned}$$

$$\begin{aligned}\text{Error}_B(\hat{V}) &= \int_{\mathbf{s}} \mu(\mathbf{s}) \int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \left(Q(\mathbf{s},\mathbf{a})^2 - 2Q(\mathbf{s},\mathbf{a})\hat{V}(\mathbf{s}) + \hat{V}(\mathbf{s})^2 \right) d\mathbf{a} d\mathbf{s} \quad (4.8)\\ &= \int_{\mathbf{s}} \mu(\mathbf{s}) \left(\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) Q(\mathbf{s},\mathbf{a})^2 d\mathbf{a} - 2V(\mathbf{s})\hat{V}(\mathbf{s}) + \hat{V}(\mathbf{s})^2 \right) d\mathbf{s} \quad (4.9)\end{aligned}$$

Both error functions are the same except for an additive constant which does not depend on $\hat{V}$. □

In difference to the original error function, the upper bound Error_B can be approximated straightforwardly by samples $\{(\mathbf{s}_i, \mathbf{a}_i), Q(\mathbf{s}_i, \mathbf{a}_i)\}_{1 \le i \le N}$ gained by following some behavior policy $\pi_b(\cdot|\mathbf{s})$.

$$\text{Error}_B(\hat{V}) \approx \sum_{i=1}^{N} \frac{\mu(\mathbf{s})\pi(\mathbf{a}_i|\mathbf{s}_i)}{\mu_b(\mathbf{s}_i)\pi_b(\mathbf{a}_i|\mathbf{s}_i)} \left(Q(\mathbf{s}_i, \mathbf{a}_i) - \hat{V}(\mathbf{s}_i) \right)^2, \quad (4.10)$$

$\mu_b(\mathbf{s})$ defines the state distribution when following the behavior policy π_b. The term $1/(\mu_b(\mathbf{s}_i)\pi_b(\mathbf{s}_i, \mathbf{a}_i))$ ensures that we do not give more weight on states and actions preferred by π_b. This is a well known method in importance sampling. In order to keep our algorithm tractable, the factors $\pi_b(\mathbf{a}_i|\mathbf{s}_i)$, $\mu_b(\mathbf{s}_i)$ and $\mu(\mathbf{s}_i)$ will all be set to $1/N$. The minimization of Equation (4.10) defines a weighted regression problem which is given by the dataset D_V, the weighting U and the weighted regression procedure WeightedRegress

$$D_V = \left\{ [(\mathbf{s}_i, \mathbf{a}_i), Q(\mathbf{s}_i, \mathbf{a}_i)]_{1 \le i \le N} \right\}, U = \{[\pi(\mathbf{a}_i|\mathbf{s}_i)]_{1 \le i \le N}\}, \quad (4.11)$$

$$\hat{V} = \text{WeightedRegress}(D_V, U) \quad (4.12)$$

The result shows that in order to approximate the value function $V(\mathbf{s})$, we do not need to carry out the expensive integration over the action space for each state $\mathbf{s}_i$. It is sufficient to know the Q-values at a finite set of state-action pairs.

4.3.2 Soft-greedy policy improvement

We use a soft-max policy (Sutton and Barto, 1998) in the policy improvement step of the FQI algorithm. Our soft-max policy $\pi_1(\mathbf{a}|\mathbf{s})$ is based on the advantage function $A(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a}) - V(\mathbf{s})$. We additionally assume the knowledge of the mean $m_A(\mathbf{s})$ and the standard deviation of $\sigma_A(\mathbf{s})$ of the advantage function at state $\mathbf{s}$. These quantities can be estimated locally or approximated by additional regressions. The policy $\pi_1(\mathbf{a}|\mathbf{s})$ is defined as

$$\pi_1(\mathbf{a}|\mathbf{s}) = \frac{\exp(\tau \bar{A}(\mathbf{s}, \mathbf{a}))}{\int_{\mathbf{a}} \exp(\tau \bar{A}(\mathbf{s}, \mathbf{a})) d\mathbf{a}}, \quad \bar{A}(\mathbf{s}, \mathbf{a}) = \frac{A(\mathbf{s},\mathbf{a}) - m_A(\mathbf{s})}{\sigma_A(\mathbf{s})}. \quad (4.13)$$

τ controls the greediness of the policy. If we assume that the advantages $A(\mathbf{s}, \mathbf{a})$ are distributed with $\mathcal{N}(A(\mathbf{s}, \mathbf{a})|m_A(\mathbf{s}), \sigma_A^2(\mathbf{s}))$, all normalized advantage values $\bar{A}(\mathbf{s}, \mathbf{a})$ have the same distribution. Thus, the denominator of π_1 is constant for all states and we can use the term $\exp(\tau \bar{A}(\mathbf{s}, \mathbf{a})) \propto \pi_1(\mathbf{a}|\mathbf{s})$ directly as weighting for the regression defined in Equation (4.12). The resulting approximated value function $\hat{V}(\mathbf{s})$ is used

Algorithm 1: FQI with Advantage Weighted Regression

```
Input: $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_i' >\}_{1 \le i \le N}$, $\tau$ and $L$ (Number of Iterations)
Initialize $\hat{V}_0(\mathbf{s}) = 0$.
for $k = 0$ to $L - 1$ do
    $D_k(\hat{V}_k) = \left\{ \left[ (\mathbf{s}_i, \mathbf{a}_i), r_i + \gamma \hat{V}_k(\mathbf{s}_i') \right]_{1 \le i \le N} \right\}$
    $Q_{k+1} = \text{Regress}(D_k(\hat{V}_k))$
    $A(i) = Q_{k+1}(\mathbf{s}_i, \mathbf{a}_i) - \hat{V}_k(\mathbf{s}_i)$
    Estimate $m_A(\mathbf{s}_i)$ and $\sigma_A(\mathbf{s}_i)$ for $1 \le i \le N$
    $U = \{[\exp(\tau(A(i) - m_A(\mathbf{s}_i))/\sigma_A(\mathbf{s}_i)]_{i \le i \le N}\}$
    $\hat{V}_{k+1} = \text{WeightedRegress}(D_k(\hat{V}_k), U)$
end for
```

to replace the greedy operator $V(\mathbf{s}_i') = \max_{\mathbf{a}'} Q(\mathbf{s}_i', \mathbf{a}')$ in the FQI algorithm. The FQI by Advantage Weighted Regression (AWR) algorithm is given in Algorithm 1. As we can see, the Q-function Q_k is only queried once for each step in the history H. Furthermore only already seen state action pairs $(\mathbf{s}_i, \mathbf{a}_i)$ are used for this query.

After the FQI algorithm is finished we still need to determine a policy for subsequent data collection. The policy can be obtained in the same way as for reward-weighted regression (Peters and Schaal, 2007b), only the advantage is used instead of the reward for the weighting - thus, we are optimizing the long term costs instead of the immediate one.

4.4 Locally-Advantage-WEighted Regression (LAWER)

Based on the FQI by AWR algorithm, we propose a new, computationally efficient fitted Q-iteration algorithm which uses Locally Weighted Regression (LWR, (Atkeson et al., 1997)) as function approximator. Similar to kernel based methods, our algorithm needs to be able to calculate the similarity $w_i(\mathbf{s})$ between a state $\mathbf{s}_i$ in the dataset H and state $\mathbf{s}$. To simplify the notation, we will denote $w_i(\mathbf{s}_j)$ as w_{ij} for all $\mathbf{s}_j \in H$. $w_i(\mathbf{s})$ is calculated by a Gaussian kernel $w_i(\mathbf{s}) = \exp(-(\mathbf{s}_i - \mathbf{s})^T \mathbf{D}(\mathbf{s}_i - \mathbf{s}))$. The diagonal matrix $\mathbf{D}$ determines the bandwidth of the kernel. Additionally, our algorithm also needs a similarity measure w_{ij}^a between two actions $\mathbf{a}_i$ and $\mathbf{a}_j$. Again w_{ij}^a can be calculated by a Gaussian kernel $w_{ij}^a = \exp(-(\mathbf{a}_i - \mathbf{a}_j)^T \mathbf{D}^a (\mathbf{a}_i - \mathbf{a}_j))$.

Using the state similarity w_{ij}, we can estimate the mean and the standard deviation of the advantage function for each state $\mathbf{s}_i$

$$m_A(\mathbf{s}_i) = \frac{\sum_j w_{ij} A(j)}{\sum_j w_{ij}}, \quad \sigma_A^2(\mathbf{s}_i) = \frac{\sum_j w_{ij}(A(j) - m_A(\mathbf{s}_j))^2}{\sum_j w_{ij}}. \tag{4.14}$$

4.4.1 Approximating the value functions

For the approximation of the Q-function, we use Locally Weighted Regression (Atkeson et al., 1997). The Q-function is therefore given by:

$$Q_{k+1}(\mathbf{s}, \mathbf{a}) = \tilde{\mathbf{s}}_{\mathbf{A}}(\mathbf{S_A}^T\mathbf{W}\mathbf{S_A})^{-1}\mathbf{S_A}^T\mathbf{W}\mathbf{Q}_{k+1} \tag{4.15}$$

where $\tilde{\mathbf{s}}_{\mathbf{A}} = [1, \mathbf{s}^T, \mathbf{a}^T]^T$, $\mathbf{S_A} = [\tilde{\mathbf{s}}_{\mathbf{A}}(1), \tilde{\mathbf{s}}_{\mathbf{A}}(2), ..., \tilde{\mathbf{s}}_{\mathbf{A}}(N)]^T$ is the state-action matrix, $\mathbf{W} = \text{diag}(w_i(\mathbf{s})w_i^a(\mathbf{a}))$ is the local weighting matrix consisting of state and action similarities, and $\mathbf{Q}_{k+1} = [\tilde{Q}_{k+1}(1), \tilde{Q}_{k+1}(2), \dots, \tilde{Q}_{k+1}(N)]^T$ is the vector of the Q-values (see Equation (4.1).

For approximating the V-function we can multiplicatively combine the advantage-based weighting $u_i = \exp(\tau \bar{A}(\mathbf{s}_i, \mathbf{a}_i))$ and the state similarity weights $w_i(\mathbf{s})$. The value $V_{k+1}(\mathbf{s})$ is given by [1]:

$$V_{k+1}(\mathbf{s}) = \tilde{\mathbf{s}}(\mathbf{S}^T\mathbf{U}\mathbf{S})^{-1}\mathbf{S}^T\mathbf{U}\mathbf{Q}_{k+1}, \tag{4.16}$$

where $\tilde{\mathbf{s}} = [1, \mathbf{s}^T]^T$, $\mathbf{S} = [\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2, ..., \tilde{\mathbf{s}}_N]^T$ is the state matrix and $\mathbf{U} = \text{diag}(w_i(\mathbf{s})u_i)$ is the weight matrix. We bound the estimate of $\hat{V}_{k+1}(\mathbf{s})$ by $\max_{i|w_i(\mathbf{s})>0.001} Q_{k+1}(i)$ in order to prevent the local regression from adding a positive bias which might cause divergence of the value iteration.

A problem with nonparametric value function approximators is their strongly increasing computational complexity with an increasing number of data points. A simple solution to avoid this problem is to introduce a local forgetting mechanism. Whenever parts of the state space are oversampled, old examples in this area are removed from the dataset.

4.4.2 Approximating the policy

Similar to reward-weighted regression (Peters and Schaal, 2007b), we use a stochastic policy $\pi(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{a}|\mu(\mathbf{s}), \text{diag}(\sigma^2(\mathbf{s})))$ with Gaussian exploration as approximation of the optimal policy. The mean $\mu(\mathbf{s})$ and the variance $\sigma^2(\mathbf{s})$ are given by

$$\mu(\mathbf{s}) = \tilde{\mathbf{s}}(\mathbf{S}^T\mathbf{U}\mathbf{S})^{-1}\mathbf{S}^T\mathbf{U}\mathbf{A}, \quad \sigma^2(\mathbf{s}) = \frac{\sigma^2_{\text{init}}\alpha_0 + \sum_i w_i(\mathbf{s})u_i(\mathbf{a}_i - \mu(\mathbf{s}_i))^2}{\alpha_0 + \sum_i w_i(\mathbf{s})u_i}, \tag{4.17}$$

where $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_N]^T$ denotes the action matrix. The variance σ^2 automatically adapts the exploration of the policy to the uncertainty of the optimal action. With σ^2_{init} and α_0 we can set the initial exploration of the policy. σ_{init} is always set to the bandwidth of the action space. α_0 sets the weight of the initial variance in comparison to the variance coming from the data, α_0 is set to 3 for all experiments.

[1] In practice, ridge regression $V_{k+1}(\mathbf{s}) = \tilde{\mathbf{s}}(\mathbf{S}^T\mathbf{W}\mathbf{S} + \sigma\mathbf{I})^{-1}\mathbf{S}^T\mathbf{W}\mathbf{Q}_{k+1}$ is used to avoid numerical instabilities in the regression.

4.5 Evaluations

We evaluated the LAWER algorithm on three benchmark tasks, the pendulum swing up task, the acrobot swing up task and a dynamic version of the puddle-world (i.e., augmenting the puddle-world by velocities, inertia, etc.) with 2 and 3 dimensions. We compare our algorithm to tree-based FQI (Ernst et al., 2005) (CE-Tree), neural FQI (Riedmiller, 2005) (CE-Net) and LWR-based FQI (CE-LWR) which all use the Cross-Entropy (CE) optimization to find the maximum Q-values. For the CE optimization we used $n_{CE} = 10$ samples for one dimensional, $n_{CE} = 25$ samples for 2-dimensional and $n_{CE} = 64$ for 3-dimensional control variables. e_{CE} was always set to $0.3n_{CE}$ and we used $k_{CE} = 3$ iterations. To enforce exploration when collecting new data, a Gaussian noise of $\varepsilon = \mathcal{N}(0, 1.0)$ was added to the CE-based policy. For the tree-based algorithm, an ensemble of $M = 20$ trees was used, K was set to the number of state and action variables and n_{min} was set to 2 (see (Ernst et al., 2005)). For the CE-Net algorithm we used a neural network with 2 hidden layers and 10 neurons per layer and trained the network with the algorithm proposed in (Riedmiller, 2005) for 600 epochs. For all experiments, a discount factor of $\gamma = 0.99$ was used. The immediate reward function was quadratic in the distance to the goal position $\mathbf{s}_G$ and in the applied torque/force $r = -c_1(\mathbf{s} - \mathbf{s}_G)^2 - c_2\mathbf{a}^2$. For evaluating the learning process, the exploration-free (i.e., $\sigma(\mathbf{s}) = 0$, $\varepsilon = 0$) performance of the policy was evaluated after each data-collection/FQI cycle. This was done by determining the accumulated reward during an episode starting from the specified initial position. All errorbars represent a 95% confidence interval.

4.5.1 Pendulum swing-up task

In this task, a pendulum needs to be swung up from the position at the bottom to the top position (Riedmiller, 2005). The state space consists of the angular deviation θ from the top position and the angular velocity $\dot{\theta}$ of the pendulum. The system dynamics are given by $0.5ml^2\ddot{\theta} = mg\sin(\theta) + u$, the torque of the motor u was limited to $[-5N, 5N]$. The mass was set to $m = 1$kg and length of the link to 1m. The time step was set to $0.05s$. Two experiments with different torque punishments $c_2 = 0.005$ and $c_2 = 0.025$ were performed.

We used $L = 150$ iterations. The matrices $\mathbf{D}$ and $\mathbf{D}_A$ were set to $\mathbf{D} = \text{diag}(30, 3)$ and $\mathbf{D}_A = \text{diag}(2)$. In the data collection phase, 5 episodes with 150 steps were collected starting from the bottom position and 5 episodes starting from a random position.

A comparison of the LAWER algorithm to CE-based algorithms for $c_2 = 0.005$ is shown in Figure 4.1(a) and for $c_2 = 0.025$ in Figure 4.1(b). Our algorithm shows a comparable performance to the tree-based FQI algorithm while being computationally much more efficient. All other CE-based FQI algorithms show a slightly decreased performance. In Figure 4.1(c) and (d) we can see typical examples of learned torque trajectories when starting from the bottom position for the LAWER, the CE-Tree and the CE-LWR algorithm. In Figure 4.1(c) the trajectories are shown for $c_2 = 0.005$ and in Figure 4.1(d) for $c_2 = 0.025$. All algorithms were able to discover a fast solution with 1 swing-up for the first setting and a more energy-efficient

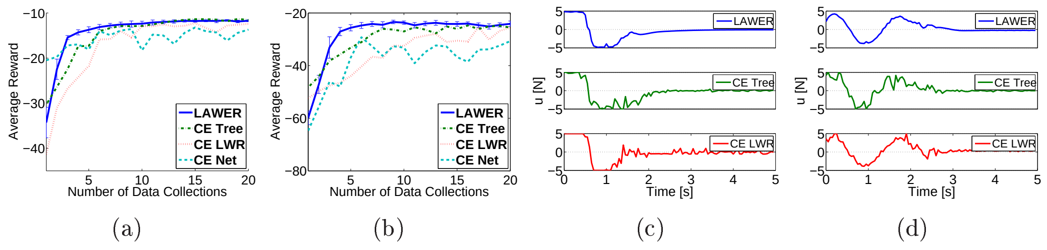


Figure 4.1: (a) Evaluation of LAWER and CE-based FQI algorithms on the pendulum swing-up task for $c_2 = 0.005$. The plots are averaged over 10 trials. (b) The same evaluation for $c_2 = 0.025$. (c) Learned torque trajectories for $c_2 = 0.005$. (d) Learned torque trajectories for $c_2 = 0.025$.

solution with 2 swing-ups for the second setting. Still, there are qualitative differences in the trajectories. Due to the advantage-weighted regression, LAWER was able to produce very smooth trajectories while the trajectories found by the CE-based methods look more jerky. In Figure 4.2(a) we can see the influence of the parameter τ on the performance of the LAWER algorithm. The algorithm works for a large range of τ values.

4.5.2 Acrobot swing-up task

In order to asses the performance of LAWER on a complex highly non-linear control task, we used the acrobot (for a description of the system, see (Sutton and Barto, 1998)). The torque was limited to $[-5N, 5N]$. Both masses were set to 1kg and both lengths of the links to 0.5m. A time step of $0.1s$ was used. $L = 100$ iterations were used for the FQI algorithms. In the data-collection phase the agent could observe 25 episodes starting from the bottom position and 25 starting from a random position. Each episode had 100 steps. The matrices $\mathbf{D}$ and $\mathbf{D}_A$ were set to $\mathbf{D} = \text{diag}(20, 23.6, 10, 10.5)$ and $\mathbf{D}_A = \text{diag}(2)$. The comparison of the LAWER and the CE-Tree algorithm is shown in Figure 4.2(a). Due to the adaptive state discretization, the tree-based algorithm is able to learn faster, but in the end, the LAWER algorithm is able to produce policies of higher quality than the tree-based algorithm.

4.5.3 Dynamic puddle-world

In the puddle-world task (Sutton, 1996), the agent has to find a way to a predefined goal area in a continuous-valued maze world (see Figure 4.3(a)). The agent gets negative reward when going through puddles. In difference to the standard puddle-world setting where the agent has a 2-dimensional state space (the x and y position), we use a more demanding setting. We have created a dynamic version of the puddle-world where the agent can set a force accelerating a k-dimensional point mass ($m =$ 1kg). This was done for $k = 2$ and $k = 3$ dimensions. The puddle-world illustrates the scalability of the algorithms to multidimensional continuous action spaces (2 respectively 3 dimensional). The positions were limited to $[0, 1]$ and the velocities to $[-1, 1]$. The maximum force that could be applied in one direction was restricted

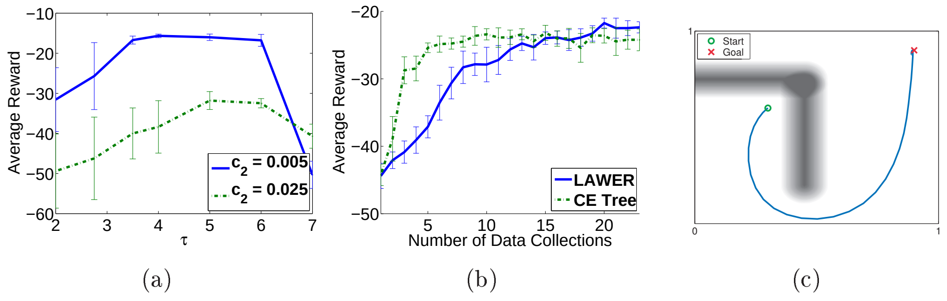


Figure 4.2: (a) Evaluation of the average reward gained over a whole learning trial on the pendulum swing-up task for different settings of τ (b) Comparison of the LAWER and the CE-Tree algorithm on the acrobot swing-up task (c) Setting of the 2-dimensional dynamic puddle-world.

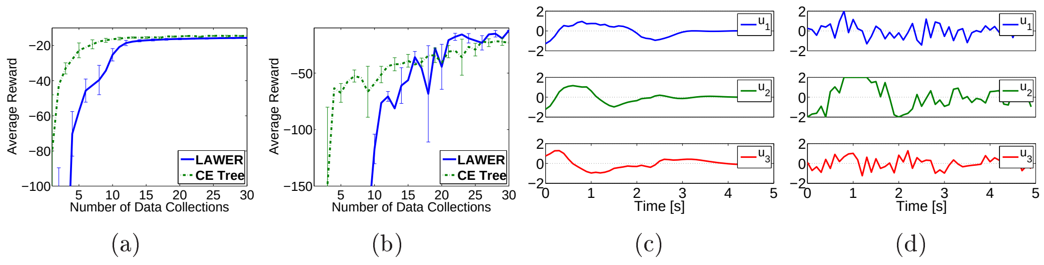


Figure 4.3: (a) Comparison of the CE-Tree and the LAWER algorithm for the 2-dimensional dynamic puddle-world. (b) Comparison of the CE-Tree and the LAWER algorithm for the 3-dimensional dynamic puddle-world. (c) Torque trajectories for the 3-dimensional puddle world learned with the LAWER algorithm. (d) Torque trajectories learned with the CE-Tree algorithm.

to $2N$ and the time step was set to $0.1s$. The setting of the 2-dimensional puddle-world can be seen in Figure 4.2(c). Whenever the agent was about to leave the predefined area, the velocities were set to zero and an additional reward of -5 was given. We compared the LAWER with the CE-Tree algorithm. $L = 50$ iterations were used. The matrices $\mathbf{D}$ and $\mathbf{D}_A$ were set to $\mathbf{D} = \text{diag}(10, 10, 2.5, 2.5)$ and $\mathbf{D}_A = \text{diag}(2.5, 2.5)$ for the 2-dimensional and to $\mathbf{D} = \text{diag}(8, 8, 8, 2, 2, 2)$ and $\mathbf{D}_A = \text{diag}(1, 1, 1)$ for the 3-dimensional puddle-world. In the data collection phase the agent could observe 20 episodes with 50 steps starting from the predefined initial position and 20 episodes starting from a random position.

In Figure 4.3(a), we can see the comparison of the CE-Tree and the LAWER algorithm for the 2-dimensional puddle-world and in Figure 4.3(b) for the 3-dimensional puddle-world. The results show that the tree-based algorithm has an advantage in the beginning of the learning process. However, the CE-Tree algorithm has problems finding a good policy in the 3-dimensional action-space, while the LAWER algorithm still performs well in this setting. This can be seen clearly in the comparison of the learned force trajectories which are shown in Figure 4.3(c) for the LAWER algorithm and in Figure 4.3(d) for the CE-Tree algorithm. The trajectories for the CE-Tree algorithm are very jerky and almost random for the first and third dimension of the control variable, whereas the trajectories found by the LAWER algorithm look very smooth and goal directed.

4.6 Conclusion and future work

In this paper, we focused on solving RL problems with continuous action spaces with fitted Q-iteration based algorithms. The computational complexity of the max operator $\max_a Q(\mathbf{s}, \mathbf{a})$ often makes FQI algorithms intractable for high dimensional continuous action spaces. We proposed a new method which circumvents the max operator by the use of a stochastic soft-max policy that allows us to reduce the policy improvement step $V(\mathbf{s}) = \max_a Q(\mathbf{s}, \mathbf{a})$ to a weighted regression problem. Based on this result, we can derive the LAWER algorithm, a new, computationally efficient FQI algorithm based on LWR.

Experiments have shown that the LAWER algorithm is able to produce high quality smooth policies, even for high dimensional action spaces where the use of expensive optimization methods for calculating $\max_a Q(\mathbf{s}, \mathbf{a})$ becomes problematic and only quite suboptimal policies are found. Moreover, the computational costs of using continuous actions for standard FQI are daunting. The LAWER algorithm needed on average 2780s for the pendulum, 17600s for the acrobot, 13700s for the 2D-puddle-world and 24200s for the 3D-puddle world benchmark task. The CE-Tree algorithm needed on average 59900s, 201900s, 134400s and 212000s, which is an order of magnitude slower than the LAWER algorithm. The CE-Net and CE-LWR algorithm showed comparable running times as the CE-Tree algorithm. A lot of work has been spent to optimize the implementations of the algorithms. The simulations were run on a P4 Xeon with 3.2 gigahertz.

Still, in comparison to the tree-based FQI approach, our algorithm has handicaps when dealing with high dimensional state spaces. The distance kernel matrices have to be chosen appropriately by the user. Additionally, the uniform distance measure throughout the state space is not adequate for many complex control tasks and might degrade the performance. Future research will concentrate on combining the AWR approach with the regression trees presented in (Ernst et al., 2005).

4.7 Acknowledgments

This chapter is based on the paper (Neumann and Peters, 2009) written by Gerhard Neumann (GN) and Jan Peters (JP). GN implemented the Advantage-Weighted Regression algorithms and conducted the experiments while JP provided the basic ideas and guidance for this paper.

Part II

Movement Representations

CHAPTER 5

Introduction

Movement representations are frequently used for motor skill learning (Kober et al., 2008). Instead of directly learning the desired trajectory, they represent a lower dimensional description of the movement which is supposed to facilitate learning of movement skills. Many types of movement representations are also denoted as movement primitive. The key idea of the term 'primitive' is that several of these elementary movements can be combined sequentially or also simultaneously in time.

There are many different approaches how to encode movements with a lower number of parameters, ranging from purely spatial (d'Avella and Bizzi, 2005) to temporal (Schaal et al., 2003, 2007) representations. We will denote $\mathbf{w}$ as the parameter vector of the movement primitive.

In this thesis I will first give an overview over relevant methods. I will also briefly discuss how movement primitives can be combined sequentially and simultaneously in time. Subsequently I will present 3 new approaches which were part of my work during my PhD. In Chapter 6, I will introduce a spatial movement representation approach for balancing control of a humanoid robot (Hauser et al., 2011). In Chapter 7, I will present a new movement representation called motion templates. Motion templates are the first representation which can be combined sequentially in time by the use of reinforcement learning. Finally, in Chapter 8 I will present a primitive which is based on inherent planning. As we use planning already at the level of the primitive, abstract features or goals of the movement can be used as parameter representation. We will show that this can significantly facilitate learning of movement skills.

5.1 Spatial Primitives

Spatial representations use a K dimensional manifold to represent the D-dimensional action space ($K \ll D$) but do not encode any temporal coherence of the movement. Thus, they do not directly specify a policy which can be used to perform the motion.

The most prominent representative of spatial representations is the *synchronous muscle synergies* approach (d'Avella and Bizzi, 2005) which have been developed in the context of explaining biological muscle activation data. The key idea is to use a K-dimensional linear basis for the action space, where $K \ll D$. Each basis vector represents a single synergy. The control vector $\mathbf{a}(t)$ is then represented by the muscle synergy matrix $\mathbf{M}$ and the synergy coefficients, given by $\mathbf{c}(t)$, i.e.

$$\mathbf{a}(t) = \mathbf{M}\mathbf{c}(t).$$

Each column vector of $\mathbf{M}$ represents a single synergy. The synchronous muscle synergies have been used to explain muscle activation patterns in frogs (d'Avella and Bizzi, 2005). The authors could show that several muscle synergies were shared for different behaviors, rendering this approach also attractive for robot control and transfer learning.

In Chapter 6 we present a similar approach based on kinematic synergies which we have applied to a humanoid robot. We use a lower dimensional manifold in the joint space of a humanoid robot to counter-balance unknown perturbations. This is one of the first application of the idea of using synergies for robot control.

Typically the synergies are extracted from experimental data (d'Avella and Bizzi, 2005) or, as in our case, constructed by the inverse kinematic model of the robot. Learning such lower dimensional spatial basis of an high-dimensional redundant action space from interaction with an environment and reinforcement is still an open problem.

5.2 Temporal Primitives

Temporal representations explicitly encode the temporal pattern of the movement. The high-dimensional state information is usually absorbed by a scalar phase or time variable t. The common approach for temporal representations is to calculate a desired trajectory $\langle \mathbf{y}(t; \mathbf{w}), \dot{\mathbf{y}}(t; \mathbf{w}) \rangle$. As we can see, the desired position and velocity of the robot only depends on the duration t of the movement. The real state (current position $\mathbf{q}_t$ and velocity $\dot{\mathbf{q}}_t$) is not used for trajectory generation.

The current state $\mathbf{s}_t = [\mathbf{q}_t, \dot{\mathbf{q}}_t]$ is only used for feedback trajectory-tracking, e.g linear PD-controllers or inverse dynamics control (Peters et al., 2008) (see Section 5.2.6). Both, the feedback controller and the desired trajectory can be parametrized, we will subsume these parameters into the parameter vector $\mathbf{w}$ of the primitive. As the desired trajectory always depends on $\mathbf{w}$ we will write $\mathbf{y}_t$ instead of $\mathbf{y}(t; \mathbf{w})$ to simplify the notation.

Temporal representations can only be used in episodic setups, i.e. we always use the same initial conditions (i.e. state of the robot and its environment) for the movement and the movement ends after a certain amount of time. Because only the duration of the movement is used as information for trajectory generation, we would have to use different primitive parameters for different initial conditions. Thus, these approaches are inherently local. While this restriction renders such approaches less powerful than global representations, temporal representations typically need fewer parameters in comparison to global representations and thus, learning is also considerably simplified. Many of the most impressive robot applications like ball-in-the-cup (Kober et al., 2008), baseball padding (Peters and Schaal, 2006), walking (Nakanishi et al., 2004) or complex balancing movements (Neumann, 2011) have been implemented with temporal primitives.

The generation of the trajectory for these approaches is often an offline process and does not incorporate knowledge of the system dynamics, proprioceptive or other sensory feedback. Because the trajectory itself is created without any knowledge of the system model, the desired trajectory might not be applicable, and thus, the real trajectory of the robot might differ considerably from the specified trajectory.

Many types of temporal movement primitives can be found in the literature, including Dynamic Movement Primitives (DMPs, (Schaal et al., 2007)), time-varying muscle synergies (Bizzi et al., 2008), splines (Kolter and Ng, 2009b), and motion templates (Neumann et al., 2009). We will now briefly review all these methods and discuss the motion templates approach in more detail as we introduced this approach used in the papers (Neumann et al., 2009) and (Neumann, 2011) which are part of this thesis.

In Chapter 8 I will also present unpublished work which proposes a new type of a temporal representation which is based on inherent probabilistic planning.

5.2.1 Dynamic Movement Primitives

The most prominent representation for movement primitives used in robot control are the Dynamic Movement Primitives (DMP) (Schaal et al., 2003). DMPs generate multi-dimensional trajectories by the use of non-linear differential equations. The basic idea is to a use for each degree-of-freedom (DoF) of the robot a globally stable, linear dynamical system which is modulated by a learnable non-linear function f :

$$\tau\dot{z} \;=\; \alpha_z(\beta_z(g-y)-z)+f, \tau\dot{y}=z,$$

where the desired final position of the joint is denoted by g. The variables y and $\dot{y}$ denote a desired joint position and joint velocity, which represent our movement plan. The temporal scaling factor is denoted by τ and α_z and β_z define the damping properties of the linear system. The non-linear function f directly adds to the derivative of the internal state variable z, which is proportional to the desired acceleration of the movement plan.

For each degree-of-freedom (DoF) of the robot an individual dynamical system, and hence an individual function f is used. The function f only depends on the phase x of a movement, which represents time, $\tau\dot{x}=-\alpha_x x$. The phase variable x is initially set to 1 and will converge to 0 for a proper choice of τ and α_x. With τ we can modulate the desired movement speed. The function f is constructed by the weighted sum of K Gaussian basis functions Ψ_i

$$f(x)=\frac{\sum_{i=1}^{K}\Psi_i(x)w_i x}{\sum_{i=1}^{K}\Psi_i(x)}, \qquad \Psi_i(x)=\exp(-\frac{1}{2\sigma_i^2}(x-c_i)^2).$$

As the phase variable x converges to zero, the influence of f vanishes with increasing time. Hence, the dynamical system is globally stable for any initial and goal state, i.e. for $f=0$ the dynamical system represents a globally stable linear dynamic system with g as a unique point attractor.

Typically only the linear weights w_i are parameters of the primitive which can modulate the shape of the movement. The centers c_i specify at which phase of the movement the basis function becomes active. The centers are usually equally spaced in the range of x and not modified during learning. The bandwidth of the basis functions is given by σ_i^2.

Integrating the dynamical systems for each DoF results into a desired trajectory $\langle \mathbf{y}_t, \dot{\mathbf{y}}_t\rangle$ of the joint angles which is subsequentially followed by feedback control laws (see Section 5.2.6). The desired acceleration $\ddot{\mathbf{y}} = \dot{\mathbf{z}}/\tau$ of the system can also be seen as control action $\mathbf{a}$ of the agent, and thus, we can define a policy

$$\pi(\mathbf{a}|x;\mathbf{w}) = \mathcal{N}(\mathbf{a}|\mathbf{\Phi}(x)\mathbf{w} + \mathbf{k}, \mathbf{\Sigma}_a),$$

which is linear in the parameter $\mathbf{w}$ of the movement primitive. The linear features are given by

$$\mathbf{\Phi}(x) = \frac{\mathbf{\Psi}(x)x}{\tau^2 \sum_{i=1}^{K} \Psi_i(x)}$$

and the offset by $\mathbf{k} = \alpha_z(\beta_z(g - \mathbf{y}) - \mathbf{z})/\tau^2$. The linear policy representation allows an efficient use of imitation learning (Schaal et al., 2003), as well as for state-of-the-art policy search algorithms (Kober et al., 2008; Peters and Schaal, 2006; Theodorou et al., 2010b) which are only available for linear representations. For a more detailed discussion on available policy search algorithms we refer to Chapter 9.

Learning with DMPs often takes place in two phases (Kober and Peters, 2010). In the first phase, imitation learning is used to reproduce recorded trajectories. Subsequently, Reinforcement Learning is used to improve the movement.

The generation of the trajectory for DMPs is typically an offline process and does not incorporate proprioceptive (i.e. the actual joint position $\mathbf{q}_t$ does not influence the desired trajectory $\mathbf{y}_t$) or other sensory feedback. Exceptions are presented in (Kober et al., 2008) and (Kober et al., 2010). In (Kober et al., 2008), an additional feedback controller has been learned to modify the shape of the trajectory in order to catch the ball in the game 'ball in the cup'. Learning such a feedback controller drastically reduces the learning speed with DMPs. In (Kober et al., 2010), the authors learned to adjust meta-parameters of the DMPs such as the time constant τ or the end-point $\mathbf{g}$ of the movement to different situations (such as shooting a ball to different positions).

5.2.2 Planning Movement Primitives

This is a new idea for movement representation which is also introduced in this thesis, see Chapter 8. The key idea is to use planning already inherently inside the movement primitive. Instead of parametrizing the shape of the resulting trajectory, we now parametrize an internal cost function used for a probabilistic planner. This allows to use abstract features or goals as parameters and therefore a more compact movement representation. For further details please refer to Chapter 8.

5.2.3 Motion Templates from Exponential Functions

Motion templates are temporally extended, parametrized actions, such as exponential torque or velocity profiles, which can be easily sequenced in time. They have been introduced in our work in (Neumann et al., 2009) and (Neumann, 2011). The parametrization of a template is typically non-linear and thus more complex as for the DMPs. For example, it also incorporates the duration of the single template, like the duration of an acceleration or a deceleration phase. However, in difference to the DMPs, where a single primitive encodes the whole movement, the motion templates are much simpler, basic building blocks of the movement. Here, always several motion templates are required to represent the whole movement.

A motion template m_p is defined by its k_p dimensional parameter space $\mathcal{W}_p \subseteq \mathcal{R}^{k_p}$, its parametrized policy $u_p(\mathbf{s}, t; \mathbf{w}_p)$ ($\mathbf{s}$ is the current state, t represents the time spent executing the template and $\mathbf{w}_p \in \mathcal{W}_p$ is the parameter vector) and its termination condition $c_p(\mathbf{s}, t; \mathbf{w}_p)$.

At each decision-time point σ_k, the agent has to choose a motion template m_p from the set $\mathcal{A}(\sigma_k)$ and also the parametrization $\mathbf{w}_p$ of m_p. Subsequently the agent follows the policy $\pi_p(\mathbf{s}, t; \mathbf{w}_p)$ until the termination condition $c_p(\mathbf{s}, t; \mathbf{w}_p)$ is fulfilled. Afterwards, we obtain a new decision-time point σ_{k+1}. The advantage of such an approach is that value-based methods such as in (Neumann et al., 2009) can be used to estimate the values of the states of the decision time points which allows the combination of motion templates by sequencing them in time. This value based approach is part of this thesis and can be found in Chapter 7. Still, the simultaneous combination of the templates for several movement tasks remains an open problem.

The functional forms of the policy $\pi_p(\mathbf{s}, t; \mathbf{w}_p)$ and the termination condition $c_p(\mathbf{s}, t; \mathbf{w}_p)$ are defined beforehand and can be arbitrary functions. So far we used 2 types of motion templates, both are based on exponential functions and specify either torque or velocity profiles. The intuition behind the use of exponential functions is that the response of linear PD-controllers also has an exponential form (at least for linear systems). The exponential functions also resemble the bell-shaped velocity profiles often measured for human motion.

Torque Profiles

In (Neumann et al., 2009), the motion templates were directly used to parametrize the torque profile. The templates itself were implemented as exponential functions and were used for learning a 1-link and 2-link pendulum swing-up task. The used motion templates represent positive (m_1 and m_2) and negative peaks (m_3 and m_4) in the torque trajectory. There is also an individual template m_5 for balancing the robot at the top position. One peak consists of 2 successive motion templates, one for the ascending and one for the descending part of the peak. It is important to note that the duration of the peaks is also included in the parameters of the template, thus, the parametrization is highly non-linear. For a more exact description of the templates please consult Chapter 7. As we directly define the torque profiles, no feedback control is used for this type of templates (except for the balancing template m_5).

Velocity Profiles

We introduced this type of motion templates in (Neumann, 2011) to illustrate a new policy search algorithm, called Variational Policy Search. This paper is also part of this thesis and can be found in Chapter 10. Instead of torque profiles, the templates now define desired velocity profiles which are subsequently integrated to get a desired trajectory. The trajectory is then again followed by feedback control laws (we used linear PD-controllers). This is a big advantage in comparison to the previously used torque profiles as feedback control makes the outcome of a template easier to predict. We used this kind of templates for dynamic balancing of a 2-link and a 4-link pendulum. The balancing movement consisted of a fast bending movement to keep balance, subsequently the robot could return into the upright position.

The motion is divided into 2 motion templates. Template m_1 drives the robot to a set-point of each joint, Template m_2 tries to stabilize the agent at the upright position. Each template consists of an acceleration phase and a deceleration phase, both implemented by exponential velocity profiles. Template m_2 runs until the episode is terminated. For a more detailed description of the parametric form of the templates please consult the appendix of Chapter 10.

5.2.4 Time-Varying Muscle Synergies

Time-varying muscle synergies (d'Avella and Bizzi, 2005; Bizzi et al., 2008) have been used to provide a compact representation of EKG data of muscle activation patterns. In contrast to synchronous muscle synergies, time-varying muscle synergies also encode the temporal course of the muscle activation pattern. The key idea is that muscle activation patterns are composed of a linear sum of simpler, elemental patterns, denoted as single muscle synergy $\mathbf{m}_i(t; \mathbf{w})$. Each muscle synergy can now be shifted in time and scaled with a linear factor to construct the whole activation pattern

$$\mathbf{a}(t) = \sum_{i=1}^{K} c_i \mathbf{m}_i(t - \tau_i; \mathbf{w}),$$

where c_i is the linear scaling coefficients and τ_i the time shift coefficient. The parameters $\mathbf{w}$ of the primitive now incorporate a description of each single synergy (which can for example be implemented by Gaussian basis functions) and additionally the scaling and shift parameters for each primitive. The synergies have the promising property that some synergies might be shared between tasks and only the scaling and shift parameters need to be relearned. This property has already been shown to be true for the muscle activation of frogs performing different movements like jumping, swimming or walking and seems to be a promising approach for transfer learning in robots.

The time-varying synergy approach allows to combine the primitives simultaneously, which is straightforward due to the linear superposition. However, except for some smaller applications (Chhabra and Jacobs, 2006), these primitives have only been used for data analysis. It is not clear whether this property also holds for robot control. One obvious drawback of the time-varying muscle synergy approach is that there is no straightforward way to incorporate feedback because the synergies are typically used to directly decompose the motor commands. In order to incorporate feedback, the synergies need to decompose the joint trajectory instead of a torque or muscle activation trajectory. In this setup, many properties like that the linear superposition of synergies is useful, are likely to be lost.

5.2.5 Splines

Splines are a common piece-wise polynomial interpolation method which was used as one of the first movement representation (Chand and Doty, 1985; Kolter and Ng, 2009b). Most commonly used are cubic splines. The trajectory is represented by m via-points which are defined by the time points $t_0 \leq t_1 \leq \cdots \leq t_{m-1}$ and the control points $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{m-1}$. In each interval $t_i \leq t \leq t_{i+1}$, the trajectory is approximated by a cubic polynomial. The polynomial is fitted such that the trajectory coincides with the via-points $\mathbf{g}_i$ and $\mathbf{g}_{i+1}$ at the time-points t_i and t_{i+1} and that the first and second order derivatives of the trajectory are smooth. However, it is not clear if this is advantageous for representing movements.

The parameters $\mathbf{w}$ of the spline primitives are defined by the via-points, where usually the first via-point (the initial-state) and sometimes the last via-point (the goal-state) are pre-specified as prior knowledge.

5.2.6 Trajectory Tracking Controllers

Having discussed different ways how to parametrize the desired trajectory $\langle \mathbf{y}_t, \dot{\mathbf{y}}_t \rangle$, we still need a control law which is used to follow this trajectory.

Linear Feedback Control

The most simple feedback controller is to define a linear PD-controller

$$\mathbf{u}_t = \mathbf{K}_{\text{pos}}(\mathbf{y}_t - \mathbf{q}_t) + \mathbf{K}_{\text{vel}}(\dot{\mathbf{y}}_t - \dot{\mathbf{q}}_t).$$

The controller gains can either be pre-specified or also be learned from reinforcement. The number of parameters in the controller gain matrices depend quadratically on the number of dimensions of the system. We can further simplify the controller by assuming diagonal matrices for the controller gains, i.e. $\mathbf{K}_{\text{pos}} = \text{diag}(\mathbf{k}_{\text{pos}})$ and $\mathbf{K}_{\text{vel}} = \text{diag}(\mathbf{k}_{\text{vel}})$.

Inverse Dynamics Control

A more sophisticated approach is to use inverse dynamics control (Peters et al., 2008; Sciavicco and Siciliano, 2005). Instead of using the torque as controls, inverse dynamics control allows us to use directly the desired acceleration $\mathbf{z}$ to control the robot. However, this requires the knowledge of the dynamics of the robot

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{u},$$

where $\mathbf{n}(\mathbf{q}, \dot{\mathbf{q}})$ represents the terms coming from Coriolis and gravity forces and $\mathbf{u}$ is the applied torque to the joints.

If we replace the actual joint acceleration $\ddot{\mathbf{q}}$ with the desired acceleration $\mathbf{z}$, we can see that the control $\mathbf{u}$ is a function of manipulator state $\mathbf{s} = [\mathbf{q}, \dot{\mathbf{q}}]$ and the desired acceleration $\mathbf{z}$

$$\mathbf{u} = \mathbf{B}(\mathbf{q})\mathbf{z} + \mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}). \tag{5.1}$$

As $\mathbf{z} = \ddot{\mathbf{q}}$ denotes a decoupled linear system we can now use a simple PD-controller with diagonal controller gain matrices to generate the desired acceleration $\mathbf{z}$

$$\mathbf{z}_t = \mathrm{diag}(\mathbf{k}_{\mathrm{pos}})(\mathbf{y}_t - \mathbf{q}_t) + \mathrm{diag}(\mathbf{k}_{\mathrm{vel}})(\dot{\mathbf{y}}_t - \dot{\mathbf{q}}_t)$$

from which we can determine the joint tourques using Equation 5.1

5.3 Combination of Primitives

In this section we briefly discuss several possibilities to combine movement primitives. In principle there are 2 types of combinations, sequential and simultaneous combinations. Sequential combinations are only useful for temporal primitives. The obvious way if we want to combine 2 primitives sequentially is to specify when the first primitive is finished (e.g. by an additional parameter) and subsequently execute the 2nd primitive. However, we now have to learn the parameters of two primitives simultaneously, and therefore, the problem has also increasing complexity. A potential solution to this problem was illustrated in (Neumann et al., 2009) with the motion template framework. Here, the parameters of each primitive can be chosen separately at the point of time where the primitive is chosen for execution. This is done by using value-based methods to estimate the values at the states of the decision time-points. This has only be tried for quite simple templates representing exponential torque profiles. An evaluation for more sophisticated primitives is still missing.

The simultaneous combination is in most cases an unsolved problem. Here, the goal is to use two primitives simultaneously in order to fullfill two different tasks, for which the primitives were made, simultaneously. While this is easy for the spatial primitives such as synchronous muscle synergies and kinematic synergies, it is unclear how this can be done with temporal representations. Since the temporal primitives typically rely on a parametrized trajectory, these approaches would combine trajectories linearly, which is for many tasks not appropriate. Here, the exception is the time-varying muscle synergy representation, which is constructed by the simultaneous combination of synergies. However, this representation has only been applied to data analysis, it is not clear how this property can be directly transferred to robot control.

An interesting new idea for the simultaneous combination is provided by the planning movement primitives (see Chapter 8), where we do not need to combine trajectories linearly, we can (linearly) combine cost functions which might result in a strongly non-linear trajectory output of the planner. However, this idea of combining several cost functions still needs to be evaluated.

Chapter 6

Kinematic Synergies for Balancing Control

Contents

Despite many efforts, balance control of humanoid robots in the presence of unforeseen external or internal forces has remained an unsolved problem. The difficulty of this problem is a consequence of the high dimensionality of the action space of a humanoid robot, due to its large number of degrees of freedom (joints), and of nonlinearities in its kinematic chains. Biped biological organisms face similar difficulties, but have nevertheless solved this problem. Experimental data show that many biological organisms reduce the high dimensionality of their action space by generating movements through linear superposition of a rather small number of stereotypical combinations of simultaneous movements of many joints, to which we refer as kinematic synergies in this paper. We show that by constructing two suitable nonlinear kinematic synergies for the lower part of the body of a humanoid robot, balance control can in fact be reduced to a linear control problem, at least in the case of relatively slow movements. We demonstrate for a variety of tasks that the humanoid robot HOAP-2 acquires through this approach the capability to balance dynamically against unforeseen disturbances that may arise from external forces or from manipulating unknown loads.

6.1 Introduction

Humanoid robots are constructed to have the form of a human body in order to be able to work in environments optimized for human needs. In the near future they are meant to work with people, and human like shape would increase the possibility of acceptance of robots in human society. However, the humanoid form carries the burden of being very difficult to control compared to wheeled robots for instance. One of the biggest problem is the issue of balance like counterbalancing unknown perturbations. This is a standard situation in a real environment and has to be solved as a prerequisite to any interaction. Due to their human structure, humanoid robots are bipedal, and have therefore a smaller support polygon (which is defined as the convex hull of the foot support area) compared to, for example, quadrupeds. In addition, two-thirds of their body mass is typically located above two-thirds of body height (Winter, 1995). Both facts contribute to the instability of humanoid robots. Furthermore, a failure of their balance control is not only bad for the robot, since a fall is likely to produce damages, but may also hurt people that interact with the robot. Therefore, a crucial point for allowing human robots to work in human environments is to find robust and effective methods for their balance control. These solutions should induce naturally looking movements in order to increase the possibility of acceptance of humanoid robots as partners of humans.

The balance control problem of humanoid robots is known to be hard to solve due to the high dimensionality of their action space (since many degrees of freedoms, i.e., joints, are involved) and the nonlinearities inherent to any kinematic chain. Because of the importance of finding solutions to this problem, a lot of effort has already been invested and many approaches from different research areas have been proposed.

A first step was made by introducing the *Zero Moment Point* (ZMP) criterion (Vukobratović and Borovac, 2004). It simplifies the high dimensional problem by reducing all acting forces above the foot (in case of single support, i.e., contact with the ground with only on foot) to one single force (Vukobratović and Borovac, 2004). Due to physical interaction between foot and ground we get at a so-called ground reaction force. This ground reaction force is located at the point where the force between foot and ground acts and has opposite sign. This two dimensional point (called ZMP) on the ground can then be used to characterize the dynamic state of the robot: If the ZMP lies within the support polygon of the robot, the state of the robot is called dynamically stable. This 'ZMP stability criterion' reduces the problem of stability to coordinate the limbs of the robot (i.e., apply appropriate torques through their servos) in such a way, that the ZMP stays within the support polygon[1].

While the ZMP can be calculated analytically, the position of this point can also be measured by pressure sensors (actually measuring the ground reaction force). From this point of view the resulting point is called accordingly *Center of Pressure* (CoP). As Goswami demonstrated (Goswami, 1999) the ZMP equals the CoP, since they describe the same phenomenon from different points of view. In this paper we are going to use the name CoP, since we use the pressure sensor information in combination with the support polygon to estimate the state of stability. Since the original ZMP definition has some limitations (Goswami, 1999), other ground reference points have been proposed, for example, the *Foot Rotation Indicator* (FRI) introduced by Goswami (Goswami, 1999) or the *Centroidal Moment Pivot* (CMP), just to name two. For a detailed discussion we refer to (Popovic et al., 2005).

[1]The robot could also change the size of the support polygon by, for example, hold on to something. For a discussion of different control strategies in this context we refer to (Goswami and Kallem, 2004).

Other approaches have been proposed that are also based on a reduced model of the robot. For example, the *Inverted Pendulum Model,* introduced by Kajita et. al. (Kajita et al., 1992), has proved to be very useful. It describes the whole robot, under some assumptions, by a linear inverted pendulum and thereby, reduces the number of dimensions. Extensions of this model have also been studied, for example, the *Three-Dimensional Inverted Pendulum Model 3D-LIPM* (Kajita et al., 2001) and the *Reaction Mass Pendulum* (RMP) (Lee and Goswami, 2007). Although all these reduced models are useful, still, at the point of implementation one has to find control schemes which map the strategy back into the full dynamic model (as Lee and Goswami pointed out (Lee and Goswami, 2007)). Hence, they have difficulties dealing with unknown external perturbations, since these perturbations present a change in the dynamics of the robot.

An alternative approach to balance control is to rely on the static model, i.e., to use the kinematic model and the mass distribution of the robot. By employing a local Jacobian Pseudo-Inverse (JPI) approach on local information, like Resolved Motion Rate Control (RMRC) (Whitney, 1969), the optimal change of the joint angles can be calculated. Some of these frameworks even allow to set priorities amongst conflicting tasks (Baerlocher and Boulic, 1998, 2004). Accordingly, balancing could be one of these tasks, typically with a high priority. In order to deal with unforeseen perturbations, the setup has to be used inside a feedback control loop, for example as proposed in (Mansard and Chaumette, 2007). However, a drawback of such an approach is that it calculates online inverse kinematics, which involves computationally expensive matrix inversions.

Other approaches try to solve directly the dynamic equations within constraints, which reflect the border of stability. For example, Kagami et. al. (Kagami et al., 2001) proposed an online balancing scheme by solving a quadratic programming problem. However, the precise dynamic model of the robot is needed in order to apply this approach. Therefore, it has difficulties in situations where the dynamic model of the robot significantly changes due to external unknown forces, for example, introduced by picking up unknown loads or contact with the environment, which are standard situations for humanoid robots working in a human environment.

Biological organisms face similar problems, but, as experimental data suggest, employ a radically different strategy for controlling their movement apparatus with many degrees of freedom (DoF), in particular for balance control. Numerous studies from the Lab of Bizzi at MIT ((Mussa-Ivaldi, 1999; d'Avella et al., 2003; d'Avella and Bizzi, 2005)) have shown that the central nervous systems of a variety of organisms employ a modular architecture for motor control, whereby many different movements (arm movements, walking, jumping, swimming) can be constructed as largely linear (but non-negative) combinations of a rather small repertoire of movement primitives. A very simple modular architecture in the context of biological data analysis are the *synchronous muscle synergies* (d'Avella and Bizzi, 2005), which are a spatial movement representation (see Chapter 5). Synchronous muscle synergies define a low-dimensional linear basis of a high dimensional control vector, like the muscle activations. The muscle activation vector $\mathbf{a}(t)$ is therefore given by

$$\mathbf{a}(t) = \mathbf{M}\mathbf{c}(t),$$

where each column-vector of $\mathbf{M}$ defines a single muscle synergy and the vector $\mathbf{c}(t)$ defines the muscle synergy coefficients. However, it is difficult to translate such representations directly to robot control as they are directly defined in the action space of the robot and therefore, do not provide any facility for incorporating (proprioceptive) feedback. In this work we introduce a robot control strategy which has been inspired by the synchronous muscle synergy approach, however, we define the synergies in the joint space of the robot, which easily allows the use of feedback control laws. Since the synergies are defined by the kinematic chain of the robot, we will denote this approach as *kinematic synergies*.

Also recent work on whole-body movements of humans ((Freitas et al., 2006; Tricon et al., 2007; Torres-Oviedo and Ting, 2007)) show that balance control and other human body movements during standing can be understood as combinations of a small set of stereotypical kinematic synergies (each of them affects several joints). Experiments, where humans where asked to bend their upper trunk, while recording the angles of the ankle, hip and knee, revealed after a Principal Component Analysis (PCA) of these angles, that already the first principle component can explain over 99% of the total angular variance (Alexandrov et al., 1998). This suggests that a set of muscles (multiple degrees of freedom) are controlled by a low dimensional (possibly even one dimensional) variable. Other experiments suggest that this principle of kinematic synergies is present over a wide range of different movements like reaching and grasping (Mason et al., 2001), upper-arm movement (Sabatini, 2002) and making a step (Wang et al., 2005). Hence, kinematic synergies seem to present a general strategy biological organisms apply.

We are specially interested in humanoid balance control. In a preceding conference paper (Hauser et al., 2007) we demonstrated how this basic modular strategy based on kinematic synergies can be adapted for balance control of a humanoid robot. The kinematic synergies were calculated offline by an optimization process based only on the *static* model (kinematics and masses) of the robot[2]. Despite the use of the static model, we could demonstrate that the concept of kinematic synergies, when plugged into a linear control loop, can provide a powerful scheme for *dynamic* balance control. This article presents an extension of the previous work (Hauser et al., 2007) by following points: (1) We demonstrate that our approach of kinematic synergies is robust to parameter changes of the model of robot. Changes of the static model present a standard situation for biological systems since they grow or even get injured (e.g., loosing a leg). (2) Additionally, we show that no special tuning of the controller parameters is needed since the proposed framework works (i.e., balances the robot) within a wide range of these parameters. (3) We demonstrate that the chosen kinematic synergies, originally designed for double support, can also be applied for the case of single support. (4) Finally, we demonstrate that the proposed approach for balance control can be transferred from a simulated humanoid robot without any changes to a real humanoid robot.

In the next section we define the kinematic synergies. Section 6.3 explains how to construct and use kinematic synergies for balance control of the humanoid robot

[2]This optimization process is closely related to the Jacobian Pseudo-Inverse approaches (Sciavicco and Siciliano, 2005), however, the computations are only needed for the offline construction of the synergies and not during online control.

HOAP-2. In Section 6.4 we present a number of experiments with the simulated and the real HOAP-2.

6.2 Formal Definitions of Kinematic Synergies

In this section we define the kinematic synergies which are used to reduce high dimensionality and nonlinearities. Typically, humanoid robots have a high number of degrees of freedom (DoF), namely joints. We interpret kinematic synergies (*KS*) as a way to reduce the DoF by putting a defined set of joints under the regime of one controlling parameter, which we refer to as the *KS*-parameter s. We define a kinematic synergy as a nonlinear mapping $\mathbf{\Phi}$ of the *KS*-parameter $s \in \mathbb{R}$ to a fixed number of m degrees of freedom (joints).

Definition 1. *A* kinematic synergy *(*KS*) is a function $\mathbf{\Phi} := \mathbf{\Phi}(s)$ which maps the* KS*-parameter $s \in \mathbb{R}$ onto a m dimensional vector of joint angles $\mathbf{q}^{KS} = \mathbf{\Phi}(s)$:*

$$\mathbf{\Phi} : \mathbb{R} \to \mathbb{R}^m . \tag{6.1}$$

The superscript ^{KS} denotes the subset of m joints, which are controlled by the *KS*. The total number of joints of the robot is denoted by n. Further, we define the function φ

$$\varphi : \mathbb{R}^m \to \mathbb{R}^n \tag{6.2}$$

to embed the m-dimensional subspace spanned by $\mathbf{\Phi}$ into the n-dimensional space of all joints of the robot. This embedding copies the angles of all joints affected by $\mathbf{\Phi}$ and leaves the remaining joints constant.

A *KS* is typically applied in order to control a low-dimensional, or even one-dimensional, variable $y \in \mathbb{R}^l$. In general the output y depends on all n joint positions $\mathbf{q} \in \mathbb{R}^n$ of the robot and can be described by a nonlinear function $\mathbf{f}(\mathbf{q})$

$$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^l . \tag{6.3}$$

We want the *KS* to control the output $y = (\mathbf{f} \circ \varphi \circ \mathbf{\Phi})(s)$. In the case of balance control, the function $\mathbf{f}$ represents the nonlinear relationship between all joints of the robot and a ground reference point like the CoP. We will use two *KS* $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ for the two dimensions of the CoP. Therefore, in this particular case each *KS* is used to control a one-dimensional output ($l = 1$).

Since such a *KS* affects m degrees of freedom that depend just on a one dimensional parameter s, we can impose further constraints on the function $\mathbf{\Phi}$. A reasonable choice for such a constraint is a linear relationship between the controlling parameter s and its corresponding output y. This reduces nonlinearities, inherent to kinematic chains, and hereby facilitates controlling and learning. Hence, we are particularly interested in the following type of *KS*:

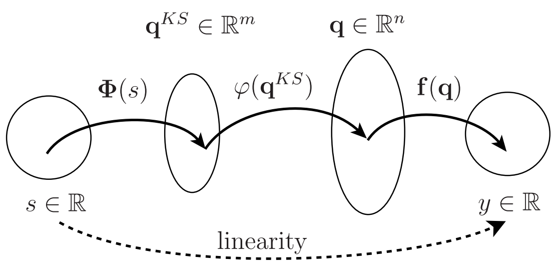

$\mathbf{q}^{KS} \in \mathbb{R}^m$
$\mathbf{q} \in \mathbb{R}^n$
$\mathbf{\Phi}(s)$
$\varphi(\mathbf{q}^{KS})$
$\mathbf{f}(\mathbf{q})$
$s \in \mathbb{R}$
$y \in \mathbb{R}$
linearity

Figure 6.1: Scheme for the composition of the functions φ and $\mathbf{f}$ according to (6.2) and (6.3) with the kinematic synergy $\mathbf{\Phi}$.

Definition 2. *A* linearizing kinematic synergy *is a kinematic synergy according to Definition 1, which has a linear relationship between its controlling parameter s and the corresponding (to be controlled) output y*

$$y = (\mathbf{f} \circ \varphi \circ \mathbf{\Phi})(s) = k \cdot s, \qquad k \in \mathbb{R}\,. \tag{6.4}$$

We restrict our attention in this article to such *linearizing KS*, to which we simply refer as *KS*.

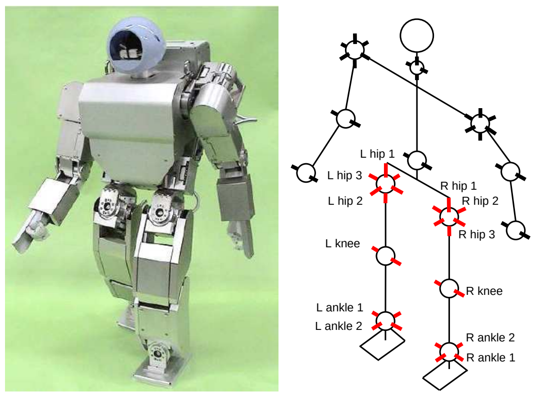
For a better understanding we provide some additional remarks:

1. As stated above the property of linearity in Definition 2 reduces inherent nonlinearities. But Equation 6.4 presents a static mapping, and therefore it will only linearize the static part (linearization at $\dot{\mathbf{q}} = \mathbf{0}$, $\ddot{\mathbf{q}} = \mathbf{0}$) of the whole dynamic model of the robot. Nevertheless, it will reduce nonlinearities in the dynamic regime to some extent too, since the dynamic part is coupled with the static part of the differential equations.

2. The controlled variable y is one-dimensional, but is controlled by $m > 1$ joints. Hence, we have additional redundant degrees of freedom and therefore, we are free to impose additional constraints on the *KS*. Naturally, the choice will depend on the task for which the *KS* are constructed. In our case of balance control we used constraints to assure double support and an upright posture (used in the optimization process described in Section 6.3.1).

3. *KSs* are calculated offline for each robot (see Section 6.3.1) and subsequently fixed during simulation as well as when used with the real robot. In a biological interpretation we assume the *KSs* to be found by evolution.

4. The presented framework was kept as simple as possible. Various extensions, which lead to a better performance for particular tasks, are possible. One could define a two dimensional kinematic synergy (i.e., $s \in \mathbb{R}^2$ and $y \in \mathbb{R}^2$) or time-varying *KSs* ($\mathbf{q}^{KS} = \mathbf{\Phi}(s,t)$*)*, which depend on a cyclic movement, for example, to be used in a walking cycle.



(a) (b)

Figure 6.2: **(a)** The real HOAP-2 robot and **(b)** its schematic structure. The red marked and labeled joint rotation axes are controlled by the kinematic synergies Φx and Φz.

6.3 Using Kinematic Synergies for Balance Control of the Humanoid Robot HOAP-2

In this section we show in detail how to use kinematic synergies for balance control of the humanoid robot HOAP-2, see Figure 6.2(a). The robot has $n = 25$ degrees of freedom (rotational joints). Its structure can be seen in Figure 6.2(b). The goal is to construct *KSs* for balance control in double support. Therefore, we have to decide **(a)** what output function $\mathbf{f}$ and output variables y we are going to use, **(b)** which subset of m joints we put under the regime of the *KSs* and **(c)** what additional constraints we are going to apply to construct the *KS*s:

(a) For balance control a natural choice for the function $\mathbf{f}$ is a ground reference point. These points are mathematically defined and can be analytically derived, but in practice, they are estimated via pressure sensors. Therefore, we will denote the reference point measured by the pressure sensors as *measured Center of Pressure* (*mCoP*). HOAP-2 has four of such sensors per feet, located at the corners (see Figure 6.3).

Since a *KS* is defined as a static mapping, we use the static version of the *mCoP* to construct our *KS*. In the static case (zero joint velocity $\dot{\mathbf{q}}$ and zero joint acceleration $\ddot{\mathbf{q}}$) the *mCoP* coincides with the *projected Center of Mass* (pCoM). Therefore, we chose the pCoM as output function $\mathbf{f}$. Since the pCoM is a two dimensional point on the supporting surface, we split it up into its two dimensions $y_x = \text{pCoM}_x$ and $y_z = \text{pCoM}_z$ and define two separate *KS*s, namely Φ_x and Φ_z, in order to control these one-dimensional outputs y_x and y_z.

(b) Next, we have to decide what joints are placed under the regime of our *KSs*. A natural choice for balance control is to use all $m = 12$ leg joints (three hip joints, one knee joint and two ankle joints for both legs). Their corresponding rotational axes are highlighted in red in Figure 6.2(b). The additional surplus

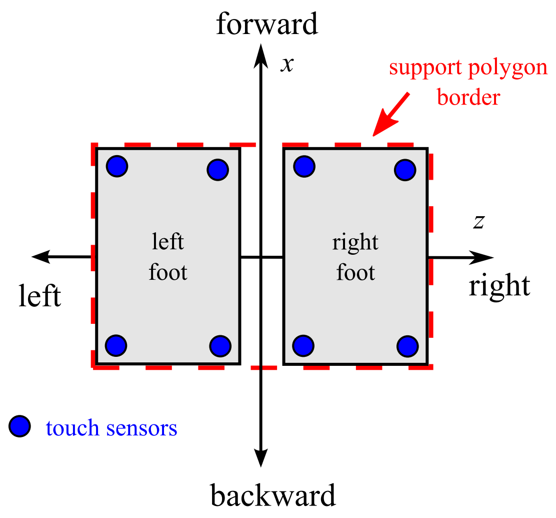


Figure 6.3: Support polygon on the support surface for the robot, including the touch sensors, which are used to measure the center of pressure ($mCoP$). Black arrows indicate the x dimension (forward/backward: range 9.5 cm) and z dimension (left/right: range: 14.3 cm) for movements of the center of pressure.

of joints are free to be used for other tasks (grasping, lifting weights, tracking objects, etc.). Their movements clearly will change the pCoM too, but as we show later in Section 6.4, our approach is able to deal with that in a natural way.

(c) Finally, we choose some additional constraints (next to the linearity property) for the KSs, which are used for the optimization process described in the next subsection. Suitable constraints for balance control are to keep the upper body as upright as possible and to maintain double support.

6.3.1 Calculating Kinematic Synergies with Inverse Kinematics

In this section we describe how do obtain the desired KSs in detail. All calculations are based only on the kinematic model of the robot including the mass information (no dynamical information like the inertia matrices is needed). The KSs were constructed offline and subsequently fixed during control action.

We defined an initial posture $\mathbf{q}_{init}$ (see Figure 6.5-A). This posture resulted (for the case of a horizontal support surface) in a pCoM at the center of the support polygon. We used a posture with wide-spread arms in order to avoid self collision when moving. The KS-parameters s_x and s_z were rescaled such that the values -1 and $+1$ corresponded to the borders of the support polygon. Therefore, the region of acting without falling was (for the case of a horizontal support surface) $s_x/s_z \in [-1, +1]$ for both dimensions x and z, see red-dashed lines in Figure 6.3.

We additionally set the origin of the coordinate system for the pCoM to the center of the support polygon and therefore, the resulting outputs in the initial posture were $\mathbf{f}_x(\mathbf{q}_{init}) = \mathbf{f}_z(\mathbf{q}_{init}) = 0$.

We will only describe the procedure for $\mathbf{\Phi}_x$. The second kinematic synergy $\mathbf{\Phi}_z$ was obtained in a similar manner. The *KS* was implemented as look-up table which maps the *KS*-parameter $s_x \in [-1, +1]$ to joint angle offsets (with regard to the initial posture)[3], i.e., $\Delta\mathbf{q}_x = \varphi(\mathbf{q}_x^{KS}) - \mathbf{q}_{init}$. Note that the look-up table represents a discretized version of a *linearizing kinematic synergy* as defined in Definition 2. In order to obtain joint angle offsets in between the table entries a linear interpolation was used. We used joint angle offsets instead of absolute joint angles in order to be able to use a linear superposition (as biological data suggest) of both *KS*s, i.e., $\Delta\mathbf{q} = \Delta\mathbf{q}_x + \Delta\mathbf{q}_z$. Although, the problem is (due to the kinematic chains) nonlinear, we will show that a linear superposition is valid for a wide range of postures. The linear superposition allows us to use two separate simple *KS*, which depend only on a one-dimensional *KS*-parameter, and which can be constructed independently[4].

In order to construct the look-up table, we divided the range of the *KS*-parameter s_x over the support polygon into 80 points. Therefore, the distance between two neighboring points represents 9.5 cm / 80 $\approx$ 0.12 cm in the pCoM space, which corresponds to a step of $\Delta s_x = 0.025$ in the *KS*-parameter space.

The construction of the *KS* consisted of two alternating optimization steps (see optimization scheme in Figure 6.4). Starting from $\mathbf{q}_{init}$ and $s_x = 0$, the first optimization step was used to move the pCoM of the robot to the next point y'_x of the look-up table (located 0.12 cm in x-direction from the origin). In addition, the optimization tried to keep the upper part of the body upright. An inverse kinematics algorithm based on the Jacobian Pseudo-Inverse (JPI) (Sciavicco and Siciliano, 2005) was used to calculate the joint movement. Therefore, the applied Jacobian matrix consisted of two $3 \times m$ sub-matrices, the Jacobian for the position of the pCoM and the Jacobian for the rotation of the torso. However, due to the movement calculated by this optimization, the position of the right foot relative to the left foot tended to change. This should be avoided in order to prevent the robot from falling. Therefore, a second JPI optimization step (see Figure 6.4) was used to move the right foot back into its original position relative to the left foot. For this optimization the same Inverse Kinematics algorithm was applied using only the 6 joints of the right leg.

These two previously described steps were iterated until the desired output value y'_x was reached. Subsequently, the joint angle offsets to the initial posture were stored in the look-up table and, now starting from the new joint position, the next entry of the look-up table was calculated. The same process was applied for the opposite direction (i.e., for s_x from 0 to -1). This finally led to a look-up table for the range $s_x \in [-1, +1]$ which mapped the *KS*-parameter s_x to joint angle offsets.

Figure 6.5 presents four typical postures for different *KS*-parameter pairs $[s_x/s_z]$. The center of the figure shows the support polygon (gray area) and the coordinate

[3]The function φ is used to project the m-dimensional vector $\mathbf{q}_x^{KS}$ into the n-dimensional space of all joints.

[4]Without this property, one would have to construct one single *KS* with a two-dimensional *KS*-parameter, i.e., $s \in \mathbb{R}^2$.

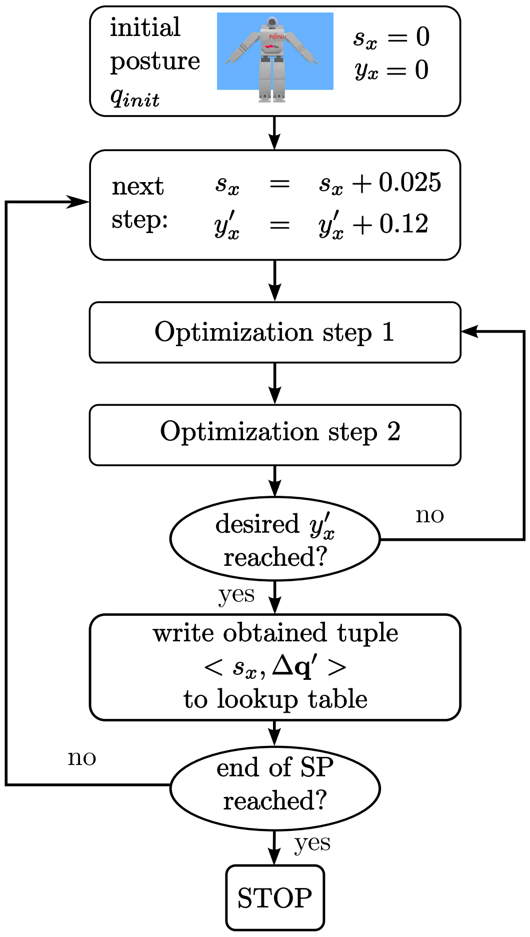


Figure 6.4: Scheme of the construction process for the look-up table for the *KS* $\mathbf{\Phi}_x$ in the form $< s_x, \Delta\mathbf{q} >$. Optimization step 1 moves the pCoM in the desired direction to y'_x, while keeping the trunk in an upright position. Optimization step 2 keeps the feet at the initial positions. The process ends when the end of the support polygon (SP) is reached.

system of the *KS*-parameters. The yellow circles (A-D) represent the postures in the *KS*-parameter space. The corresponding screenshots can be seen in the corners of the figure.

Figure 6.6(a) shows the mapping of the *KS*-parameter s_x to the outputs y_x =pCoM$_x$ and y_z =pCoM$_z$ for the *KS* $\mathbf{\Phi}_x$. We can clearly identify a linear relationship between s_x and y_x, whereas the second output dimension y_z is unaffected by s_x. The same plot for the *KS* $\mathbf{\Phi}_z$ is shown in Figure 6.6(b).

A graphical representation of the joint angle offsets over the range of the *KS*-parameter space (from -1 to $+1$) for the kinematic synergies $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ is presented in Figure 6.7. Similar to their biological prototypes (see Figure 4 in (d'Avella and Bizzi, 2005)), the two *KSs* largely affect disjoint sets of joints. The joints mainly responsible for the movement in x-direction are orthogonal to the joints mainly responsible for the z-direction. Note that the human muscle-skeleton system exhibits, although more complex, a similar structure. This orthogonality suggests that we

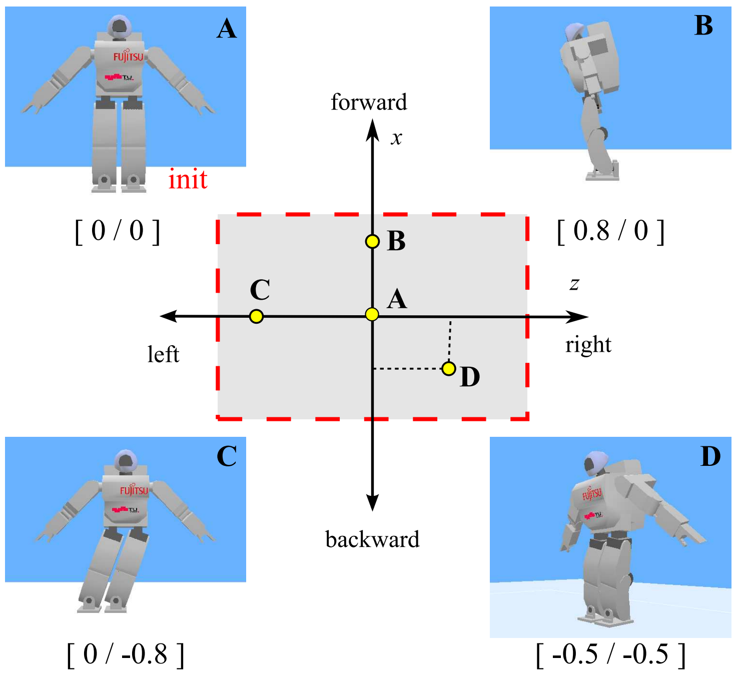


Figure 6.5: Typical postures of the simulated HOAP-2 resulting from the *KSs* $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ for different *KS*-parameters. The center of the figure shows the defined coordinate system for the *KS*-parameters s_x and s_z. The gray shaded area indicates the support polygon (SP) of our robot standing with both feet on the ground. The red dashed lines depict the limits of the SP and correspond to the values $s_x = \pm 1.0$ and $s_z = \pm 1.0$. The yellow points show typical postures in the *KS*-parameter space. Corresponding postures can be seen in the corners (labeled from **A** to **D**). The used *KS*-parameters $[s_x/s_z]$ can be seen below the screenshots. Screenshot **[A]** shows the initial posture $\mathbf{q}_{init}$ ($s_x = s_z = 0$ / at the origin) **[B]** shows the robot bending forward with $s_x = 0.8$ and $s_z = 0.0$, while in **[C]** the robot is bending to the left (with $s_x = 0.0$ and $s_z = -0.8$). Screenshot **[D]** presents a combination of both kinematic synergies with $s_x = -0.5$ and $s_z = 0.5$.

can combine the two *KSs* linearly, which is done by summing up the initial posture and the two joint angle offsets $\mathbf{q}_L = \mathbf{q}_{init} + \Delta\mathbf{q}_x + \Delta\mathbf{q}_z$.

In order to show the validity of the linear superposition of the two *KS*s, we evaluated empirically the deviation of the actual pCoM $< \mathbf{f}_x(\mathbf{q}_L), \mathbf{f}_z(\mathbf{q}_L) >$ from the case of perfect linear superposition $< \mathbf{f}_x(\mathbf{q}_{init} + \Delta\mathbf{q}_x), \mathbf{f}_z(\mathbf{q}_{init} + \Delta\mathbf{q}_z) >$. The deviations for the whole support polygon can be seen in Figure 6.8. Except for extremal cases, where the pCoM is located at a corner of the support polygon, the deviations from linearity are quite small.

Note that the described optimization procedure is closely related to standard JPI approaches. However, these approaches are typically used for online control, involving computationally expensive real-time calculations. With the use of kinematic synergies most of this computational load can be transferred to the offline optimization scheme. As a consequence, and as we will demonstrate later, without a significant loss of performance the robot can be balanced with very little compu-

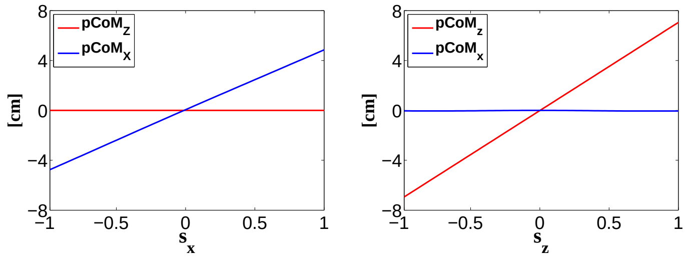


Figure 6.6: **(a)** The plot shows the mapping of the KS-parameter s_x to the outputs $y_x = \text{pCoM}_x$ and $y_z = \text{pCoM}_z$ for the KS $\mathbf{\Phi}_x$. While the relationship between s_x and y_x is linear (as demanded by the definition of a *linearizing kinematic synergy*), y_z is nearly unaffected by s_x. **(b)** The same plot for the second KS-parameter s_z.

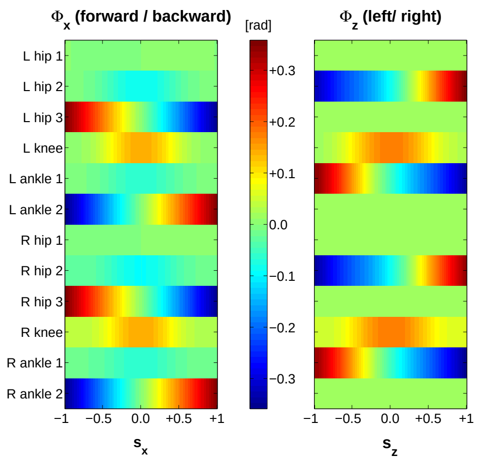


Figure 6.7: Graphical representation of the KSs $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$. Shown are the joint angle offsets (in color coding) for the kinematic synergies $\mathbf{\Phi}_x$ (moves the pCoM forward/backward) and $\mathbf{\Phi}_z$ (moves the pCoM left/right) for the HOAP-2 over the range $[-1, +1]$ for the KS-parameters s_x and s_z. Note that these two *KSs* affect largely disjoint sets of joints.

tational power.

6.3.2 From Statics to Dynamics by Using Linear Controllers

The kinematic synergies $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$ were constructed using the pCoM as output function, and therefore they were based on the static model of the robot. However, the robot can only estimate the $mCoP$ with its pressure sensors[5], which is also affected by the dynamics of the robot. Nevertheless, we are still able to use the obtained *KSs* in a dynamic context if following assumption holds:

***Assumption*:** The robot moves sufficiently slowly such that

$$mCoP \approx \text{pCoM}.$$

[5]In our simulations of the HOAP-2 we also used simulated pressure sensors to calculate the $mCoP$.

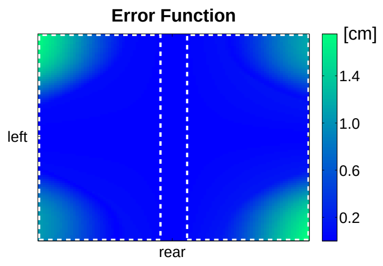


Figure 6.8: Empirical evaluation of the validity of the linear superposition of the KSs $\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$. We calculated the deviation of the actual pCoM $< \mathbf{f}_x(\mathbf{q_{init}} + \mathbf{\Delta q_x} + \mathbf{\Delta q_z}), \mathbf{f}_z(\mathbf{q}_{init} + \Delta\mathbf{q}_x + \Delta\mathbf{q}_z) >$ from the case of perfect linear superposition $< \mathbf{f}_x(\mathbf{q}_{init} + \Delta\mathbf{q}_x), \mathbf{f}_z(\mathbf{q}_{init} + \Delta\mathbf{q}_z) >$. The Euclidean norm of the deviations is shown in color code for the whole support polygon. Except for extremal cases, where the pCoM is located at a corner of the support polygon, the deviations from linearity are quite small. The white dotted lines depict the contours of the feet.

As we will demonstrate in this section, the assumption allows us to use simple linear controllers in conjunction with the KSs. Due to the assumption we are in principle limited to "sufficiently slow" movements. However, we will demonstrate in our experiments that a wide range of unknown external forces can be counterbalanced by our approach, despite this limitation.

We now explain how the kinematic synergy $\mathbf{\Phi}_x$ can be used in combination with a linear controller for balancing the robot in x-direction (forward/backward). For the other KS $\mathbf{\Phi}_z$ the process is similar. As long as the assumption holds, the function from the time derivative $\dot{s}_x$ of the KS-parameter to $mCoP_x$ can be approximated by a linear transfer function

$$P(\mathsf{z}) = \frac{K}{(\mathsf{z} - 1)} \ , \tag{6.5}$$

with $K \in \mathbb{R}^+$ and with z being the time shift operator for discrete systems (Oppenheim and Willsky, 1992). The denominator polynomial represents an integrator (one pole at $\mathsf{z} = +1$), which integrates the velocity $\dot{s}_x$ of the KS-parameter to obtain s_x.

As long as the dynamical effects are small enough, they can be seen as uncertainties in the linear model of Equation 6.5. Already a simple linear feedback controller can handle these small uncertainties. In order to obtain a closed control loop we define a feedback error

$$e_x := \tilde{y}_x - y_x \tag{6.6}$$

with $\tilde{y}_x$ being the desired output value and $y_x = mCoP_x$. The goal is to prevent the robot from falling. Therefore, the mCop$_x$ should stay close to the center of the support polygon. Since we have defined the center of SP at the origin, see Figure 6.3, the desired value $\tilde{y}_x$ is set to 0.

We can now use a general standard PID controller to get the controller output $\dot{s}_x$

$$\dot{s}_x = K_P e_x + K_I \int e_x dt + K_D \frac{de_x}{dt} \quad , \tag{6.7}$$

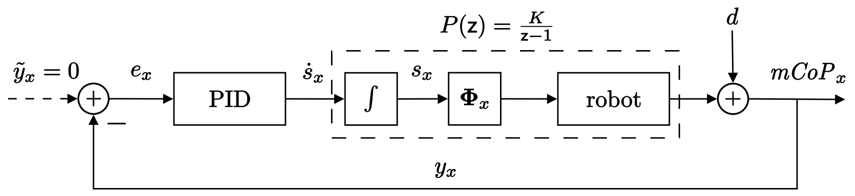


Figure 6.9: Closed control loop for the kinematic synergy $\mathbf{\Phi}_x$. Since we want to have the $mCoP_x$ at the center of the support polygon, the reference point is set to $\tilde{y}_x = 0$. The external perturbation d results from external forces and/or model uncertainties.

where K_P, K_I and K_D are the positive PID controller parameters. Figure 6.9 shows the described closed control loop for the kinematic synergy $\mathbf{\Phi}_x$. Since the plant (see Equation 6.5) already contains an integrator, the use of PD controllers ($K_I = 0$) is sufficient. For the *KS* $\mathbf{\Phi}_z$ we used a similar control loop, which worked independently from and in parallel to the first control loop.

We have described the control scheme to control around a set point ($\tilde{y}_x = \tilde{y}_z = 0$). However, the control loop can also be used to move the *mCoP* on any desired time varying trajectory[6], i.e., $\tilde{y}_x(t)$ and $\tilde{y}_z(t)$. This is useful in many applications. For example, for the purpose of initiating a walking cycle, the robot has to move its *mCoP* under the future supporting foot in order to be able to raise the other leg without falling.

The controller parameters used in the experiments were empirically found to have a reasonable performance. As we demonstrate (see Subsection 6.4.3) there is a wide range of appropriate controller parameters and therefore the choice of the parameters is not critical.

Linear and nonlinear control theory offers a number of possible improvements for the controllers, for example, adaptive control (see (Astrom and Wittenmark, 1995)) or robust control schemes, optimal control and different trial and error approaches to find good control parameters (see for example (Kuo and Golnaraghi, 2002)). Even higher order controllers or different control structures than in Figure 6.9 could be used. However, in order to illustrate the capability of using kinematic synergies for balance control, we only use the previously presented, simple PID controllers.

6.3.3 Examination of Different Possible Perturbations

Lets take a closer look at possible perturbations d for the proposed control loop (Figure 6.9). We will distinguish between three different kinds of perturbations:

1. *Model perturbations:* Since we obtained our *KSs* from the static model of the robot, unmodeled dynamics, which will always be present to some extent, result in model perturbations.

2. *Internal perturbations*: The *mCoP* is also influenced by movements of joints, which are not under the control of the kinematic synergies. For example, if our humanoid robot uses the presented *KSs* for balancing and additionally

[6] We have already demonstrated that in (Hauser et al., 2007).

moves a heavy weight with its arms, this movement will also change the $mCoP$ position. Note that the proposed control loop does not need any information about the movements of these joints.

3. *External perturbations:* For example pushes, pulls, contact with the environment or a moving support platform.

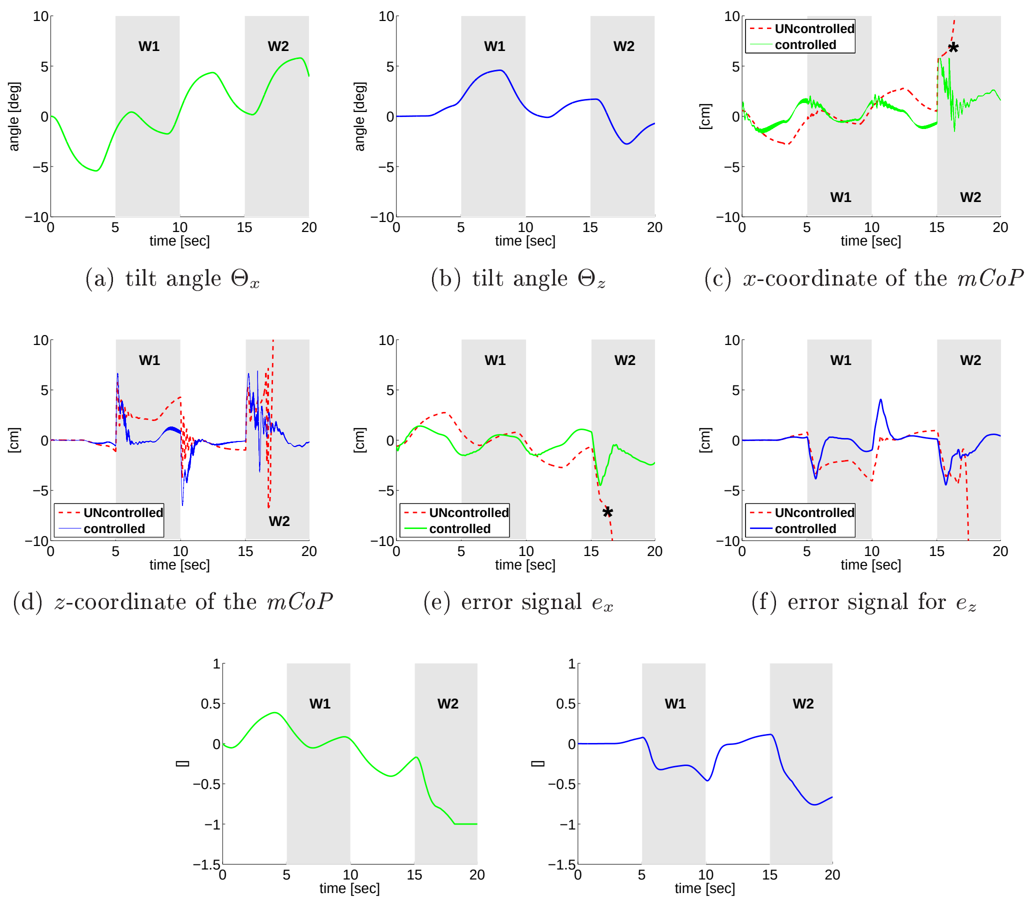
Since a standard feedback control loop has the property to suppress the perturbations d, our approach works for a wide range of tasks. As shown in our experiments (Hauser et al., 2007), these tasks include counteracting external forces, following trajectories, compensating for forces introduced by movements of the limbs of the robot or even a mixture of these tasks. If the perturbation is too large, the assumption ($mCoP \approx$ pCoM) might be violated and the controller will therefore not be able to compensate the resulting error anymore. Yet, as our experiments show, the proposed system is capable to react appropriately to a wide range of perturbations.

6.4 Experiments

We conducted experiments with our proposed approach for a variety of possible applications. We demonstrate that kinematic synergies with linear controllers empower a humanoid robot to counterbalance different kinds of dynamic perturbations. In our first experiments the robot had to counteract a moving support surface (platform where it stood on) and abrupt unforeseen external forces at the same time (see Subsection 6.4.1). Subsequently, we show that the approach can also be extended easily to balancing in single support (the robot only stood on the left foot, see Subsection 6.4.2) and that robustness against parameter changes is an inherent property (Subsection 6.4.3). Furthermore, we compare our approach to an online Jacobian Pseudo-Inverse (JPI) algorithm. Finally, we demonstrate that our approach can be easily transferred from the simulation to the real robot without any special precautions (Subsection 6.4.5).

All simulations were implemented in the robot simulation software Webots (Michel, 2004). A detailed model of the dynamics of the HOAP-2 robot, based on data provided by the vendor Fujitsu, was used. The basic simulation time step was set to 2 ms and the time steps for the control loops were set to 8 ms. In the general setup we had two kinematic synergies ($\mathbf{\Phi}_x$ and $\mathbf{\Phi}_z$), which were used within two separate control loops. They reacted independently from each other on their corresponding output dimension x and z. In dependence on their errors e_x and e_z, both linear controllers calculated the velocities $\dot{s}_x$ and $\dot{s}_z$ of their KS-parameters. The velocities were integrated numerically to obtain s_x and s_z, which were then mapped via the look-up table into joint angle offsets. Subsequently, these joint angle offsets were linearly combined as described in Subsection 6.3.1 to get the actual joint target angles. Finally, these angles were transformed into torques by local PD controllers[7] at the servos.

[7]Note that these are the hardware controller of the servos and not the controllers from our proposed control loops.



(a) tilt angle Θ_x (b) tilt angle Θ_z (c) x-coordinate of the $mCoP$

(d) z-coordinate of the $mCoP$ (e) error signal e_x (f) error signal for e_z

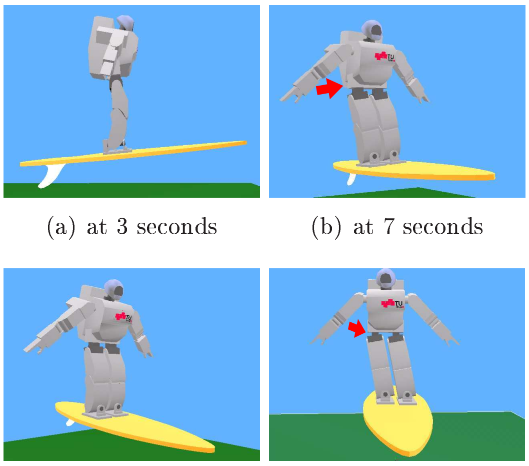
(g) activation s_x of KS $\mathbf{\Phi}_x$ (h) activation s_z of KS $\mathbf{\Phi}_z$

Figure 6.10: Result for the experiment with a moving support platform (surfboard) and unexpected external forces (wind) $W1$ and $W2$. The balance of the HOAP-2 is controlled by two linear controllers combined with the kinematic synergies. Without balance control (red dashed line in **(c)** and **(d)**) the $mCoP$ left the support polygon after 16s (in response to the wind $W2$), and the robot fell over. With balance control (solid lines) the stability of the robot was maintained in spite of unexpected external forces.

We provide supplementary multimedia material in form of two videos, available at http://ieeexplore.ieee.org. The first one (*simulation_videos.avi*) shows all simulated experiments of the following sections. The second video (*real_robot_videos.avi*) shows the experiments with the real robot. Both videos (in compressed form) are about 13 MB in size.

6.4.1 Moving Support Platform (Surfboard Task)

In this task we simulated the HOAP-2 robot standing on a movable support platform (surfboard). The surfboard could rotate about the x-axis with angle Θ_x and about the z-axis with the angle Θ_z. Typical scenarios of the setup can be seen in Figure 6.11.



(a) at 3 seconds (b) at 7 seconds

(c) at 12 seconds (d) at 16 seconds

Figure 6.11: Screenshots of the posture of the (simulated) HOAP-2 at 4 time points during the balancing experiment with the random moving support surface (surfboard) and external perturbations (winds). In Figure **(b)** the wind $W1$ was blowing from the right (point of view of the robot ; red arrow). As a consequence, the robot was leaning against the wind in order to move its $mCoP$ back into the middle of the support polygon. In Figure **(d)** another wind $W2$ was blowing from the right and the back (red arrow), resulting in a diagonal force. Again, the robot responded properly to this online modification of its dynamic model.

We considered the case where the surfboard was tilted dynamically in random directions. The random trajectories for the angles Θ_x and Θ_z were generated independently from each other by smoothing (by the use of a discrete low-pass FIR-filter[8]) random trajectories of jumps (steps) with random amplitudes and random durations. Typical resulting trajectories are presented in Figures 6.10(a) and 6.10(b).

In addition to the random movement of the surfboard, unforeseen external forces (for example wind forces or contact with other objects) were applied to the torso of the robot at various points in time. We designed this scenario in order to show that our proposed approach is able to deal with different kinds of external perturbations simultaneously. Furthermore, control strategies that require knowledge of the dynamic model of the robot are inapplicable in this scenario, because the external forces change the dynamic model of the robot in an unknown, online manner. Figure 6.10 shows the results when an external force $W1 = [0, 0, 5]^T$N (a force from the right side) was applied at the torso of the robot during the interval $[5\text{s}, 10\text{s}]$, and another external force $W2 = [5, 0, -5]^T$N (a force from the right and the back) was applied during the interval $[15\text{s}, 20\text{s}]$ (we shaded these two time intervals in gray). Note that the onsets of the winds were abrupt (i.e., a step function in time) and therefore represented highly dynamical perturbations to the system.

Typical trajectories of the $mCoP$ for the described setup, with and without balance control, are shown in Figures 6.10(c) and 6.10(d). Without balance control, the robot lost balance after 16s (indicated by a black star in Figures 6.10(c) and 6.10(e)), whereas with our controllers, balance was maintained. The error signals

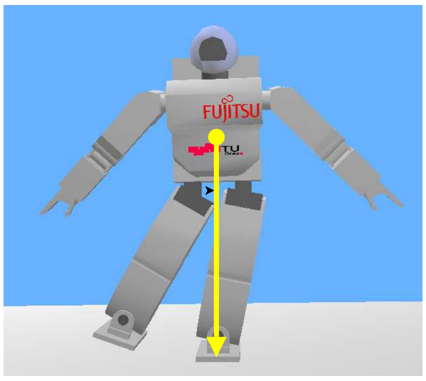
[8]The used FIR-filter had three poles at 0.997.

for both dimensions x and z can be seen in Figures 6.10(e) and 6.10(f). Note that both perturbations, the movements of the surfboard and the external forces, are *external perturbations*. In addition, as the setup was dynamic, inherent *model perturbations* were also always present. With this experiment, we demonstrated that our approach is able to react online against a mixture of different types of unforeseen perturbations.

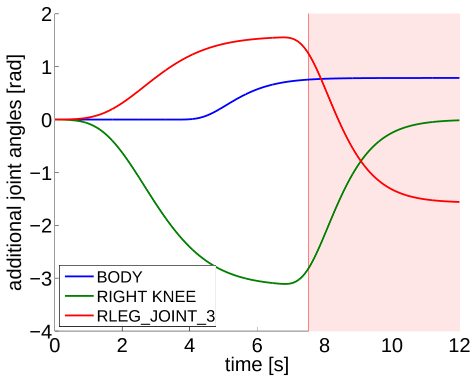
6.4.2 Kinematic Synergies in Single Support

In this experiment we demonstrate how to apply our approach in single support. We used two different strategies. The first strategy reused the *KSs* previously calculated for double support (referred to as *DS-KS*). We switched off the output of the control loop for the joints of the lifted leg and set the desired *mCoP* position to the center of the reduced support polygon (defined by the single supporting foot). The second strategy was to design new *KSs* for single support (referred to as *SS-KS*). We used the same procedure as described previously in Subsection 6.3.1, with the distinction, that we used a different initial position (the one shown in Figure 6.12(a)) and we only optimized the joint angles of the supporting leg.

In the experimental setup the robot stood only on its left foot. The right foot had no contact to the ground and therefore the right leg was free to perform any desirable movement. In our example the robot is supposed to perform a kick motion. The initial posture can be seen in Figure 6.12(a). The corresponding s-values for this posture were $s_x = 0$ and $s_z = 0.195$ for *DS-KS* and $s_x = s_z = 0$ for *SS-KS*.
In order to demonstrate the validity of both strategies, we moved the body joint and the hip joints of the left leg (these joints were not under the control of the *KS*s) in order to perform a kick motion, which also included the upper trunk (see Figure 6.12(b)). For the robot this movement represented an *internal perturbation* as discussed in Subsection 6.3.3. When no balancing control was active, after about 7.5s of simulation time, the robot tipped over and fell. With the controllers switched on, the robot was able to keep balance during the kick motion (in both cases, *SS-KS* and *DS-KS*). Figure 6.13 shows the time course from 2s to 12s of this experiment with *DS-KS*. Similar results were obtained with *SS-KS*. Figures 6.14(a) and 6.14(b) show the trajectories of the *KS*-parameters s_x and s_z. Note that in the case of *DS-KS,* there was an offset at the beginning of the simulation for the *KS*-parameter s_z. This reflects the offset of the initial posture for single support from the original initial posture for double support. Figures 6.14(c) and 6.14(d) present the errors during the simulation. The controllers counteracted the disturbances correctly and kept the errors close to zero for both strategies. The dashed red curve shows the errors when no controllers were activated. Note that the scales of the y-axes of the plots in Figure 6.14 are different for the dimensions x and z. This is a consequence of the used kick motion which mostly affected the *mCoP* in the x direction (forward/backward). Both strategies (*DS-KS* and *SS-KS*) showed a similar performance (see Figures 6.14(c) and 6.14(d)). As a consequence, we can see that the *KS*s can also be used for different, albeit related tasks, for which, in the first place, they have not been designed for. This might also help to reduce the number of needed *KSs* in real world applications, because related tasks might share the same set of *KS*s.



(a) initial starting posture



(b) joint trajectories for the kick motion

Figure 6.12: Setup for the single support task. Figure **(a)** shows the initial posture. The yellow circle denotes the CoM of the robot and the arrow points to the pCoM, which is located at the center of the support polygon. Figure **(b)** shows the joint angle trajectories which were used for the kick motion. When no balance control was applied, the robot lost balance and tipped over at about $7.5s$.

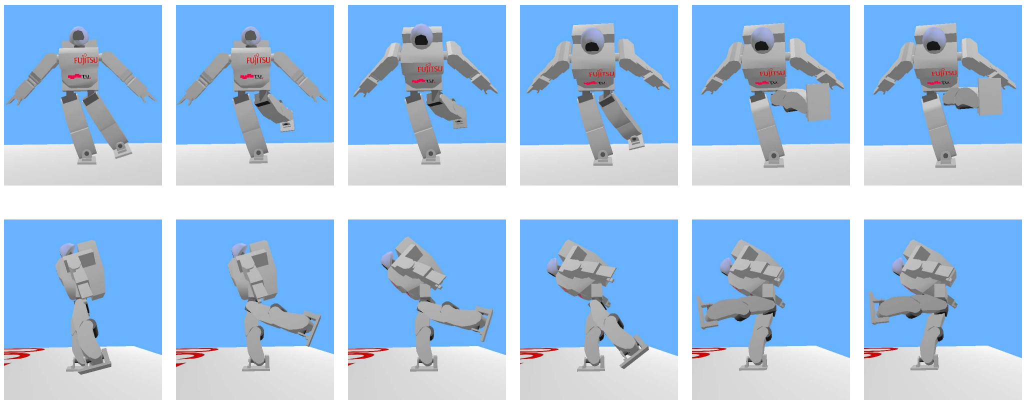
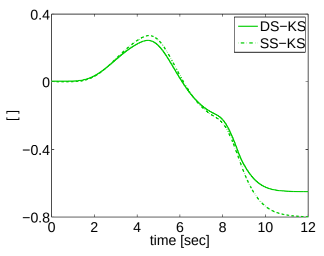


Figure 6.13: Postures of the simulated HOAP-2 for the single support task. The first row shows the robot from the front and the second row shows the robot from the side. Screenshots were taken every second from the 2nd to the 12th second.

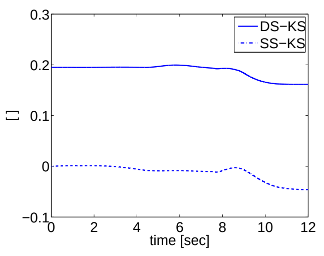
6.4.3 Robustness to Changes in the Model of the Robot and the Controller Parameters

The kinematic synergies are based on the static model of the robot. Since uncertainties in the model parameters (lengths and masses) are common, it is desirable to have a framework that is robust to changes in those parameters. Moreover, such a robustness simplifies a transfer from the simulation to a real robot. In addition, it would be beneficial to have a wide range of valid control parameters, i.e., K_P and K_D, which are able to balance the robot. In the following experiments we demonstrate that our proposed setup is widely robust to variations of these parameters.

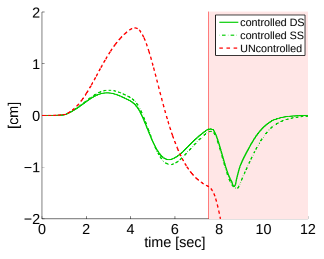
In a first experiment we varied the size of the robot by changing the length of every link by a multiplicative *length factor*, ranging from 0.5 to 2.5. We used the



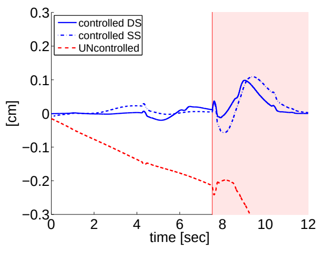
(a) KS-parameter s_x



(b) KS-parameter s_z



(c) error e_x



(d) error e_z

Figure 6.14: Results for the single support task. The left column shows the results for the x-dimension ($\mathbf{\Phi}_x$, forward/backward) and the right column the results for the z-dimension ($\mathbf{\Phi}_z$, left/right). The Figures **(a)** and **(b)** show the responses of the KS-parameter s_x and s_z for both approaches (*SS-KS* and *DS-KS*). Figures **(c)** and **(d)** show the errors. The red dashed curves denote the errors when no balance control was active. In this case the beginning of the red region indicates, when the robot tipped over and lost balance.

trajectory following task from our previous work (Hauser et al., 2007), where the robot had to follow a figure eight trajectory with its *mCoP*, while it manipulated a heavy weight. The *KSs* were kept constant. First, we used the same controller parameters for both controllers as in the original task ($K_P = 80$ and $K_D = 0.1$). The robot was able to keep balance for a *length factor*, which ranged from 0.85 to 1.1. In Figure 6.15 the mean squared errors for the x-dimension[9] for successful *length factors* (the robot kept balance) are indicated by red circles for these controller parameters. In order to demonstrate how to improve robustness, we increased the response time of the controllers by setting the controller parameters to $K_P = 50$ and $K_D = 0.0$. In this case, successful *length factors* ranged from 0.85 to 1.45 (indicated by blue crosses in Figure 6.15). Note that the mean squared error only increased slightly. We also tested an even slower controller ($K_P = 20$ and $K_D = 0.0$), which resulted in a fairly large range from 0.7 to 2.25 (indicated by green triangles in Figure 6.15). However, the used controller was too slow to follow the desired trajectory, which can be seen in the high mean squared error values. The corresponding *mCoP* trajectories of all three controllers can be seen in the right plots of Figure 6.15. The black lines are the target trajectories. The conclusion of the experiment is that the proposed setup is robust to changes in the lengths of the robot. In addition, the results suggest that there is a tradeoff between the robustness of the approach and the response times of the controllers. Similar results were obtained, when the

[9]Similar plots were obtained for the z-dimension.

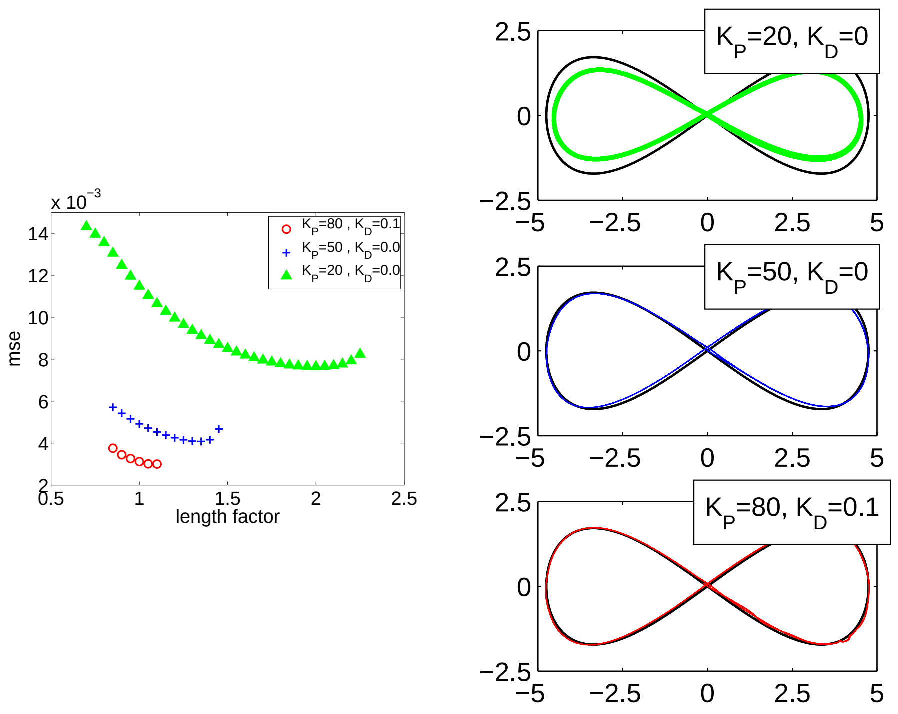


Figure 6.15: Results on robustness to changes in the lengths. The lengths of all links were multiplied by a *length factor*. The plot shows three different settings for the controller parameters, resulting in different response times of the controllers. The red circles show the mean squared error (mse) for the controller ($K_P = 80$ and $K_D = 0.1$) with the shortest response time. The red circles are shown for the range of successful *length factors* (the robot kept balance) from 0.85 to 1.1. By increasing the response time (controller parameters were set to $K_P = 50$ and $K_D = 0.0$.) the range (from 0.85 to 1.45) of successful *length factors* and therefore the robustness of our approach could be increased. However, also the mse increased slightly, which indicates a worse tracking performance. With an even longer response time ($K_P = 20$ and $K_D = 0.0$), the region of successful *length factors* (from 0.7 to 2.25) also grows, however, the controller was no longer able to follow the desired trajectory (indicated by the large mse values). The results point to the fact, that there is a tradeoff between the robustness of the approach and the response time of the controller. The right plots show the corresponding *mCoP* trajectories for the three controllers (at a length factor $= 1$).

masses as well the lengths were changed simultaneously to simulate growing.

In a second experiment we provide an evaluation of the robustness of our approach to the choice of the controller parameters. We used the single support task described in Subsection 6.4.2 (using the previously described *SS-KS*) and varied the K_P and K_D parameters over several decades. We evaluated which parameter settings (K_P/K_D-pairs) were successful, i.e., the robot was able to keep balance. The results can be seen in Figure 6.16. Successful parameter settings are highlighted in green. Note that the region of successful settings ranges over two decades for both parameters. This suggests that our approach is robust to the choice of the controller parameters and, thus, appropriate parameters are easily found. Moreover, this robustness potentially allows us to combine our approach with adaptive con-

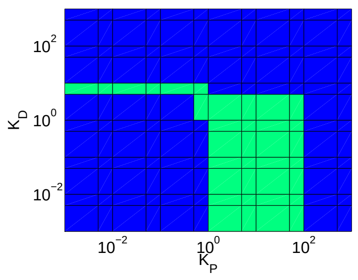


Figure 6.16: Region of evaluated controller parameters for the single support task described in 6.4.2. Successful parameter settings (for which the robot was able to keep balance) are highlighted in green. Note that the scales of the axes are logarithmic. The region of successful controller parameters ranges over two decades for both parameters, indicating that our approach is robust to the choice of the control parameters.

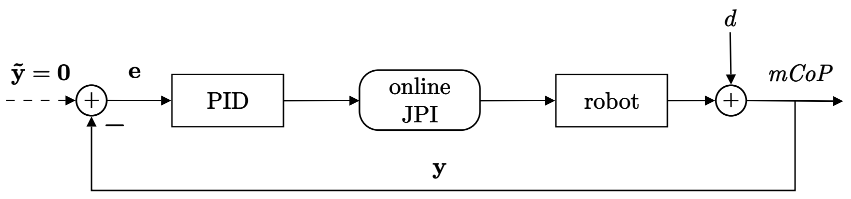


Figure 6.17: Schematic setup of the online Jacobian Pseudo-Inverse (JPI) approach, to which we compared our approach (Figure 6.9). Instead of fixed kinematic synergies this approach has to run online an optimization process (based on a JPI) at every single time step to calculate the optimal joint angles velocities.

trol (Astrom and Wittenmark, 1995) or online policy search methods (Kober et al., 2008).

6.4.4 Comparison to an Online Jacobian Pseudo-Inverse Approach

We performed a comparison of our kinematic synergy setup to an online Jacobian Pseudo-Inverse (JPI) approach (Sciavicco and Siciliano, 2005). This approach performed online an optimization similar to the one we used for the offline construction of the *KSs*. In order to be responsive to external perturbations and model uncertainties we had to plug the JPI into a feedback control loop. Figure 6.17 shows the considered setup. In order to compare both approaches the robot had to track a rectangular trajectory (with rounded edges) centered at the center of the support polygon. We systematically increased the size of the rectangle and the speed of the trajectory and compared the maximum quantities, at which the robot tipped over. The differences between the two approaches for both limits (rectangle size and speed) were less than 1%. Hence, there was no significant difference in their performances. This suggests that the complex Jacobian Pseudo-Inverse computations can be performed offline (in order to construct the *KSs*) without a significant loss of performance. Note that the JPI approach needs to apply online sophisticated, time intensive calculations, while our approach is based on a much simpler control law using only a PID controller. A comparison of the online computation time of both approaches revealed a speed-up factor of 80 in favor of our approach. The results also show that the performance loss due to the linear superposition[10] of the two *KSs* is negligible for humanoid balancing.

6.4.5 Experiments with a Real HOAP-2 Robot

In our final experiment we transferred our approach to a real HOAP-2 robot. Due to the previously demonstrated robustness against model uncertainties, we were able to simply reuse the same *KSs* as in our simulations, even though the static model used for the *KSs* did not perfectly match the static model of the real robot.

We investigated two different setups. In the first setup the robot stood on the floor (denoted by **F**) and we applied external forces. This was done by applying an almost constant force from different directions for approximately 1 to 2 seconds by pushing the robot. In the second setup (denoted by **P**) we reproduced the surfboard task. The robot stood on a movable platform, which was mounted on a plastic sphere in order to resemble the surfboard with its two degrees of freedom. In contrast to the simulated experiment, no additional external forces (winds) were used (only the movement of the platform represented an external force). Note that in both setups the robot had no knowledge about the onset times, the directions or the amplitudes of the applied external forces.

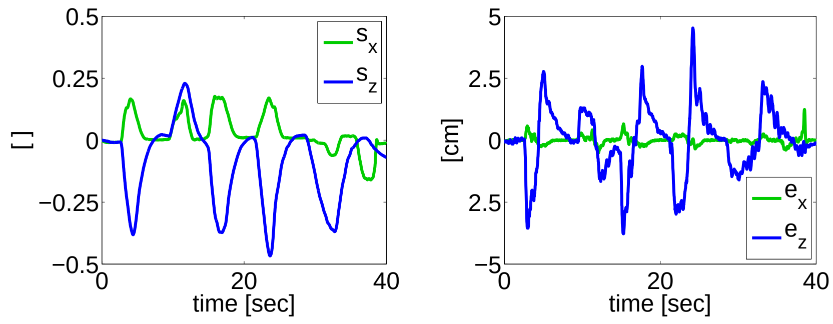
The first row of Figure 6.19 shows the responses of the robot to pushes from different directions (setup **F**). The second row shows responses of the robot to different movements of the supporting platform (setup **P**). The robot counterbalanced the applied external forces in order to keep its *mCoP* at the middle of the support polygon in each of these cases.

In Figure 6.18 we show typical *KS*-parameters and the error signals recorded while the robot was pushed from different directions (in setup **F**). Note that, except for a short time period after a change of the applied external force, the error was kept close to zero. This indicates that the robot always tried to maintain its *mCoP* at the center of the support polygon. Note that videos of the experiments are provided in the additional multimedia file, available at http://ieeexplore.ieee.org/.

6.5 Conclusion

We have presented a new approach to transfer spatial movement representations coming from experimental data analysis such as synchronous muscle synergies to robot control. We used the approach for balance control of a humanoid robot. We have formalized the concept of a kinematic synergy (*KS*) that resembles the concept of a muscle synergy in physiology, and which reduces the dimensionality of the action space of the robot. We have shown that two kinematic synergies can be constructed for balance control of the humanoid robot HOAP-2 in such a way that their superposition is almost linear (like in biological paradigms), although each *KS* itself is highly nonlinear. Based on this concept we were able to demonstrate that it is possible to move the time intensive calculations of the optimization process

[10]Note that the JPI approach does not use a linear superpositions, but rather simultaneously optimize for both output dimensions, i.e., $y \in \mathbb{R}^2$.



(a) *KS*-parameters s_x and s_z (b) errors e_x and e_z

Figure 6.18: The *KS*-parameters and the errors signals recorded during an experiment with the real HOAP-2 robot. The robots was pushed from different directions (setup **F**). The left figure shows the *KS*-parameters and the right figure shows the corresponding error signals. We can see that, except for a short time period after a change of the applied external force, the error is kept close to zero. This indicates that the robot always tried to maintain its *mCoP* at the center of the support polygon.

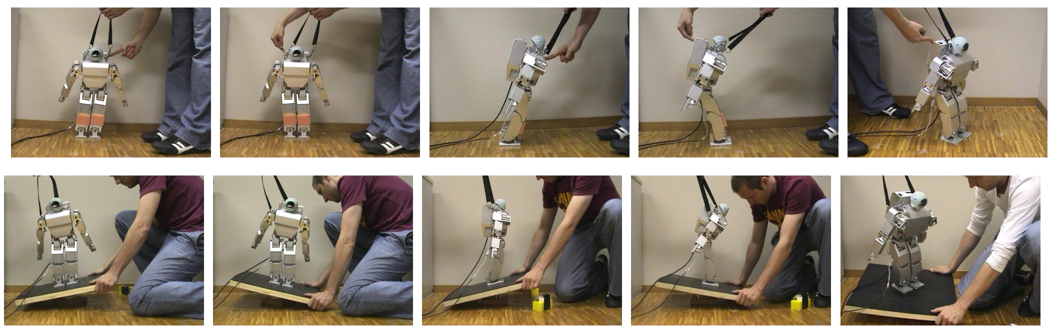


Figure 6.19: Top row: Resulting responses of the HOAP-2 to external forces. The screenshots were made during dynamic action. The top row shows screenshots for experiments while standing on the floor (setup **F**). Bottom row: External forces were applied by pushes. The second row shows screenshots of experiments with the robot standing on a movable platform (setup **P**). External forces were applied by moving the platform. In any of these situations the robot acted correctly and moved its *mCoP* to the desired position at the center of the support polygon. Note that there are videos of the experiments available (at http://ieeexplore.ieee.org/).

offline and therefore keep the needed online calculations simple and fast. We have demonstrated, both through computer simulations and through experiments with the real robot HOAP-2, that this strategy makes it possible to virtually reduce the highly nonlinear balance control problem of the robot to a linear control problem (as long as the required movements are not too fast).

We showed that, in contrast to other approaches, which are based on an exact dynamic model of the robot, our proposed combination of *KSs* and linear controllers enables a humanoid robot to counterbalance unknown external forces of different kinds. Additionally, we showed that robustness to parameter changes in the model as well to changes in the controller parameters is an inherent property of the proposed approach. Based on this robustness we were able to transfer in straightforward manner this new approach for balance control from a simulated to a real HOAP-2 robot.

We expect that both, the drastic dimensionality reduction of the action space and the resulting linearization of the robot control through the use of suitable *KSs*, pave the way for future learning-based solutions to movement control problems for humanoid robots.

6.6 Acknowledgments

This chapter is based on the publication '*Biologically Inspired Kinematic Synergies enable Linear Balance Control of a Humanoid Robot.*' (Hauser et al., 2011). The paper was written by Helmut Hauser (HH), Gerhard Neumann (GN), Auke Ijspeert (AJ) and Wolfgang Maass (WM). HH conducted the experiments and implemented the linear control laws while GN implemented the synergies including the inverse kinematics optimization.

Chapter 7

Learning Complex Motions by Sequencing Simpler Motion Templates

Contents

Abstraction of complex, longer motor tasks into simpler elemental movements enables humans and animals to exhibit motor skills which have not yet been matched by robots. Humans intuitively decompose complex motions into smaller, simpler segments. For example when describing simple movements like drawing a triangle with a pen, we can easily name the basic steps of this movement.

Surprisingly, such abstractions have rarely been used in artificial motor skill learning algorithms. These algorithms typically choose a new action (such as a torque or a force) at a very fast time-scale. As a result, both policy and temporal credit assignment problem become unnecessarily complex - often beyond the reach of current machine learning methods.

We introduce a new framework for temporal abstractions in reinforcement learning (RL), i.e. RL with motion templates. We present a new algorithm for this framework which can learn high-quality policies by making only few abstract decisions. This is the first algorithm which allows efficient sequencing of movement primitive representations.

7.1 Introduction

Humans use abstractions to simplify the motor tasks occurring during their daily life. For example when describing simple movements like drawing a triangle with a pen, we can easily name the basic steps of this movement. In a similar manner, many complex movements can be decomposed into smaller, simpler segments. This sort of abstraction is for example often used by engineers for designing hybrid control solutions (Xu and Antsaklis, 2002) where the single segments are implemented as local, linear continuous controllers. We will call these building blocks *motion templates*. Other names that can be found in the literature are “motion primitives”, “movement schema’s”, “basis behaviors” or “options” (Ijspeert and Schaal, 2003; Arbib, 1981; Dautenhahn and Nehaniv, 2002; Sutton et al., 1999).

Motor skill learning is a challenging problem for machine learning and, in particular, for the subfield of reinforcement learning (RL). Primarily used in motor skill learning is the flat RL setting without the use of abstractions. In this setting the agent has to choose a new action (typically a motor force or torque) at a very small sampling frequency. While this allows the representation of arbitrary policies, this flexibility makes the learning problem so complex that it is often beyond the reach of current methods. A common approach for limiting the potential complexity of the policy in the flat RL setting is to use a parametrized policy. Ijspeert et al. (Ijspeert and Schaal, 2003) introduced a special kind of parametrized policies called *motion primitives*, which are based on dynamical systems. In most applications to date, only a single *motion primitive* is used for the whole movement. Parametrized policy search methods such as policy gradient descent and EM-like policy updates (Kober and Peters, 2010) have been used in order to improve single-stroke motor primitives.

Currently, only few abstractions are used in RL algorithms for continuous environments, with few exceptions such as (Huber and Grupen, 1998; Ghavamzadeh and Mahadevan, 2003). In (Huber and Grupen, 1998) the policy acquisition problem is reduced to learning to coordinate a set of closed loop control strategies. In (Ghavamzadeh and Mahadevan, 2003) the given task is manually decomposed into a set of subtasks. Both, the lower-level subtasks and the higher-level subtask-selection policies are learned. In all these approaches the structure for the hierarchy of abstraction is manually designed and fixed during learning which limits the generality of these approaches. In our approach, an arbitrary parametrization of the abstracted level can be learned.

In this paper, we introduce a new framework for abstraction in RL, i.e. RL with motion templates. Motion templates are our building blocks of motion. A template m_p is represented as parametrized policy and executed until its termination condition is fulfilled. We assume that the functional forms of the motion templates remain fixed, and thus, our task is to learn the correct order and parameters of the motion templates by reinforcement learning. As motion templates are temporally extended actions, they can be seen as parametrized *options* in continuous time. There are a few well-established learning algorithms for the options framework (Sutton et al., 1999). However, these algorithms are designed for discrete environments.

Choosing the parameters of a motion template is a continuous-valued decision. However, a single decision has now much more influence on the outcome of the whole motion than in flat RL. Thus, the decisions have to be made more precisely, though, the overall learning problem is simplified because much fewer decisions are needed to fulfill a task. As RL in continuous action spaces is already challenging in the flat RL setting, the requirement of learning highly-precise policies has limited the use of this sort of abstraction for motor control learning.

This paper introduces a new algorithm which satisfies this requirement and therefore permits learning at an abstract level. The algorithm is based on the Locally-Advantage WEighted Regression (LAWER) algorithm. LAWER is a fitted Q-Iteration (Ernst et al., 2005) based algorithm which has been shown to learn high-quality continuous valued policies for many flat RL settings (Neumann et al., 2009). However, two substantial extensions are needed to render motion template learning possible. Firstly, we propose an improved estimation of the goodness of an state action pair. Secondly, we introduce an adaptive kernel, which is based on randomized regression trees (Ernst et al., 2005).

We conduct experiments on 3 different tasks, a 1-link and a 2-link pendulum swing-up task and also a 2-link balancing task.

7.2 Motion Templates

A motion template m_p is defined by its k_p dimensional parameter space $\mathcal{W}_p \subseteq \mathcal{R}^{k_p}$, its parametrized policy $u_p(\mathbf{s}, t; \mathbf{w}_p)$ ($\mathbf{s}$ is the current state, t represents the time spent executing the template and $\mathbf{w}_p \in \mathcal{W}_p$ is the parameter vector) and its termination condition $c_p(\mathbf{s}, t; \mathbf{w}_p)$.

At each decision-time point σ_k, the agent has to choose a motion template m_p from the set $\mathcal{A}(\sigma_k)$ and also the parametrization $\mathbf{w}_p$ of m_p. Subsequently the agent follows the policy $p_p(\mathbf{s}, t; \mathbf{w}_p)$ until the termination condition $c_p(\mathbf{s}, t; \mathbf{w}_p)$ is fulfilled. Afterwards, we obtain a new decision-time point σ_{k+1}.

The functional forms of the policy $u_p(\mathbf{s}, t; \mathbf{w}_p)$ and the termination condition $c_p(\mathbf{s}, t; \mathbf{w}_p)$ are defined beforehand and can be arbitrary functions. For example, consider again the task of drawing a triangle. We can define a motion template m_{line} for drawing a line with the endpoint of the line and the velocity of moving the pen as parameters. The policy u_{line} moves the pen from the current position with the specified velocity in the direction of the endpoint of the line. The template is terminated when the pen has reached a certain neighborhood of the endpoint.

In our experiments, sigmoidal functions and linear controllers are used to model the motion templates.

7.2.1 Reinforcement Learning with Motion Templates

Each motion template is a temporally extended, continuous valued action. Thus, we deal with a continuous-time Semi-Markov Decision Process (SMDP). We will review only the relevant concepts from the continuous-time SMDP framework. For a detailed definition, please refer to (Bradtke and Duff, 1995).

Unlike in standard Markov Decision Processes (MDPs), the transition probability function $P(s', d|s, a)$ is extended by the duration d of an action. The Bellman equation for the value function $V^\pi(s)$ of policy π is given by

$$\begin{aligned} V^\pi(s) = \int_a \pi(a|s) \, (r(s,a) + \\ \int_{s'} \int_{t=0}^{\infty} \exp(-\beta t) P(s', t|s, a) V^\pi(s') dt ds' \Big) \, da, \end{aligned} \tag{7.1}$$

where β is the discount factor[1]. The action value function $Q^\pi(s,a)$ is given by

$$\begin{aligned} Q^\pi(s,a) &= r(s,a) + \\ &\int_{s'}\int_{t=0}^{\infty} \exp(-\beta t) P(s',t|s,a) V^\pi(s') dt ds'. \end{aligned} \tag{7.2}$$

A policy is now defined as $\pi(m_p, \mathbf{w}_p|s_k)$. It can be decomposed into $\pi(m_p|s_k)\pi_p(\mathbf{w}_p|s_k)$, where $\pi(m_p|s_k)$ is the template selection policy and $\pi_p(\mathbf{w}_p|s_k)$ is the policy for selecting the parameters of template m_p.

7.3 Fitted Q-Iteration

As LAWER is a Fitted Q-iteration (FQI) (Ernst et al., 2005; Riedmiller, 2005) based algorithm we quickly review the relevant concepts. FQI is a batch mode reinforcement learning (BMRL) algorithm. In BMRL algorithms we assume that all the experience of the agent up to the current time is given in the form $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i >\}_{1\le i\le N}$. FQI estimates an optimal control policy from this historical data. Therefore it approximates the state-action value function $Q(\mathbf{s},\mathbf{a})$ by iteratively using supervised regression techniques. New target values for the regression are generated by

$$\begin{aligned} \tilde{Q}_{k+1}(i) &= r_i + \gamma V_k(\mathbf{s}'_i), \\ &= r_i + \gamma \max_{\mathbf{a}'} Q_k(\mathbf{s}'_i, \mathbf{a}'), \end{aligned} \tag{7.3}$$

which are subsequently used to learn the Q-function $Q_{k+1}(\mathbf{s},\mathbf{a})$. For more details please refer to (Neumann et al., 2009).

7.3.1 Fitted Q-Iteration for SMDPs

For SMDPs we have to include the duration d_i of each action to our historical data $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, d_i, \mathbf{s}'_i >\}_{1\le i\le N}$. Instead of using Equation 7.3, new Q-values can now be calculated by

$$\tilde{Q}_{k+1}(i) = r_i + \exp(-\beta d_i) \max_{\mathbf{a}'} Q_k(\mathbf{s}'_i, \mathbf{a}'). \tag{7.4}$$

7.3.2 Locally-Advantage-WEighted Regression (LAWER)

A severe problem when using fitted Q-iteration for continuous action spaces is the use of the greedy operation $V_k(\mathbf{s}) = \max_{a'} Q_k(\mathbf{s}, \mathbf{a}')$ which is hard to perform. LAWER (Neumann et al., 2009) is a variant of FQI which avoids this max operator and is therefore well suited for continuous action spaces. The algorithm has been shown to learn high quality policies for many flat RL settings.

Instead of using the max operator, a soft-max operator is used which can be efficiently approximated by an advantage-weighted regression. The advantage-weighted

[1]In order to achieve the same discounting rate as in a flat MDP, β can be calculated from the relation $\gamma = \exp(-\beta\Delta t)$, where γ is the discount factor and Δt is the time step of the flat MDP.

regression solely uses the given state action pairs $(\mathbf{s}_i, \mathbf{a}_i)$ to estimate the V-function and therefore avoids an exhaustive search in the action space. State-action pairs with an higher expected advantage[2] have a higher influence on the regression.

The regression uses the state vectors $\mathbf{s_i}$ as input dataset, the Q-values $\tilde{Q}_{k+1}(i)$ as target values and an additional weighting u_i for each data point. The authors proved that the result of the advantage-weighted regression is an approximation of the V-function $V(\mathbf{s}) = \max_{a'} Q_k(\mathbf{s}, \mathbf{a}')$. The weighting u_i can be seen as goodness of using action $\mathbf{a}_i$ in state $\mathbf{s}_i$. It is estimated by $u_i = \exp(\tau \bar{A}(\mathbf{s}_i, \mathbf{a}_i))$, where $\bar{A}(\mathbf{s}_i, \mathbf{a}_i)$ denotes the normalized advantage function and the parameter τ sets the greediness of the soft-max operator. We skip the description of the normalization of the advantage function, because, for this paper, it is enough to know that the normalization, and also the proof of the algorithm, assume normally distributed advantage values. For a more detailed description of $\bar{A}(\mathbf{s}_i, \mathbf{a}_i)$ please refer to (Neumann et al., 2009).

LAWER uses Locally Weighted Regression (LWR, by Atkeson et al., 1997) for approximating the Q and the V-function. It therefore needs to be able to calculate the similarity $w_i(\mathbf{s})$ between a state $\mathbf{s}_i$ in the dataset H and state $\mathbf{s}$. The state similarities $w_i(\mathbf{s})$ can be calculated by a Gaussian kernel $w_i(\mathbf{s}) = \exp(-(\mathbf{s}_i - \mathbf{s})^T \mathbf{D}(\mathbf{s}_i - \mathbf{s}))$. In this paper we also introduce an adaptive kernel in Section 7.4.1. For simplicity, we will denote $w_i(\mathbf{s}_j)$ as w_{ij} for all $\mathbf{s}_j \in H$.

Standard LWR is used to estimate the Q-function. The V-function is approximated by a combination of LWR and advantage-weighted regression. In order to do so, the advantage weighting u_i is multiplicatively combined with the state similarity weighting, resulting again in a standard weighted linear regression. For the exact equations, please refer to (Neumann et al., 2009).

The optimal policy $\pi(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{a}|\mu(\mathbf{s}), \Sigma(\mathbf{s}))$ is modelled as stochastic policy with Gaussian exploration. The mean $\mu(\mathbf{s})$ can be determined by a similar locally and advantage-weighted regression, just the actions $\mathbf{a}_i$ are used as targets instead of the Q-values. The covariance matrix $\Sigma(\mathbf{s})$ is given by calculating the advantage-weighted covariance of locally neighbored actions.

Intuitively speaking, the V-function is calculated by interpolating between the Q-values of locally neighbored state action pairs, but only examples with a high goodness u_i (i.e. high normalized advantage value) are used. The same is true for the policy, we just interpolate between the action vectors.

7.4 Fitted Q-iteration for Motion Templates

In order to apply the LAWER algorithm to the motion template framework we use a separate dataset H^p and individual estimations Q^p and V^p of the Q and V-function for each motion template m_p. The functions V^p and Q^p represent the state and state-action value function when choosing motion template m_p in the first decision and subsequently following the optimal policy. We implement the template selection policy $\pi(m_p|\mathbf{s}_k)$ by a soft-max policy. The overall value function is determined by $V(\sigma_k) = \max_{m_p \in \mathcal{A}(\sigma_k)} V^p(\sigma_k)$. LAWER is used to learn the single Q and V-function estimates Q^p and V^p.

[2]The advantage function is given by $A(\mathbf{s}_i, \mathbf{a}_i) = Q(\mathbf{s}_i, \mathbf{a}_i) - V(\mathbf{s}_i)$

In this section we present two extensions which improve the accuracy of LAWER and render learning with motion templates possible. Firstly, adaptive tree-based kernels are used to improve the estimation of the state similarities w_{ij}. This kernel also adapts to spatially varying curvatures of the regression surface and therefore needs an estimate of the V-function. Secondly, we show how to improve the estimate of the goodness u_i by the use of an additional optimization. Based on the current estimate of the state similarities w_{ij}, new u_i values, and subsequently also new estimates of the V-function are calculated. Both algorithms are applied intertwined to get improved estimates of w_{ij} and u_i.

7.4.1 Adaptive Tree-based Kernels

The use of an uniform weighting kernel is often problematic in the case of high dimensional input spaces ('curse of dimensionality'), spatially varying data densities or spatially varying curvatures of the regression surface. This problem can be alleviated by varying the 'shape' of the weighting kernel.

We use the Extremely Randomized Tree (Extra-Tree) algorithm (Ernst et al., 2005) to obtain a varying kernel function. This algorithm has been particularly successful for approximating the Q-function in FQI. We modify this approach to calculate the weighting kernel. The resulting kernel has the same properties as the Extra-Trees, and therefore adapts to the local state density as well as to the local curvature of the V-function.

The standard Extra-Tree algorithm builds an ensemble of regression trees. It has 3 parameters, the number M of regression trees, the number K of randomized splits to evaluate per node and the maximum number of samples per leaf $n_{\min}$. For more details about the algorithm please refer to (Ernst et al., 2005).

We use the trees for calculating the state similarities w_{ij} instead of approximating the Q-function. In order to do so, we learn the mapping from the states $\mathbf{s}_i$ to the V-values $V(\mathbf{s}_i)$ with the Extra-Tree algorithm. The kernel is then given by the fraction of trees in which two states $\mathbf{s}_i$ and $\mathbf{s}_j$ are located in the same leaf

$$w_{ij} = \frac{1}{M} \sum_{k=1}^{M} \text{isSameLeaf}(T_k, \mathbf{s}_i, \mathbf{s}_j), \tag{7.5}$$

where T_k is the kth tree in the ensemble and isSameLeaf is a function returning 1 if both examples are located in the same leaf and 0 otherwise. In our experiments we will show the superiority of the tree-based kernels to the Gaussian kernels.

7.4.2 Optimized LAWER

As already pointed out in Section 7.3.2, LAWER assumes normally distributed advantage values. Often this assumption does not hold or the normalization of the advantages is imprecise due to too few data points in the neighborhood. This effect is even more drastic if high τ values are used because the inaccuracies may result in low activations in areas with a low sample density and therefore also in inaccurate regressions. This restriction on the τ parameter also limits the quality of the estimated policy.

But how can we improve the estimation of the weightings u_i? Let us first consider a greedy policy π_D in a discrete environment. We formulate π_D as stochastic policy $u_{ij} = \pi_D(\mathbf{a}_j|\mathbf{s}_i)$. The u_{ij} can be found by solving the following constraint optimization problem

$$\begin{aligned} &\mathbf{u} = \arg\max_{\mathbf{u}} \textstyle\sum_{i,j} u_{ij} A(\mathbf{s}_i, \mathbf{a}_j) \\ &\text{subject to:} \quad \textstyle\sum_j u_{ij} = 1 \text{ for all states } \mathbf{s}_i \\ &\qquad\qquad\quad\; 0 \leq u_{ij} \leq 1 \text{ for all } i,\, j, \end{aligned} \tag{7.6}$$

where $\mathbf{u}$ is the vector of all u_{ij} and A is again the advantage function. In our setting, we also have a finite number of state-action pairs $(\mathbf{s}_i, \mathbf{a}_i)$, but typically all the states are different. However, the states are linked by the state similarities w_{ij}. The first constraint of the optimization problem can therefore be reformulated as

$$\sum_j w_{ij} u_j = 1 \text{ for all states } s_i, \tag{7.7}$$

while the remaining formulation of the optimization is unchanged. We also skipped the second index of u_{ij} because there is only one action for each state $\mathbf{s}_i$.

Due to this optimization we only use the u_i with the highest advantage values while ensuring that the summed activation $\sum_j w_{ij} u_j$ is high enough at each state $\mathbf{s}_i$ for applying an accurate weighted linear regression.

We solve the constraint optimization problem by maximizing the performance function C

$$\begin{aligned} C =& \frac{1}{Z} \sum_j u_j (Q(\mathbf{s}_j, \mathbf{a}_j) - V(\mathbf{s}_j)) - \\ & \lambda \sum_i \frac{(\sum_j w_{ij} u_j - \eta)^2}{\sum_j w_{ij}}, \end{aligned} \tag{7.8}$$

with $\eta = 1$, where Z is a normalization constant for the advantage values given by $Z = \sum_i |Q(\mathbf{s}_i, \mathbf{a}_i)|/N$. The second term of Equation 7.8 specifies the squared summed activation error for each state $\mathbf{s}_i$. It is normalized by the summed state-similarity of this state (i.e. $\sum_j w_{ij}$). This ensures that the activation error is equally weighted throughout the state space, independent of the local state density. We also introduced a new parameter λ which sets the tradeoff between maximizing the greediness of u_i or minimizing the summed activation error. It replaces the greediness parameter τ of the LAWER algorithm.

The function C can be maximized with respect to u_i using gradient ascent, the derivation of C is given by

$$\begin{aligned} \frac{dC}{du_k} =& \frac{1}{Z}(Q(\mathbf{s}_k, \mathbf{a}_k) - V(\mathbf{s}_k)) \\ & - 2\lambda \sum_i \frac{(\sum_j w_{ij} u_j - \eta)}{\sum_j w_{ij}} w_{ik}. \end{aligned} \tag{7.9}$$

The learning rate for the gradient ascent algorithm is always chosen such that the maximum change of an activation u_i is fixed to 0.01. After each gradient update the weights u_i are restricted to the interval $[0;1]$. The gradient ascent update is repeated for N_{opt} iterations, every $M_{opt} << N_{opt}$ iterations the value estimates $V(\mathbf{s}_i)$ are recalculated using the current weights u_i. When using the tree-based kernels, we also recalculate the state similarities w_{ij} with the new estimate of $V(\mathbf{s}_i)$. Typical values for N_{opt} and M_{opt} are 1000 and 100.

The covariance matrix of the exploration policy is also calculated slightly differently to the original LAWER algorithm. We require that always the best η_{exp} locally neighbored actions are used. We therefore use a separate set of advantage weightings u_{exp} for the covariance calculation which can be obtained by the same optimization defined in Equation 7.8, we just have to set η to η_{exp}. With η_{exp} we can scale the exploration rate of the algorithm.

7.5 Results

We evaluated the motion template approach on a 1-link and a 2-link pendulum swing-up task and a 2-link balancing task. For each task the immediate reward function was quadratic in the distance to the goal position $\mathbf{s}_G$ and in the applied torque/force, i.e., $r = -c_1|\mathbf{s} - \mathbf{s}_G|^2 - c_2|\mathbf{a}|^2$. For all our experiments we assume that the goal position $\mathbf{s}_G$ is known.

We collect L new episodes with the currently estimated exploration policy and one episode with the greedy policy (without exploration). After estimating the optimal policy, its performance is evaluated (without exploration) and the data collection is repeated. The initial distributions of the motion template parameters were set intuitively and were by no means optimal. We compared the motion template approach to flat RL with the standard LAWER algorithm.

7.5.1 Swing-Up Tasks

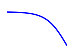
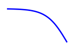
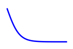
In this task a pendulum needs to be swung up from the position at the bottom to the top position.

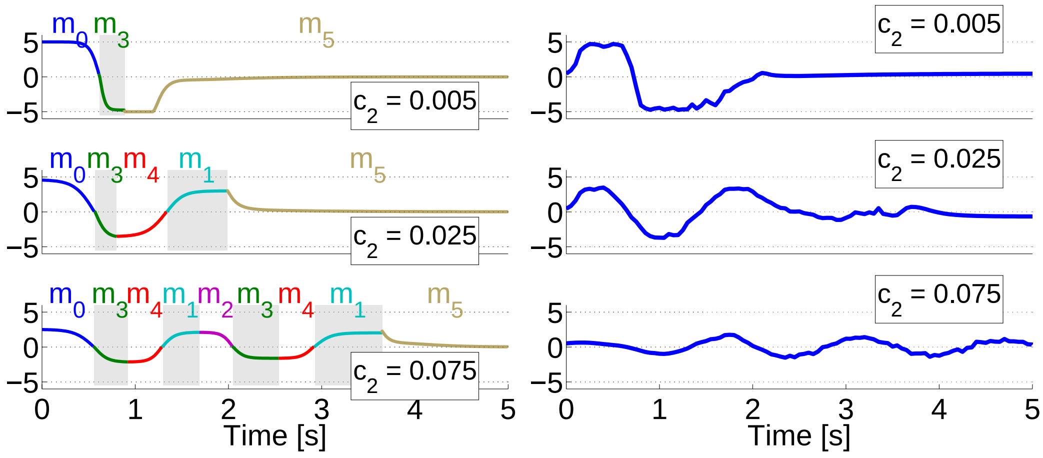
7.5.1.1 1-link Pendulum

The link of the pendulum had a length of 1m and a mass of 1kg, no friction was used. The used motion templates represent positive (m_1 and m_2) and negative peaks (m_3 and m_4) in the torque trajectory. There is also an individual template m_5 for balancing the robot at the top position. One peak consists of 2 successive motion templates, one for the ascending and one for the descending part of the peak.

The parametrization of the motion templates can be seen in Table 7.1. In order to form a proper peak, template m_2 and m_4 always start with the last torque u_t taken in the end of the previous template. Therefore parameter a_2 of these templates is already determined by u_t and consequently the outcome of template m_2 and m_4 depend on u_t. For this reason, the state space of template m_2 and m_4 was extended by u_t. The balancing template m_5 is implemented as linear PD-controller (see Table 7.1). The duration of the peak templates is an individual parameter of the templates

Table 7.1: MTs for the swing up motion. The functional forms resemble sigmoid functions. Parameter a_i coresponds to the height of the peak, o_i to the initial time offset and d_i to the duration of the motion template. k_1 and k_2 are the PD-controller constants of the balancer template. m_3 and m_4 resemble m_1 and m_2 except for a negative sign. The sketches illustrate the torque trajectories of these templates (x-axis: time, y-axis: acceleration).

MT	Functional Form	Parameters	Sketch
m_0	$a_0(1-\frac{2}{1+\exp(o_0-\frac{o_0}{d_0}t)})$	a_0, o_0, d_0	
$m_{1,3}$	$a_1(\frac{2}{1+\exp(-\frac{6o_1}{d_1}t)}-1)$	a_1, o_1, d_1	
$m_{2,4}$	$a_2(1-\frac{2}{1+\exp(o_2-\frac{o_2}{d_2}t)})$	o_2, d_2	
m_5	$-k_1\mathbf{w}-k_2\mathbf{w}'$	k_1, k_2	



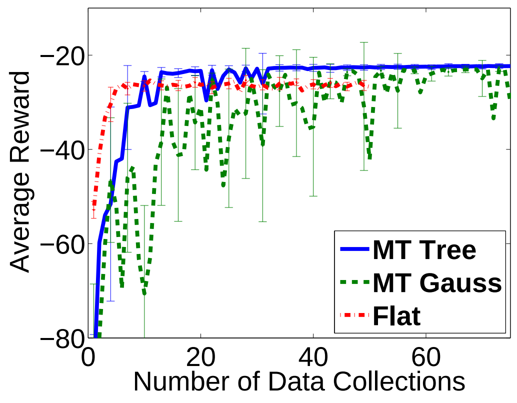
(a) (b)

Figure 7.1: (a) Torque trajectories and motion templates learned for different action punishment factors c_2. (b) Torque trajectories learned with flat RL

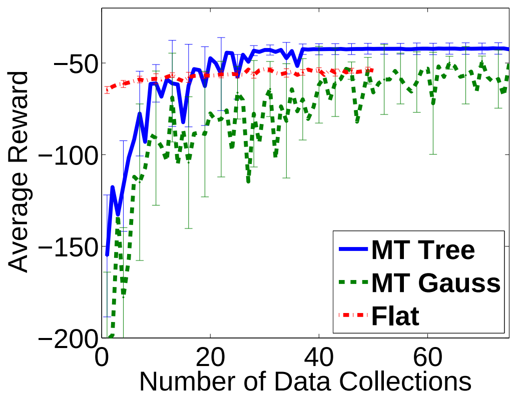
(d_i), m_5 is always the final template and runs for $20s$. Subsequently the episode is ended.

The agent always started from the bottom position with motion template m_0. Afterwards it could either choose to use the peak templates in the predefined order (m_3, m_4, m_1, m_2, m_3...) or use the balancing template m_5. Thus, the agent had to learn the correct parametrization of the motion templates and the number of swing-up motions.

For all experiments a discount factor of $\beta = 0.2$ was used, λ was set to 0.025 and η_{exp} to 20. For the Gaussian kernel we used a bandwidth matrix of $\mathbf{D} = \text{diag}(30, 3)$ for m_1, m_3 and m_5 and $\mathbf{D} = \text{diag}(30, 3, 1)$ for the extended state space of templates m_2 and m_4. For the tree-based kernels we used the parameters $n_{min} = 7$, $M = 80$ and $K = 20$. For the comparison with the flat LAWER algorithm τ was set to 4



(a)



(b)

Figure 7.2: Learning curves for the Gaussian kernel (MT Gauss) and the tree-based kernel (MT Tree) for (a) $c_2 = 0.025$ and (b) $c_2 = 0.075$

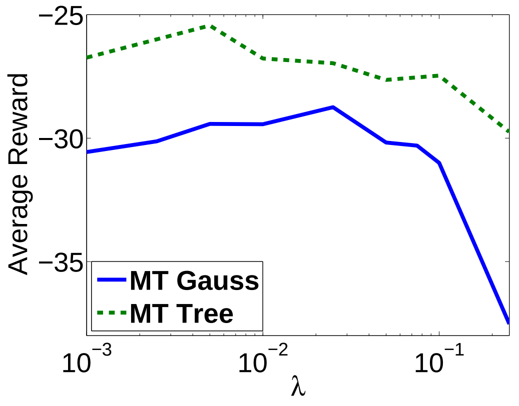
and a time step of 50ms was used. We used $L = 50$ episodes per data collection.

We carried out 3 experiments with different torque punishment factors ($c_2 = 0.005$, $c_2 = 0.025$ and $c_2 = 0.075$). We compared the learning process of flat RL, motion template learning with Gaussian state similarities (MT Gauss) and with adaptive tree-based state similarities (MT Tree) (see Figure 7.2). In the initial learning phase, the flat RL approach is superior to motion template learning, probably due to the larger number of produced training examples. However, RL with motion templates is able to produce policies of significantly higher quality and quickly outperforms the flat RL approach. This can also be seen in Figure 7.1(a) and (b), where the resulting torque trajectories are compared. Flat RL has difficulties particularly with the hardest setting ($c_2 = 0.075$) where we received a maximum average reward of -48.6 for flat RL and -38.5 for the motion template approach. From Figure 7.2 we can also see that the tree-based kernel is much more sample efficient than the Gaussian kernel. An evaluation of the influence of the λ parameter can be seen in Figure 7.3(a) and of the parameter n_{min} of the tree-based kernel in Figure 7.3(b). The approach works robustly for a wide range of parameters.

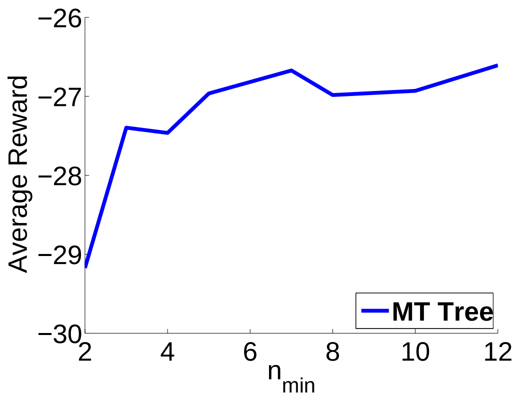
7.5.1.2 2-link Pendulum

We also conducted experiments with a 2-link pendulum. The lengths of the links were set to 1m, each link had a mass of 1kg (located at the center of the link). We use the same templates as for the 1-dimensional task, the peak templates have now 2 additional parameters, the height of the peak a_i and the time offset o_i for the second control dimension u_2. Including the duration parameter, this results in 5 parameters for m_0, m_1 and m_3 and 3 parameters for m_2 and m_4. The parameters of the balancer template m_5 consists of two 2×2 matrices for the controller gains.

Experiments were done for the tree-based kernels with $n_{min} = 8$, $\lambda = 0.025$ and $\eta_{exp} = 25$. At each data collection, 50 new episodes were collected. For comparison to the flat RL approach we used a bandwidth matrix of $D =$

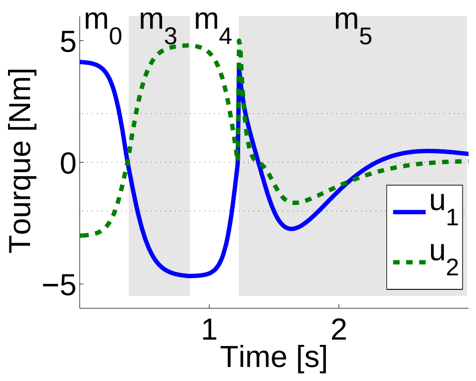


(a)

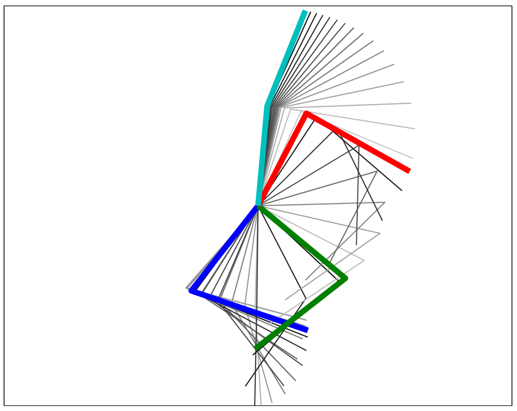


(b)

Figure 7.3: (a) Evaluation of the influence of λ for the Gaussian (MT Gauss) and the tree-based kernel (MT Tree, $n_{\min} = 5$) (b) Evaluation of the $n_{\min}$ parameter for $\lambda = 0.025$. c_2 was set to 0.025 for both evaluations.



(a)



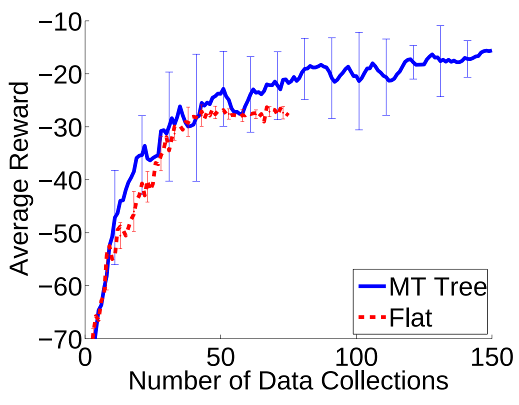
(b)

Figure 7.4: (a) Torque trajectories and decomposition in the motion templates for the 2-link pendulum swing-up task. (b) Illustration of the motion. The bold postures represent the switching time points of the motion templates.

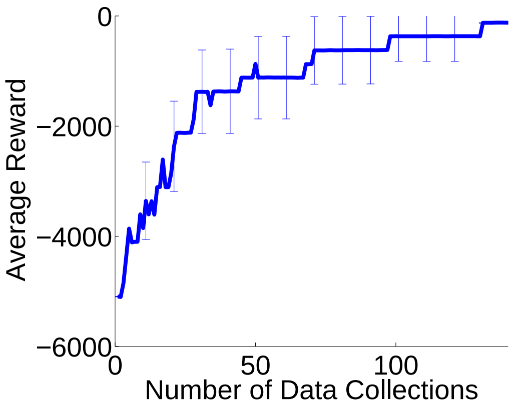
diag(6.36, 2.38, 3.18, 1.06) and $\tau = 4$. The evaluation of the learning process can be seen in Figure 7.5(a) and the learned motion and torque trajectories are shown in Figure 7.4. Also for this challenging task, the motion template approach was able to learn high-quality policies. While the flat RL approach stagnates at an average reward of −28.7, the motion template approach reaches an average reward of −15.6.

7.5.2 2-link Balancing

In this task a 2-link pendulum needs to be balanced at the top position after being pushed. The model parameters were chosen to loosely match the characteristics of a human, i.e. $l_i = 1$m and $m_i = 35$kg. The hip-joint was limited to $[-0.1; 1.5]$rad



(a)



(b)

Figure 7.5: Learning curves for motion template learning with tree-based kernels for the (a) 2-link swing-up task and the (b) 2-link balancing task.

and the ankle-joint to $[-0.8; 0.4]$rad. Whenever the robot left this area of the state space, we assumed that the robot had fallen, i.e. a negative reward of -10000 was given. The hip-torque was limited to ± 500Nm and the ankle torque to ± 70Nm.

In the beginning of an episode, the robot stands upright and gets pushed with a certain force F. This results in an immediate jump of the joint velocities. The agent has to learn to keep balance for different perturbations. In (Atkeson and Stephens, 2007) this problem was solved exactly using Dynamic Programming techniques. The authors found out that two different balancing strategies emerge. For small perturbations, the ankle strategy, which uses almost only the ankle joint, is optimal. For larger perturbations ($F > 17.5$Ns), the ankle-hip strategy, which results in a fast bending movement, is optimal. In this experiment we want to reproduce both strategies by motion template learning.

We use two motion templates to model the balancing behavior, both resemble linear controllers. The first motion template (m_0) keeps the robot at the upright position and is similar to m_5 from the previous experiment. The second template m_1 additionally defines a set-point of the linear controller for each joint and a duration parameter d_1. In addition to the 8 controller gains, this results in 11 parameters. The agent can now choose to use m_0 directly in the beginning or to use m_1 and subsequently m_0. We used 4 different perturbations, i.e., $F = 10, 15, 20$ and 25Ns. For each perturbation, we collected $L = 20$ episodes.

We again used the tree-based approach with the same parameter setting as in the previous experiment. The learning curve can be seen in Figure 7.5(b). The resulting torque trajectories are shown in Figure 7.6(a) and (b). We can clearly identify the ankle strategy for the two smaller perturbations and the ankle-hip strategy for larger perturbations using both motion templates.

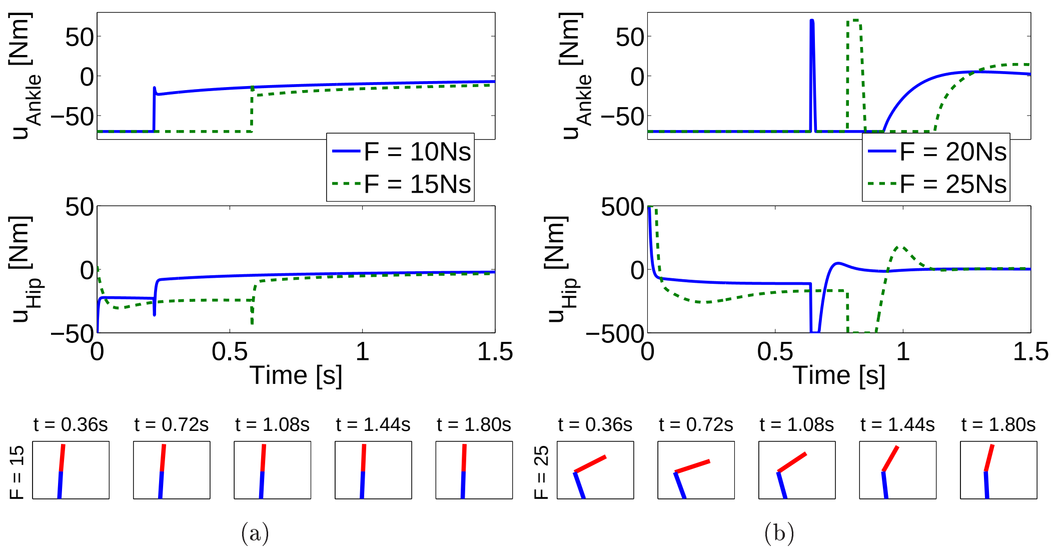


Figure 7.6: Learned solutions for the 2-link balancing problem for (a) $F = 10$Ns and $F = 15$Ns (ankle strategy) (b) $F = 20$Ns and $F = 25$Ns (ankle-hip strategy). The sketches bellow illustrate the temporal course of the balancing movement for the ankle strategy (a) and the ankle-hip strategy (b)

7.6 Conclusion and Future Work

In this paper we proposed a new framework for temporal abstraction for RL in continuous environments, i.e. RL with motion templates. Learning the overall control task is decomposed into learning a sequence of simpler controllers. Because of the used abstractions the agent has to make fewer decisions, which simplifies the learning task. We strongly belief that this kind of abstractions may help scaling RL algorithms to more complex domains.

The motion templates approach also raises several interesting research questions to which we will dedicate our future work. For example, how can we efficiently add feedback to the motion templates? Which functional forms of the templates can facilitate learning? When do we terminate a motion template, in particular in the case of unforeseen events? Future work will also concentrate on applying the approach to more complex environments such as planar walking robots.

7.7 Acknowledgments

This chapter is based on the paper '*Learning Complex Motions by Sequencing Simpler Motion Templates*' (Neumann and Peters, 2009) which has been written by Gerhard Neumann (GN), Wolfgang Maass (WM) and Jan Peters (JP). The algorithm design, implementation and the experiments have been conducted by GN while the initial basic idea was provided by WM. JP greatly helped to improve the paper writting and also provided useful guidance.

CHAPTER 8

Planning Movement Primitives

Contents

A common approach for motor skill learning in robotics is to use parametrized movement plans, also called movement primitives. Currently used approaches endow the primitives with dynamical systems. Here, the parameters of the primitive indirectly define the shape of the desired trajectory. This trajectory is then followed with feedback control laws. Instead of endowing the primitives with dynamical systems, we propose to endow movement primitives with an intrinsic probabilistic planning system, exploiting the power of stochastic optimal control methods already at the level of the primitive. The parametrization of the primitive now specifies a cost function for the intrinsic planning system. We parameterize this intrinsic cost function using use task-relevant features, such as the importance of passing through certain via-points as parameters of the movement. These task-relevant features are learned using standard reinforcement learning, which implies that a (typically) sparse reward signal is transformed into a intrinsic cost function for planning. Simultaneously we learn the dynamics model of the robot. Together, the intrinsic cost function and the dynamics model fully specify a graphical model for movement planning. In difference to current methods, the probabilistic planner can naturally deal with noisy systems, exploiting the stochastic dynamics by suppressing the inherent noise in the system only if necessary. This is also known as the minimum intervention principle, a basic property of human movement control. We evaluate our approach on a complex 4-link balancing task. Our experiments show that our movement representation facilitates learning and allows learning of motor skills up to one order of magnitude faster than traditional approaches. The representation can be easily generalized to new task settings without re-learning and also generates policies with higher quality.

8.1 Introduction

The use of movement primitives has often been shown to facilitate learning of complex movement skills (d'Avella et al., 2003; Schaal et al., 2003; Neumann et al., 2009). They allow an efficient abstraction of the high-dimensional continuous action spaces which often occur in robotics. Movement primitives are parametrized representations of elementary movements. For current approaches the parameters of the primitive determine the shape of the desired trajectory either directly or indirectly. This trajectory is then followed by feedback control laws. An example for an indirect trajectory parametrization are the widely used Dynamical Movement Primitives (DMPs) (Schaal et al., 2003). This approach uses parametrized dynamical systems to determine a movement trajectory. The idea of DMPs to endowing movement primitives with an intrinsic dynamic system has many benefits: They provide a linear policy parametrization which can be used for imitation learning and policy search (Kober and Peters, 2010). The complexity of the trajectory can be scaled by the number of parameters (Schaal et al., 2003) and one can adapt meta-parameters of the movement such as the movement speed or the goal state of the movement (Kober et al., 2010; Pastor et al., 2009).

The general idea of the present work is to endow movement primitives with an intrinsic planning system instead of an intrinsic dynamic system. While the dynamic system of a DMP is to some degree reactive to the environment—namely by adapting the temporal scaling factor and thereby de- or accelerating the movement execution as needed (Schaal et al., 2003)—the trajectory shape itself is fixed and non-reactive to the environment. In contrast, a movement primitive that is endowed with an intrinsic planning component can react to the environment by optimizing the trajectory for the specific current situation. Training such a movement primitive now means to train a planner to generate an appropriate policy in a given situation instead of training a dynamical system to generate a fixed (temporally flexible) reference trajectory. This implies a different level of generalization. For instance, if some endeffector target changes between training and testing phase, a planner that has learned to generate trajectories towards targets will generalize to the new target without retraining. A system that directly encodes a trajectory would either have to be retrained or use heuristics to be adapted (Pastor et al., 2009).

Stochastic optimal control, besides its high relevance in engineering problems, has proven itself as an excellent computational theory of human movement control (Todorov and Jordan, 2002). For example, the *minimum intervention principle* implies that we should only intervene the system if it is necessary to fulfill the given task. If the task constraints are not violated it is inefficient to suppress the inherent noise in the stochastic system. As an example consider the problem of performing a tennis serve. The most task-relevant feature of the movement is the state of the arm (including velocities, accelerations and stiffness) at the point in time when the racket hits the ball. For this time point the movement has to be very precise, but elsewhere less accurate control is sufficient. Human movements account for such principles, suggesting that stochastic optimal control principles are involved on the lowest level of movement execution. On the other hand: A tennis serve is certainly also a highly trained movement primitive. This exemplifies our general view of a movement primitive system which can be trained in a reinforcement learning setting, but which also involves a low-level movement planner that accounts for fundamental optimality principles. In contrast, a movement primitive system that implies a fixed reference trajectory would force the movement to follow this reference more or less accurately and can only choose how much noise is suppressed by the feedback control law throughout the execution of the whole trajectory. It can not generate movements that fulfill optimality principles for variable target or task constraints.

Therefore we propose *Planning Movement Primitives* (PMPs) which exploit the power of stochastic optimal control (SOC) methods (Todorov and Li, 2005; Kappen, 2007) *within* the primitive. As with DMPs, a PMP is trained in a standard reinforcement learning (RL) setting. Instead of parametrizing the shape of the trajectory directly, a PMP has parameters that determine the the intrinsic cost function of the intrinsic SOC planner. While the reward function (typically) gives a single scalar reward for a whole movement, the learned intrinsic cost function in the standard SOC form defines task and control costs for every time step of the movement. In other terms, training a PMP is the problem learning from a sparse reward signal an intrinsic cost function such that the SOC planner will, with high probability, generate rewarded movements. Parametrizing the intrinsic cost function allows us to use task-relevant features as parameters of the movement, e.g. the importance of passing through a certain via-point.

Training a PMP also requires to learn an approximate model of the system dynamics within the RL setting since the intrinsic SOC planner requires some approximate model to estimate optimal control. Therefore, PMP learning combines model-based and model-free RL: it learns a model of the system dynamics while at the same time training PMP parameters based on the reward signal. (It does not learn an approximate model of the reward function itself.) We can exploit supervised learning methods such as (Vijayakumar et al., 2005; Nguyen-Tuong et al., 2008a,b) for learning the system dynamics and at the same time use policy search methods to adapt the PMP parameters that determine the intrinsic cost function. This two-fold learning strategy has the promising property of fully exploiting the data by also estimating the system dynamics instead of only adapting policy parameters.

As planning algorithm we employ a probabilistic planner called Approximate Inference Control (AICO), (Toussaint, 2009). AICO generates the movement by performing inference in a graphical model. The graphical model is defined by the system dynamics and the intrinsic cost function. Since we learn both from experience (the latter via policy search) all conditional probability distributions of this graphical model are determined empirically. The output of the planner is a linear regulator for each time slice.

Our experiments show that, by the use of task relevant features, we can significantly facilitate learning and generalization of complex movement skills. Moreover, due to the intrinsic SOC planner, our primitive representation implements all principles of optimal control, which allows to learn solutions of high quality which are not representable with traditional trajectory-based methods.

In the following section we review in more detail related previous work and the background on which our methods build. Section 8.2 then introduces the proposed Planning Movement Primitives. In Section 8.3 we evaluate the system on a one-dimensional via-point task and a complex dynamic humanoid balancing task and compare to DMPs. We conclude this work with a discussion in Section 8.4.

8.1.1 Related Work and Background

This section reviews the related work based on parametrized movement policies, policy search methods and stochastic optimal control.

8.1.2 Parametrized Movement Policies

Movement primitives represent a parametric description of elementary movements (d'Avella et al., 2003; Schaal et al., 2003; Neumann et al., 2009). We will denote the parameter vector of a movement primitive by $\boldsymbol{\theta}$ and the possibly stochastic policy of the primitive as $\pi(\mathbf{u}|\mathbf{x}, t; \boldsymbol{\theta})$, where $\mathbf{u}$ is the applied action and $\mathbf{x}$ denotes the state. The key idea of the term 'primitive' is that several of these elementary movements can be combined not only sequentially but also simultaneously in time. However, in this paper, we want to concentrate on the parametrization of a single primitive, i.e. only learn a single elementary movement. Using several primitives simultaneously is part of future work for our approach as well as for existing approaches such as (Schaal et al., 2003; Neumann et al., 2009).

Many types of movement primitives can be found in the literature. The currently most widely used movement representation used for robot control are the Dynamic Movement Primitives (DMPs) (Schaal et al., 2003). DMPs evaluate parametrized dynamical systems to generate trajectories. The dynamical system is constructed such that the system is stable. In order to do so, a linear dynamical system is used which is modulated by a learnable non-linear function f. A great advantage of the DMP approach is that the function f depends linearly on the parameters $\boldsymbol{\theta}$ of the primitive, i.e $f(s) = \boldsymbol{\Phi}(s)^T\boldsymbol{\theta}$, where s is the time or phase variable. As a result, imitation learning for DMPs is straightforward as this can simply be done by performing a linear regression (Schaal et al., 2003). Furthermore, it also allows the use of many well-established reinforcement learning methods such as policy gradient methods (Peters and Schaal, 2008b) or Policy Improvements by Path Integrals (Theodorou et al., 2010a). The complexity of the trajectory can be scaled by the number of features used for modelling f. However, as the features $\boldsymbol{\Phi}(s)$ are fixed, the ability of the approach to extract task-relevant features is limited. We can also adapt meta-parameters of the movement such as the movement speed or the goal state of the movement (Kober et al., 2010; Pastor et al., 2009). Yet, the change of the desired trajectory due to the change of the meta-parameters is based on heuristics and does not consider task relevant constraints. As the DMPs are the most common movement representation we will use it as a baseline in our experiments. For a more detailed discussion of the DMP approach please consult the appendix.

Another type of movement representation was introduced in (Neumann et al., 2009) by the movement template framework. Movement templates are temporally extended, parametrized actions, such as sigmoidal torque, velocity or joint position profiles, which can be sequenced in time. This approach uses a more complex parametrization as the DMPs. For example, it also incorporates the duration of different phases, like an acceleration or deceleration phase. The division of a movement into single phases allows the use of reinforcement learning methods to learn how to sequence these primitives. However, as the approach still directly specifies the shape of the trajectory, defining complex movements for high dimensional systems is still complicated, which has restricted the use of movement templates to rather simple applications.

An interesting movement representation coming from experimental data analysis are the muscle synergies (d'Avella et al., 2003; Bizzi et al., 2008). They have been used to provide a compact representation of electromyographic muscle activation patterns. The key idea of this approach is that muscle activation patterns are composed of a linear sum of simpler, elemental patterns, called muscle synergies. Each muscle synergy can be shifted in time and scaled with a linear factor to construct the whole activation pattern. While the synergy approach has promising properties such as the linear superposition and the ability to share synergies between tasks, except for some smaller applications (Chhabra and Jacobs, 2006), these primitives have only been used for data analysis, and not for robot control.

All the so far presented primitives are inherently local approaches. The specified trajectory and hence the resulting policy are only valid for a local (typically small) neighborhood of our initial state. If we are in a new situation, it is likely that we need to re-estimate the parameters of the primitive. The generation of the reference trajectory for these approaches is often an offline process and does not incorporate knowledge of the system dynamics, proprioceptive or other sensory feedback. Because the reference trajectory itself is usually created without any knowledge of the system model, the desired trajectory might not be applicable, and thus, the real trajectory of the robot might differ considerably from the specified trajectory.

There are only few movement representations which can also be used globally, i.e. for many different initial states of the systems. One such methods is the Stable Estimator of Dynamical Systems (SEDS) (Khansari-Zadeh and Billard, 2011) approach. However, this method has so far only been applied to imitation learning, using the approach for learning or improving new movement skills is not straight forward. We will therefore restrict our discussion to local movement representations.

Our Planning Movement Primitive approach is, similar to the DMPs, a local approach. In a different situation, different abstract goals and features might be necessary to achieve a given task. However, as we extract task relevant features and use them as parameters, the same parameters can be used in different situations as long as the task relevant features do not change. As we will show, the valid region where the local primitives can still be applied is much larger for the given control tasks in comparison to trajectory based methods.

8.1.3 Policy Search for Movement Primitives

Let $\mathbf{x}$ denote the state and $\mathbf{u}$ the control vector. A trajectory τ is defined as sequence of state control pairs, $\tau = \langle \mathbf{x}_{0:T}, \mathbf{u}_{0:T-1} \rangle$, where T is the length of the trajectory. Each trajectory has associated costs $C(\tau)$ (denoted as extrinsic cost), which can be an arbitrary function of the trajectory. It can, but need not be composed of the sum of intermediate costs during the trajectory. For example, it could be based on the minimum distance to a given point throughout the trajectory. We want to find a movement primitive's parameter vector $\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ which minimizes the expected costs $J(\boldsymbol{\theta}) = \mathbb{E}\left[C(\tau)|\boldsymbol{\theta}\right]$. We assume that we can evaluate the expected costs $J(\boldsymbol{\theta})$ for a given parameter vector $\boldsymbol{\theta}$ by performing roll-outs on the real system.

In order to find $\boldsymbol{\theta}^*$ we can apply policy search methods. Here a huge variety of possible methods exists. Policy search methods can be coarsely divided into step-based exploration and episode-based exploration approaches. Step-based exploration approaches such as (Theodorou et al., 2010a; Peters and Schaal, 2008b; Kober and Peters, 2010) apply an exploration noise to the action of the agent at each time step of the episode. Subsequently, the policy is updated such that the (noisy) trajectories with higher reward are more likely to be repeated. In order to do this update, step-based exploration techniques strictly rely on a policy which is linear in its parameters. This is true for the DMPs. Currently, the most common policy search methods are step-based approaches, including the REINFORCE (Williams, 1992), the episodic Natural Actor Critic (Peters and Schaal, 2008b), the PoWER (Kober and Peters, 2010) or the PI^2 (Theodorou et al., 2010a) algorithm. This also explains partially the popularity of the DMP approach for motor skill learning because DMPs are, from those introduced above, the only representation which can be used for these step-based exploration methods (apart from very simple ones like linear controllers).

However, recent research has also intensified on episode-based exploration techniques (Sehnke et al., 2010; Wierstra et al., 2008; Hansen et al., 2003). These methods directly perturb the policy parameters $\boldsymbol{\theta}$ and then estimate the performance of the perturbed $\boldsymbol{\theta}$ parameters by performing roll-outs on the real system. During the episode no additional exploration is applied (i.e. a deterministic policy is used). The policy parameters are then updated in the estimated direction of increasing performance. Episode-based exploring methods do not depend on a specific form of parametrization of the policy. In addition, episode-based exploration techniques easily allow the use of second order stochastic search methods that estimate correlations between policy parameters (Heidrich-Meisner and Igel, 2009b; Wierstra et al., 2008). This ability to apply correlated exploration in parameter-space is often beneficial in comparison to the uncorrelated exploration techniques applied by all step-based exploration methods.

Since the resulting control policies of our PMPs depend non-linearly on the parameters $\boldsymbol{\theta}$, step-based exploration techniques can not be used in our setup. Hence, we will use the second order stochastic search method CMA (Covariance Matrix Adaptation, (Hansen et al., 2003)) which makes no assumptions on the parametrization of the primitive. CMA uses a multivariate Gaussian distribution to represent the belief over the optimal parameters and has been shown to be highly competitive for policy search in high dimensional spaces. We will compare our PMP approach to both, DMPs learned with CMA policy search and DMPs learned with the state of the art step-based method PI^2 (Theodorou et al., 2010a). Interestingly, the second order stochastic search method outperformed PI^2 for DMPs, illustrating the benefits of second order optimization.

8.1.4 Stochastic Optimal Control and Probabilistic Inference for Planning

Stochastic optimal control (SOC) methods such as (Todorov and Li, 2005; Kappen, 2007; Toussaint, 2009) have been shown to be powerful methods for movement planning in high-dimensional robotic systems. The incremental Linear Quadratic Gaussian (iLQG) (Todorov and Li, 2005) algorithm is one of the most commonly used SOC algorithms. It uses Taylor expansions of the system dynamics and cost function to convert the non-linear control problem in a Linear dynamics, Quadratic costs and Gaussian noise system (LQG). The algorithm is iterative - the Taylor expansions are recalculated at the newly estimated optimal trajectory for the LQG system.

In (Toussaint, 2009), the SOC problem has been reformulated as inference problem in a graphical model, resulting in the Approximate Inference Control (AICO) algorithm. The graphical model is given by a simple dynamic Bayesian network with states $\mathbf{x}_t$, actions $\mathbf{u}_t$ and task variables $\mathbf{g}^{[i]}$ (representing the costs) as nodes, see Figure 8.1. The dynamic Bayesian network is fully specified by conditional distributions encoded by the cost function and by the state transition model. If beliefs in the graphical model are approximated as Gaussian the resulting algorithm is very similar to iLQG. Gaussian message passing iteratively re-approximates local costs and transitions as LQG around the current mode of the belief within a time slice. A difference to iLQG is that is uses forward messages instead of a forward roll-out to determine the point of local LQG approximation and can iterate belief re-approximation with in a time slice until convergence, which may lead to faster overall convergence. For a more detailed discussion of the AICO algorithm with Gaussian message passing see Section 8.2.5 and the appendix.

Local planners have the advantage that they can be applied to high-dimensional dynamical systems, but the disadvantage of requiring a suitable initialization. Global planning (Kuffner and LaValle, 2000) on the other hand does not require an initial solution, however, they have much higher computational demands. Our motivation for using only a local planner as component of a Planning Movement Primitive is related to the learning of an intrinsic cost function:

Existing planning approaches for robotics typically use hand-crafted cost functions and the dynamic model is either analytically given or learned from data (Mitrovic et al., 2010). PMPs use reinforcement learning to train an intrinsic cost function for planning instead of trying to learn a model of the extrinsic reward directly. The reason is that a *local* planner often fails to directly solve realistically complex tasks by optimizing directly the extrinsic cost functions. From this perspective, PMPs learn to translate complex tasks to a simpler intrinsic cost function that can efficiently be optimized by a local planner. This learning is done by trial-and-error in the reinforcement learning setting: the PMP essentially learns from experience which intrinsic cost function the local planner can cope with and use to generate good trajectories. Thereby, the reinforcement learning of the intrinsic cost function can compensate the limitedness of the local planner.

8.2 Planning Movement Primitives

In this section we introduce the proposed Planning Movement Primitives (PMPs), in particular the parametrization of the intrinsic cost function. The overall system will combine three components: (1) a regression method for learning the system dynamics, (2) a policy search method for finding the PMP parameters, and (3) a SOC planner for generating movements with the learned model and PMP parameters.

8.2.1 Problem Definition

We assume an unknown dynamic system of the general form

$$\mathbf{x}_{t+1} = f_{\text{Dyn}}(\mathbf{u}_t, \mathbf{x}_t) + \varepsilon_t, \tag{8.1}$$

with state variable $\mathbf{x}_t$, controls $\mathbf{u}_t$ and Gaussian noise $\varepsilon_t \sim \mathcal{N}(0, \S)$. The agent is to realize a control policy $\pi : \mathbf{x}_t \mapsto \mathbf{u}_t$, which in our case will be a linear regulator for each time slice. The problem is to find a policy that minimizes the expected costs of a finite-horizon episodic task. That is, we assume there exists a cost function $C(\tau)$, where $\tau = (\mathbf{x}_{0:T}, \mathbf{u}_{0:T})$ is roll-out of the agent controlling the system. The problem is to find $\text{argmin}_\pi \langle C(\tau) \rangle_\pi$.

The system dynamics f_{Dyn} as well as the cost function $C(\tau)$ are analytically unknown. Concerning the system dynamics we can compute an approximate model of the systems dynamics from a set of roll-outs—as standard in model-based reinforcement learning (RL). However, concerning costs, we only receive the single scalar cost $C(\tau)$ after a roll out indicating the quality or success of a movement. Note that $C(\tau)$ is a function of the whole trajectory, not only the final state. Learning C from data would be an enormous task, more complex that learning a reward function $\mathbf{x}_t \mapsto \mathbf{r}_t$ as in standard model-based RL. Further, if we try to model $C(\tau)$ directly and apply SOC methods to optimize it, $C(\tau)$ would have to be modelled in the form $C(\tau) = \sum_t h_t(\mathbf{x}_t, \mathbf{x}_{t+1})$—assigning separate costs to each time step of the roll-out. This implies an enormous credit assignment problem.

Generally, approaches to learn $C(\tau)$ directly in a form useful for applying SOC methods seems an overly complex task and violates the maxim "never try to solve a problem more complex than the original". Therefore, our approach will not try to learn $C(\tau)$ from data but to employ reinforcement learning to learn *some* intrinsic cost function that can efficiently be optimized by SOC methods and generates control policies that, by empiricism, minimizes $C(\tau)$.

8.2.2 Parametrization of PMP's intrinsic cost function

In PMPs the parameters $\boldsymbol{\theta}$ specify task-relevant abstract goals or features of the movement, which specify an intrinsic cost function

$$L(\tau; \boldsymbol{\theta}) \quad := \quad \sum_{t=0}^{T} l(\mathbf{x}_t, \mathbf{u}_t, t; \boldsymbol{\theta}) + c_p(\mathbf{x}_t, \mathbf{u}_t), \tag{8.2}$$

where l denotes the intermediate intrinsic cost function for every time-step and $c_p(\mathbf{x}_t, \mathbf{u}_t)$ is used to represent basic known task constraints, such as torque or joint limits. We will assume that basic task constraints like joint and torque limits are part of our prior knowledge, thus c_p is given and not included in our parametrization. For the description of PMPs we will neglect the constraints c_p for simplicity. We will use a via-point representation for the intermediate intrinsic cost function $l(\mathbf{x}_t, \mathbf{u}_t, t; \boldsymbol{\theta})$.

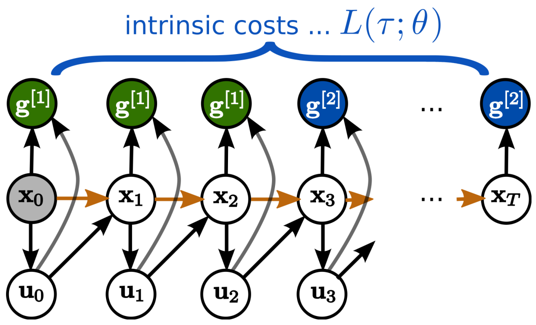


Figure 8.1: Planning Movement Primitives are can be illustrated using graphical models. States are denoted by $\mathbf{x}_t$, controls by $\mathbf{u}_t$ and the time horizon is fixed to T time-steps. In this example the graphical model is used to infer the movement by conditioning on two abstract goals $\mathbf{g}^{[1]}$ and $\mathbf{g}^{[2]}$, which are specified in the *learned* intrinsic cost function $L(\tau;\boldsymbol{\theta})$.

Therefore, parameter learning corresponds to extracting goals which are required to achieve a given task, such as passing through a via-point at a given time. As pointed out in the previous section, $L(\tau;\boldsymbol{\theta})$ is *not* meant to approximate $C(\tau)$. It should to provide a feasible cost function that empirically generates policies that minimize $C(\tau)$.

There are many ways to parametrize the intermediate intrinsic cost function l. We choose a simple via-point approach. The movement is decomposed in N shorter phases with duration $d^{[i]}$, $i = 1,..,N$. In each phase the cost function is assumed to be quadratic in the state and control vectors. In the ith phase ($\sum_{j=1}^{i-1} d^{[i]} < t \leq \sum_{j=1}^{i} d^{[i]}$) we assume the intrinsic cost has the form:

$$l(\mathbf{x}_t, \mathbf{u}_t, t; \boldsymbol{\theta}) \quad = \quad (\mathbf{x}_t - \mathbf{g}^{[i]})^T \mathbf{R}^{[i]} (\mathbf{x}_t - \mathbf{g}^{[i]}) + \mathbf{u}_t^T \mathbf{H}^{[i]} \mathbf{u}_t. \tag{8.3}$$

It is parametrized by the reference point $\mathbf{g}^{[i]}$ in state space; by the precision vector $\mathbf{r}^{[i]}$ which determines $\mathbf{R}^{[i]} = \text{diag}(\exp \mathbf{r}^{[i]})$ and therefore how steep the potential is along each state dimension; and by the parameters $\mathbf{h}^{[i]}$ which determine $\mathbf{H}^{[i]} = \text{diag}(\exp \mathbf{h}^{[i]})$ and therefore the control costs along each control dimension. We represent the importance factors $\mathbf{r}^{[i]}$ and $\mathbf{h}^{[i]}$ both in log space as we are only interested in to relationship of this factors. At the end of each phase (at the via-point), we multiply the quadratic state costs by the factor 1/dt where dt is the time step used for planning. This ensures that at the end of the phase the via-point is reached, while during the phase the movement is less constraint. With this representation, the parameters $\boldsymbol{\theta}$ of our PMPs are given by

$$\boldsymbol{\theta} = [d^{[1]}, \mathbf{g}^{[1]}, \mathbf{r}^{[1]}, \mathbf{h}^{[1]} \; \dots \; d^{[N]}, \mathbf{g}^{[N]}, \mathbf{r}^{[N]}, \mathbf{h}^{[N]}] \tag{8.4}$$

Cost functions of this type are commonly used—and hand-crafted—in control problems. They allow to specify a reference, but also to determine whether only certain dimensions of the state need to be controlled to the reference and how this trades of with control cost. Instead of hand-designing such cost functions, our method will use CMA policy search to learn these parameters of the intrinsic cost function. As for the DMPs we will assume that the desired final state at time point T is known, and thus $\mathbf{g}^{[N]}$ and $d^{[N]}$ are fixed and not included in the parameters. Still, the algorithm can choose the importance factors $\mathbf{r}^{[N]}$ and $\mathbf{h}^{[N]}$ of the final phase. In addition, we fix the velocities in the via-points $\mathbf{g}^{[i]}$ to zero, however, the algorithm can still reach the via-points with non-zero velocities by choosing very low importance factors for the velocities (included in $\mathbf{r}^{[i]}$).

8.2.3 Dynamic Model Learning

In order to use planning we need to learn a model of the system dynamics f_{Dyn} in Equation 8.1. The planning algorithm can not interact with the real environment, it solely has to rely on the learned model. Only after the planning algorithm is finished, the resulting policy is executed on the real system and new data points $\langle[\mathbf{x}_t, \mathbf{u}_t], \dot{\mathbf{x}}_t\rangle$ are collected for learning the model.

Many types of function approximators can be applied in this context (Vijayakumar et al., 2005; Nguyen-Tuong et al., 2008a,b). We use the lazy learning technique Locally Weighted Regression (LWR) (Atkeson et al., 1997) as it is a very simple and effective approach. LWR is a memory-based, non-parametric approach, which fits a local linear model to the locally-weighted set of data points. For our experiments, the size of the data set was limited to 10^5 points implemented as a first-in-first-out queue buffer because the computational demands of LWR drastically increases with the size of the data set.

8.2.4 Policy search

Model learning takes place simultaneously to learning the parameters $\boldsymbol{\theta}$ of the primitive. In general this could lead to some instability. However, while the distribution $P(\mathbf{x}_t)$ depends on the policy and the data for model learning is certainly non-stationary, the conditional distribution $P(\mathbf{x}_{t+1}|\mathbf{u}_t, \mathbf{x}_t)$ is stationary. A local learning scheme as LWR behaves rather robust under such type of non-stationarity of the input distribution only. On the other hand, from the perspective of $\boldsymbol{\theta}$ optimization, the resulting policies may change and lead to different payoffs $C(\tau)$ even for the same parameters $\boldsymbol{\theta}$ due to the adaption of the learned system dynamics.

We employ the second order stochastic search method CMA (Heidrich-Meisner and Igel, 2009b) to optimize the parameters $\boldsymbol{\theta}$ w.r.t. $C(\tau)$. Roughly, CMA is an iterative procedure that, from the current Gaussian distribution, generates a number of samples, evaluates the samples, computes second order statistics of those samples that reduced $C(\tau)$ and uses these to update the Gaussian search distribution. In each iteration, all parameter samples $\boldsymbol{\theta}$ use the same learned dynamic model to evaluate $C(\tau)$. Further, CMA includes an implicit forgetting in its update of the Gaussian distribution and therefore behaves robust under the non-stationary introduced by adaptation of the system dynamics model.

Note that even if the learned model is only a roughly approximation of the true dynamics, the policy search for parameters of the intrinsic cost function can compensate for an imprecise dynamics model: The RL approach will find parameters $\boldsymbol{\theta}$ of the intrinsic cost function such that—even with a mediocre model—the resulting controller will lead to low extrinsic costs in the real system.

8.2.5 Probabilistic Planning Algorithm

We use the probabilistic planning method Approximate Inference Control (AICO) (Toussaint, 2009) as intrinsic planning algorithm. It offers the interpretation that a movement primitive can be represented as graphical model and the movement itself is generated by inference in this graphical model.

The graphical model is fully determined by the learned system dynamics and the learned intrinsic cost function, see Figure 8.1. In order to transform the minimization of $L(\tau;\theta)$ into an inference problem, for each time-step an individual binary random variable z_t is introduced. This random variable indicates a reward event. Its probability is given by

$$P(z_t = 1|\mathbf{x}_t, \mathbf{u}_t, t) \propto \exp(-c_t(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta})),$$

where $c_t(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}) = l(\mathbf{x}_t, \mathbf{u}_t, t; \boldsymbol{\theta}) + c_p(\mathbf{x}_t, \mathbf{u}_t)$ denotes the cost function for time step t. AICO now assumes that a reward event $z_t = 1$ is observed at every time-step. Given that evidence, AICO calculates the posterior distribution $P(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}|z_{1:T} = 1)$ over trajectories.

We will use the simplest version of AICO, where an extended Kalman smoothing approach is used to estimate the posterior. The extended Kalman smoothing approach uses Taylor expansions to linearize the system and subsequently uses Gaussian message passing to perform the inference. Subsequently the system is linearized again at the new mode of the belief over the trajectories. AICO is only a local optimization method and we have to provide an initial solution which is used for the first linearization. We will use the direct path to the via-points $\mathbf{g}^{[i]}$ in Equation 8.3 as initial solution. AICO provides us with an linear feedback controller for each time slice of the form

$$\mathbf{u}_t = \mathbf{O}_t \mathbf{x}_t + \mathbf{o}_t, \tag{8.5}$$

which is used as policy of the movement primitive.

The original formulation of the AICO method (Toussaint, 2009) does not consider torque limits, which are important for our dynamic balancing experiments, and hence, we needed to extend the algorithm. This extension yields not only a modified form of the immediate cost function but also results in different update equations for the messages and finally different equations of the optimal feedback controller. A complete derivation of the extension including the resulting messages and the corresponding feedback controller is given in Appendix 8.5.

The complete learning framework is organized the following. Given the parameters $\boldsymbol{\theta}$, AICO is initialized with an initial solution which is the direct path from via-point to via-point. AICO is then used to optimize the parametrized intrinsic cost function and the result of this optimization process is a linear feedback controller for each time slice, see Equation 8.5. This feedback control law is executed either on a real or simulated robot and the overall task performance (or extrinsic costs) $C(\tau)$ of the resulting trajectory is evaluated. Model learning takes place in parallel and uses all data collected during the roll-outs, see Figure 8.2.

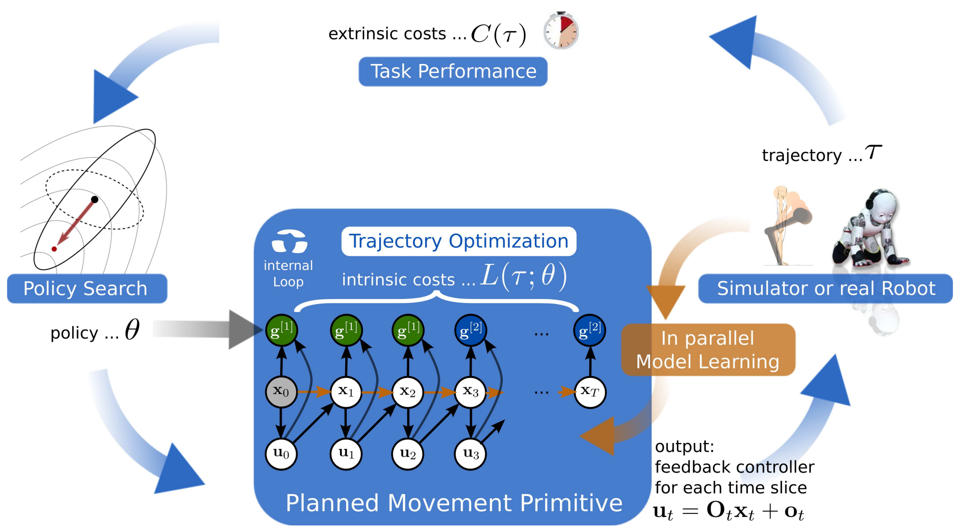

extrinsic costs ... $C(\tau)$
Task Performance
trajectory ... τ
Policy Search
Trajectory Optimization
internal Loop
intrinsic costs ... $L(\tau;\theta)$
Simulator or real Robot
policy ... θ
In parallel Model Learning
output: feedback controller for each time slice
$\mathbf{u}_t = \mathbf{O}_t\mathbf{x}_t + \mathbf{o}_t$
Planned Movement Primitive

Figure 8.2: We decompose motor skill learning into two different learning problems. At the highest level we find parameters of an intrinsic cost function $L(\tau;\boldsymbol{\theta})$ using policy search. Given parameters $\boldsymbol{\theta}_i$ the probabilistic planner at the lower level uses the intrinsic cost function $L(\tau;\boldsymbol{\theta})$ to estimate a non-linear feedback controller for each time step. The feedback controller is subsequently executed on the real robot and the extrinsic cost $C(\tau_i)$ is evaluated. Simultaneously we collect samples of the system dynamics $\langle[\mathbf{x}_t, \mathbf{u}_t], \dot{\mathbf{x}}_t\rangle$ while executing the movement primitive. These samples are used to improve our learned dynamics model which is used for planning.

8.3 Experiments

We start our evaluation of the proposed Planning Movement Primitive (PMP) approach on a one-dimensional via-point task to illustrate basic characteristics. In order to demonstrate our approach on a more challenging dynamic robot task we choose a complex 4-link humanoid balancing task. In our experiments, we focus on investigating robustness to noise, optimality of the solution, learning speed and generalizability to different initial or target states. For a comparison we take the commonly used DMPs as a baseline where we use the newest version of the DMPs (Pastor et al., 2009) as discussed in detail in Appendix 8.5. In difference to most applications of the DMPs, are we learning from scratch without the use of imitation learning. As described above we use 2nd order stochastic search to learn the PMP and DMP parameters. In order to compare to a more commonly used policy search algorithm we additionally test using the PI^2 algorithm for learning the DMPs. For all experiments we empirically evaluate the optimal settings of the algorithms (such as the exploration rate of CMA and PI^2 or the number of centers for the DMPs), which are listed in the Appendix 8.5.

8.3.1 One-dimensional via-point task

In this task the agent has to control a one dimensional point mass. The state at time t is denoted by $\mathbf{x}_t = [\phi_t, \dot{\phi}_t]^T$ and we directly control the acceleration. The time horizon was limited to $T = 0.5$s. Starting at $\mathbf{x}_0 = [0, 0]^T$ the agent has to pass through a given via-point $g_v = -0.2$ at $t_v = 300$ms and the final target g_T was set to 1, see Figure 8.3. We define the extrinsic costs for this task:

$$C(\tau) = 10^4(\dot{\phi}_T^2 + 10(g_T - \phi_T)^2) + 10^5(g_v - \phi_{t_{300\mathrm{ms}}})^2 + 5 \cdot dt \cdot 10^{-1} \sum_{t=0}^{T} u_t^2.$$

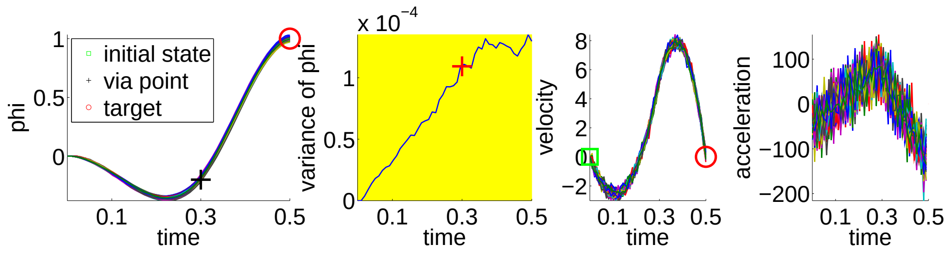
The first two terms punish deviations from the target g_T and the via-point g_v. The target should be reached with zero velocity at $T = 0.5$s. The last term punishes high energy consumption. The control action is noisy, we always add a Gaussian noise term with a standard deviation of $\sigma = 20$ to the control action. The simulation time step was set to 10ms. As this is a very simple task, we use it just to show different characteristics of the DMPs and PMPs.

A quite similar task has been used in (Todorov and Jordan, 2002) to study human movement control. The experiments showed that humans were able to reach the given via-points with high accuracy, however, in between the via-points, the trial-to-trial variability was rather high. This is a well known concept from optimal control, called the *minimum intervention principle*, showing also that human movement control follows basic rules of optimal control. This observation also contradicts that humans use a pure trajectory based movement representation. Still, the *minimum intervention principle* is consistent with the dynamical system view of movement control, however, much more complex dynamical systems than the DMPs are needed (i.e. coupled dynamical systems with non-constant system parameters such as the damping constants) to reproduce this effect.

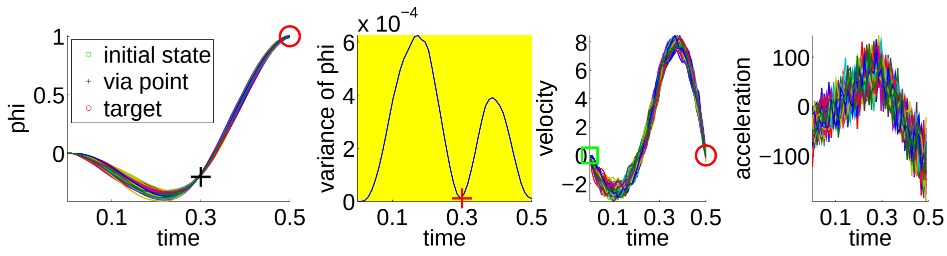
We first estimate the quality of the *best available* policy with the DMP and the PMP approach. We therefore use the PMPs with two via-points and set the parameters $\boldsymbol{\theta}$ per hand. As we are using a linear system model and a simple extrinsic cost function, the PMP parameters can be directly obtained by looking at the extrinsic costs. As the PMPs use the AICO algorithm which always produces optimal policies for LQG systems, the PMP solution is the optimal solution. We subsequently use the mean trajectory returned by AICO and use imitation learning to fit the DMP parameters. We also optimized the feedback controllers used for the DMPs. In Figure 8.3 we plotted 100 roll-outs of the DMP and PMP approach using this optimal policies. The second column illustrates the trial-to-trial variability of the trajectories. The optimal solution has minimum variance at the via-point and the target. As expected this solution is reproduced with the PMP approach, because the parameters of the PMPs are able to reflect the importance of passing through the via-point. The DMPs could not adapt the variance during the movement because the used (optimized) feedback controller uses constant controller gains. As we can see, the variance of the DMP trajectory is simply increasing with time.

Comparing the optimal solutions we find that PMPs, in contrast to DMPs, can naturally deal with the inherent noise in the system. This is also reflected by the average cost values over 1000 trajectories, 1286 ± 556 for the DMPs and 1173 ± 596 for the PMPs. The $\pm$ symbol always denotes the standard deviation.

This advantage would not be very useful if we were not able to learn the optimal PMP parameters from experience. Next we test using CMA policy search to learn the parameters for the DMPs and the PMPs. In addition, in order to compare to



(a) Best found policy with DMPs (average costs over 1000 traj: 1286 ± 556)



(b) Best found policy with PMPs (average costs over 1000 traj: 1173 ± 596)

Figure 8.3: Best available policies for the PMPs and the DMPs for the via-point task. The agent has to pass the via-point at $t_v = 0.3$s and deal with the stochasticity of the system (Gaussian control noise with a variance of 20^2). The plot shows 100 trajectories reproduced with the (hand-crafted) optimal PMPs parameters and 100 trajectories with the optimal parameters for the DMPs. The PMP approach is able to reduce the variance at the movement if it is relevant for the task while the DMPs can only suppress the noise in the system throughout the trajectory in order to get an acceptable score. This advantage is also reflected by the average costs over 1000 trajectories.

a more commonly used policy search method, we also compare to the PI^2 approach (Theodorou et al., 2010a) which we could only evaluate for the DMP approach. We evaluated the learning performance in the case of no control noise, Figure 8.4(a), and in the case of control noise $\sigma = 20$, Figure 8.4(b). Without control noise the quality of the learned policy found by 2nd order search is similar for the DMPs and the PMPs. PI^2 could not find as good solutions as the stochastic search approach. The reason for this is that PI^2 could not find the very large weight values which are needed for the last few centers of the DMPs in order to have exactly zero velocity at the final state (note that the weights of the DMPs are multiplied by the phase variable s which almost vanishes in the end of the movement and therefore these weight values have to be very high). Because CMA policy search uses second order information, such large parameter values are easily found. This comparison clearly shows that using 2nd order search for policy search is justified. If we compare the learning speed in terms of required samples between DMPs and PMPs, we find an advantage for PMPs which could be learned an order of magnitude faster than the DMPs.

The second experiment (with control noise of $\sigma = 20$) was considerably harder to learn. Here, we needed to average each performance evaluation over 20 roll-outs. The use of more sophisticated extensions of CMA (Heidrich-Meisner and Igel, 2009a) which can deal with noisy performance evaluations and hence improve the learning speed of CMA policy search in the noisy setup is part of future work. In Figure 8.4(b) we find that the PMPs could be learned an order of magnitude faster than the DMPs. As expected from the earlier experiment, the PMPs could find clearly better solutions as the DMPs as they can adapt the variance of the trajectory to the task constraints. Again, PI^2 showed a worse performance than 2nd order search. Illustrated are mean values and standard deviations over 15 trials of learning (1034 ± 1.46 for the PMPs and 1876 ± 131 for the DMPs using CMA). To compare these results to the optimal costs we evaluated the best learned policies of both approaches and generated 1000 trajectories. The learned solution for the PMPs was similar to the hand-coded optimal solution, 1190 ± 584 versus costs of 1173 ± 596 for the optimal solution. DMPs achieved costs of 1478 ± 837, illustrating that, eventhough the DMPs are able to represent much better solutions with costs of 1286 ± 556 (see Figure 8.3), it is very hard to find this solution.

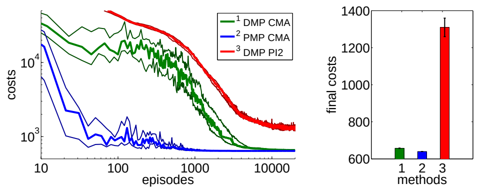
In Table 8.1, we show the mean and variance of the found parameters for the first via-point in comparison to the optimal PMP parameters. We can see that the found parameters closely matched the optimal ones. Interestingly, in the experiment with no noise, the found parameters had a larger deviation from the optimal ones, especially for the first via-point $g^{[1]}$ in Table 8.1. The reason for this is the simple observation that without noise, we can choose many via-points which results in the same trajectory, whereas with noise we have to choose the correct via-point in order to reduce the variance of the trajectory at this point in time.

Table 8.1: Learned parameters using PMPs for the via-point task (1st via-point), ± denotes the standard deviation.

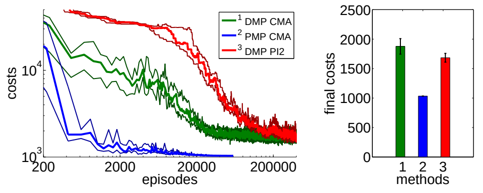
scenario	$d^{[1]}$	$g^{[1]}$	$\log(\mathbf{r}^{[1]})$	$\log(\mathbf{h}^{[1]})$
optimal	0.3	$-\mathbf{0.2}$	$[5, 0]$	-2.3
no noise	0.29 ± 0.01	$-\mathbf{0.27} \pm 0.03$	$[4.08 \pm 4.18, -0.8 \pm -0.77]$	-3.05 ± -4
with noise	0.29 ± 0.01	$-\mathbf{0.23} \pm 0.05$	$[4.93 \pm 5.29, -0.31 \pm -0.12]$	-2.85 ± -3

Next, we investigate the ability of both approaches to generalize to different situations. With generalization we mean that the same learned parameters can be re-used to generate different movements, e.g. used for different start or target states. The change of the initial state or the target state is also allowed by the DMP framework. However, how the movement is generalized to these new situations is based on heuristics (Pastor et al., 2009) and does not consider any task constraints.

In Figure 8.5 the learned policies are applied to reach different final targets $\phi_T \in [1.5, 1.25, 1, 0.75, 0.5]$. All plots show the mean trajectory. In order to change the final state of the movement we have to change the point attractor of the DMPs, which changes the complete trajectory. Due to this heuristic, the resulting DMP trajectories shown in Figure 8.5(a) do not pass through the via-point any more. Note that we use a modified version of the DMPs (Pastor et al., 2009) which has already been built for generalization to different target points. The PMPs on the other hand

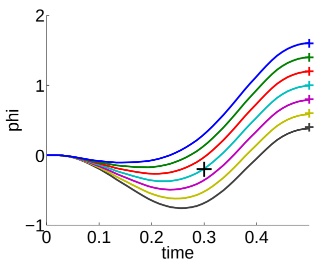


(a) Learning performance without noise

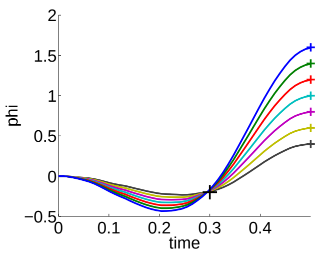


(b) Learning performance with noise

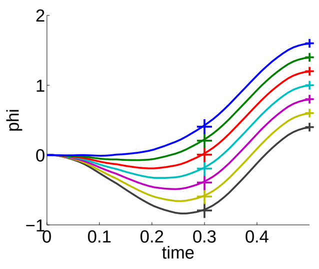
Figure 8.4: This figure illustrates the learning performance of the two movement representations, DMPs and PMPs, for the one-dimensional via-point task. Illustrated are mean values and standard deviations over 15 trials after CMA policy search. In addition, we also compare to the PI^2 approach (Theodorou et al., 2010a) which we could only evaluate for the DMP approach. Without noise the final costs of the two representations are similar if CMA policy search is used (a). In the second example (b) we use zero-mean Gaussian noise with $\sigma = 20$ for the controls. In this setup we needed to average each performance evaluation for CMA over 20 roll-outs. For both setups the PMPs could considerably outperform the DMPs in terms of learning speed. For the noisy setup the PMPs could additionally produce policies of much higher quality as they can adapt the variance of the trajectories to the task constraints. PI^2 could not find as good solutions as the CMA policy search approach in both setups.



(a) DMPs with varying goals



(b) PMPs with varying goals



(c) PMPs with varying goals and adapted via-points

Figure 8.5: In this experiment we evaluated the generalization of the learned policies to different goal states $\phi_T \in [1.5, 1.25, 1, 0.75, 0.5]$. Always the same parameters $\boldsymbol{\theta}$ have been used, i.e the parameters were not relearned. The DMPs (a) are not aware of task-relevant features and hence do not pass through the via-point any more. (b) PMPs can adapt to varying final goals with small effects on passing through the learned via-point. Furthermore the PMP representation is very flexible and we can also use a via-point $\tilde{g}_1 = g_1 + [0.5, 0.25, 0, -0.25, -0.5]$ with constant distance to the goal state to emulate the heuristic DMP behavior (c).

still navigate through the learned via-point when changing the goal state as shown in Figure 8.5(b). We can also adapt the via-point $\tilde{g}_1 = \tilde{g}_1 + [0.5, 0.25, 0, -0.25, -0.5]$ to encode, for example, a via-point which always has the same distance from the target state 8.5(c). This would somehow emulate the adaption of the trajectory used in the DMP approach. It is hard to argue which behavior is better suited for this task as we have not specified any cost function for the changed situations, however, the PMP approach offers much more control how the policy is changed. For generalization to different initial states the behavior is basically the same thus this evaluation is not shown here.

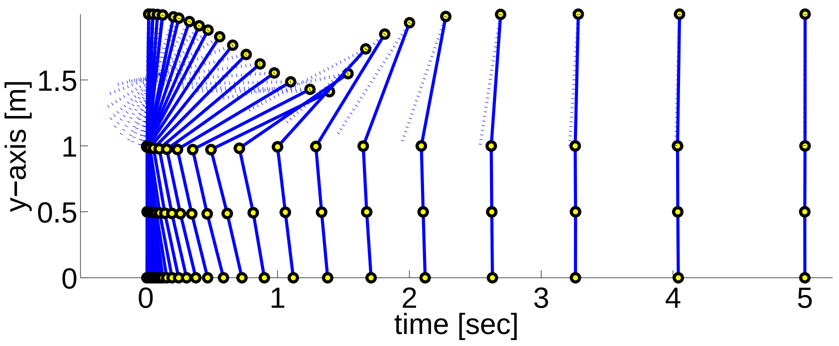


Figure 8.6: This figure illustrates a dynamic balancing movement learned using the proposed Planning Movement Primitives. The 4-link robot modelling a humanoid (70kg, 2m) gets pushed from behind with a specific force ($F = 25$Ns) and has to move such that it maintains balance. The optimal policy is to perform a fast bending movement and subsequently return to the upright robot posture. The circles denote the ankle, the knee, the hip and the shoulder joint.

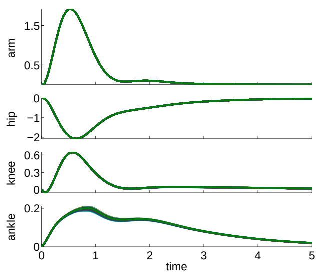
8.3.2 Dynamic humanoid balancing task

In order to assess the PMPs on a more complex task, we evaluate the PMP and DMP approach on a dynamic non-linear balancing task (Atkeson and Stephens, 2007) where a robot gets pushed with a specific force F and has to keep balance. The push results in an immediate change of the joint velocities. The motor torques are limited which makes direct counter-balancing of the force unfeasible. The optimal strategy is therefore to perform a fast bending movement and subsequently return to the upright position, see Figure 8.6. This is a very non-linear control problem, using any type of (linear) balancing control or local optimal control algorithm such as using AICO with the extrinsic cost function fails. Thus, we have to use a parametric movement representation. Like in the previous experiment, we take the Dynamic Movement Primitive (DMP) (Schaal et al., 2003) approach as a baseline.

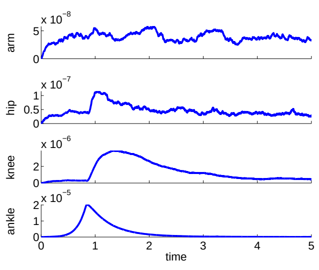
We use a 4-link robot as a simplistic model of a humanoid (70kg, 2m) (Atkeson and Stephens, 2007). The 8-dimensional state $\mathbf{x}_t$ is composed of the arm, the hip, the knee and the ankle positions and their velocities. Table 8.6 shows the initial velocities (resulting from the force F which always acts at the shoulder of the robot) and the valid joint angle range for the task. In all experiments the applied force was $F = 25$Ns. If one of the joints leaves the valid range the robot is considered to be fallen. If the robot manages to keep balance for 5s the episode is considered to be successful and the simulation is stopped. Additionally to the joint limits, the controls are limited to the intervals $[\pm 250, \pm 500, \pm 500, \pm 70]$Ns (arm, hip, knee and ankle). For more details we refer to (Atkeson and Stephens, 2007).

Let t_s be the last point in time where the robot has not fallen and let $\mathbf{x}_{t_s}$ be the last valid state. The final target state (upright position with zero velocity) is denoted by $\mathbf{x}_r$ and $T = 5$s. As extrinsic cost function $C(\tau)$ we use

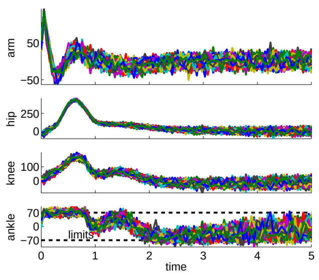
$$C(\tau) = 2 \cdot 10^3 (t_s - T)^2 + (\mathbf{x}_{t_s} - \mathbf{x}_r)^T \mathbf{R}_E (\mathbf{x}_{t_s} - \mathbf{x}_r) + \sum_{t=0}^{t_s} \mathbf{u}_t^T \mathbf{H}_E \mathbf{u}_t \ . \tag{8.6}$$



(a) Joint angles



(b) Variance of the joints



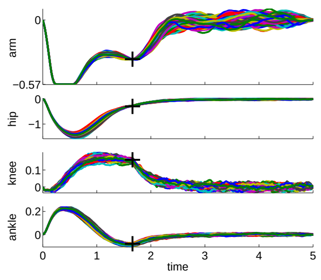
(c) Controls

Figure 8.7: The figure illustrates generated movements for the 4-link balancing task using DMPs. The controls were perturbed by zero-mean Gaussian noise with $\sigma = 10\text{Nm}$, the plots show 100 roll-outs using the same parameter setting $\boldsymbol{\theta}$. The trial-to-trial variability of the trajectories is shown in (b). This variance is determined by the learned controller gains of the inverse dynamics controller. As constant controller gains are used the variance can not be adapted during the movement. The noisy controls for all 100 roll-outs are illustrated in (c). This illustrated best available policy achieved costs of 568.

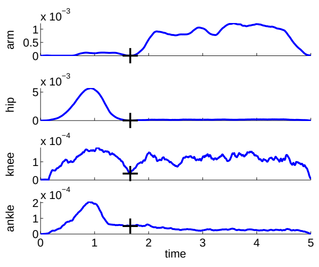
The first term $(t_s - T)^2$ is a punishment term for falling over. If the robot falls over, this term typically dominates. The precision matrix $\mathbf{R}_E$ determines how costly it is not to reach $\mathbf{x}_r$. The diagonal elements of $\mathbf{R}_E$ are set to 10^3 for joint angles and to 10 for joint velocities. Controls are punished by $\mathbf{H}_E = 5 \cdot 10^{-6}\mathbf{I}$. As we can see the extrinsic cost function cannot be directly encoded as a sum of intermediate costs which is usually required for stochastic optimal control algorithms. Therefore, we need to extract such a cost function in order to use a SOC planner.

We use additive zero-mean Gaussian noise with a variance $\sigma = 10$. In contrast to the simple via-point task here imitation learning fails for the DMPs. The best achieved policy using PMPs shown in Figure 8.8 is very close to the control and joint constraints and since the DMPs have no knowledge about these constraints, CMA policy search could not learn any control gain settings which fulfills them. (Although that the trajectories were first perfectly matched using imitation learning.) Therefore the illustrated DMP policy was learned from scratch and differs from the best PMP solution. Figure 8.7 illustrates 100 roll-outs of the best policies found by the DMP approach and Figure 8.8 shows 100 roll-outs of the PMP method. The second column in each figure illustrates the variance of the trajectories for the different roll-outs. While the DMPs cannot adapt the variance during the movement, the PMPs in Figure 8.8 can reduce the variance at the learned via-point (denoted by crosses). As the PMPs are able to control the variance of the trajectory, we can also see that the variance of the movement is much higher compared to the DMPs as accuracy only matters at the via-points. We can also see that the arm trajectory has a high variance after the robot is close to a stable up-right posture, see Figure 8.8(a), because it is not necessary to strictly control the arm in this phase. The best found policy of the DMPs had costs of 568 while the best result using PMPs was 307. This strongly suggests that it is advantageous to reduce the variance at certain points in time in order to improve the quality of the policy.

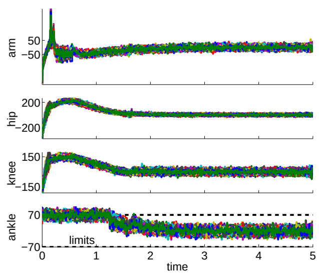
Next, we again want to assess the learning speed of both approaches. We again



(a) Joint angles



(b) Variance of the joints



(c) Controls

Figure 8.8: This figure illustrates the generated movements for the 4-link balancing task using PMPs. The crosses in (a) and (b) mark the learned via-point. As we can see the variance is minimized at these points, reflecting the importance to reach this point. (c) shows the noisy controls applied by the PMPs. The illustrated best available policy achieved costs of 307.

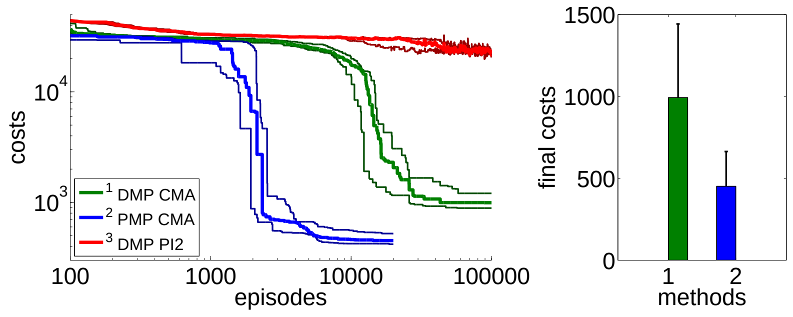
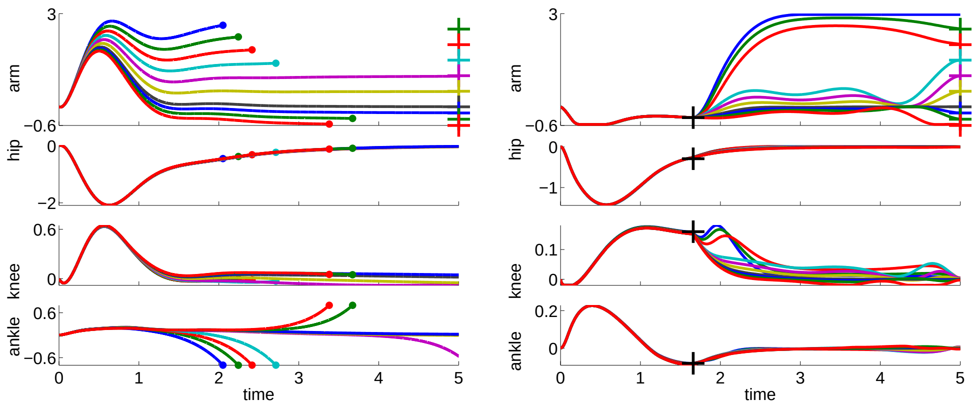


Figure 8.9: The figure illustrates the learning performance of the two movement representations, DMPs and PMPs for the 4-link balancing task. Illustrated are mean values and standard deviations over 20 trials after CMA policy search. The controls (torques) are perturbed by zero-mean Gaussian noise with $\sigma = 10$Nm. The PMPs are able to extract characteristic features of this task which is a specific posture during the bending movement, shown in Figure 8.8(a). Using the proposed Planning Movement Primitives good policies could be found at least one order of magnitude faster compared to the trajectory based DMP approach. Also, the quality of the best-found policy was considerably better for the PMP approach (993 ± 449 for the DMPs and 451 ± 212 for the PMPs). For the DMP approach we additionally evaluated PI^2 for policy search which could not find good policies.

used CMA policy search for the PMPs and DMPs as well as PI^2 for the DMP approach. The learning curves are illustrated in Figure 8.9. Using the PMPs as movement representation, good policies could be found at least one order of magnitude faster compared to the trajectory based DMP approach. The quality of the found policies was better for the PMP approach (mean values and standard deviations after learning: 993 ± 449 for the DMPs and 451 ± 212 for the PMPs). For the DMP approach we additionally evaluated PI^2 for policy search, however, PI^2 was not able to find good solutions—the robot always fell over.

In the next step we again test the generalization to different targets. We used the learned policies to generate movements to different final targets of the arm



(a) DMPs for changing targets (b) PMPs for changing targets

Figure 8.10: This figure illustrates the joint angle trajectories (arm, hip, knee and ankle) of a 4-link robot model during a balancing movement for different final targets of the arm joint ($[3, 2.5, 2, 1.5, 1, 0.5, 0, -0.2, -0.4, -0.6]$). The applied policies were learned for a final arm posture of $\phi_{T_{\text{arm}}} = 0$. (a) The valid range of the arm joint using DMPs is $\phi_{T_{\text{arm}}} \in \{-0.2, 1\}$. Large dots in the plot indicates that the robot has fallen. (b) PMPs could generate valid policies for all final arm configurations.

joint $\phi_{T_{\text{arm}}} \in [3, 2.5, 2, 1.5, 1, 0.5, 0, -0.2, -0.4, -0.6]$. Note that the used policy was learned for an final arm posture of $\phi_{T_{\text{arm}}} = 0$, we only change either the arm-position of the last via-point or the point attractor of the dynamical system. The results shown in Figure 8.10 confirm the findings of the one-dimensional via-point task. The PMPs first move to the via-point, always maintaining the extracted task constraints and afterwards move the arm to the desired position while keeping balance. All desired target positions of the arm could be fulfilled. In contrast, the DMPs managed to keep balance only for few target positions. The valid range of the target arm position with DMPs was $\phi_{T_{\text{arm}}} \in \{-0.2, 1\}$. This shows the advantage of generalization while keeping task constraints versus generalization per using the DMP heuristics.

So far all experiments for the PMPs were performed using the known model of the system dynamics, these experiments are denoted by PMP in Figure 8.11. Note that also for the DMPs the known system model has been used for inverse dynamics control. Now we want to evaluate how model learning affects the performance of our approach. This can be seen in Figure 8.11. In the beginning of learning the extrinsic costs are larger compared to motor skill learning with a given analytic model. However, as the number of collected data-points $\langle [\mathbf{x}_t; \mathbf{u}_t], \dot{\mathbf{x}}_t \rangle$ increases the PMPs with model learning quickly catch up and converge finally to the same costs. The PMP representation with model learning in parallel considerably outperforms the trajectory based DMP approach in learning speed and in the final costs.

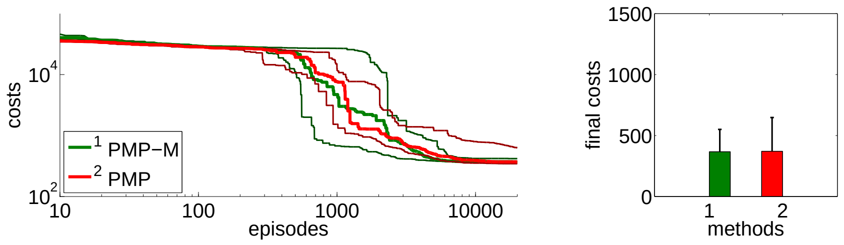


Figure 8.11: The figure shows the influence of model learning on the 4-link balancing task. Illustrated are mean values and standard deviations over 20 trials. The learning performance with the given system model is denoted by PMP. Instead of using the given model we now want to learn the system model from data (as described in Section 8.2.3). In the beginning of learning the extrinsic costs are larger compared to motor skill learning with a given analytic model. However, as the number of collected data-points $\langle [\mathbf{x}_t; \mathbf{u}_t], \dot{\mathbf{x}}_t \rangle$ increases the PMPs with model learning quickly catch up and converge finally to the same costs. The PMP representation with model learning in parallel considerably outperforms the trajectory based DMP approach in learning speed and in the final costs.

8.4 Conclusion and Future Work

We have proposed a new type movement representation which endows a movement primitive with an intrinsic probabilistic planning system instead of endowing a movement primitive with a dynamical system such as the widely used Dynamic Movement Primitives (DMPs) (Schaal et al., 2003) approach. While the dynamical system of a DMP is to some degree reactive to the environment—namely by adapting the temporal scaling factor and thereby de- or accelerating the movement execution as needed (Schaal et al., 2003)—the trajectory shape itself is fixed and non-reactive to the environment. In contrast, a movement primitive that is endowed with an intrinsic planning component can react to the environment by optimizing the trajectory for the specific current situation. Training such a movement primitive now means to train a planner to generate an appropriate trajectory in a given situation instead of training a dynamical system to generate a fixed (temporally flexible) reference trajectory.

Our approach to parameterize the intrinsic cost function is to use task-relevant features, such as the location of via-points or the importance of reaching this via-point. The idea is that such learnt task-relevant features as parameters of the movement representation should generalize well across situations. As our experiments show such an parametrization facilitates learning - good policies can be found an order of magnitude faster as with parametrizations which define the shape of the trajectory. It also allows an efficient generalization to new situations (e.g. new movement endpoints) because the planner always tries to fulfills the extracted task constraints. Once a motor skill is learned additional constraints like an unexpected appearing obstacle during a walking movement can directly be considered for modulating the behavior. Thus re-learning of the motor skill is not necessary since the planning machinery can integrate the new knowledge immediately. This properties will be further investigated for footplacement planning in future research.

Stochastic optimal control (SOC) is also an excellent method to describe human motor control (Todorov and Jordan, 2002). Thus, by the use of SOC methods already at the level of the primitive, our approach can implement many of these principles. For example, the *minimum intervention principle* implies that we should only intervene the system if it is necessary to fulfill a given task. A representation which strictly follows a desired trajectory by feedback control laws can not reproduce such a behavior. This has also been confirmed by our experiments, where the DMP representation produces sub-optimal policies in the presence of noise in our system. Only for deterministic systems the optimal solution can be represented by such an approach. In contrast the proposed Planning Movement Primitives could reproduce the optimal policy after learning.

An additional interesting aspect of using movement primitives is that, ideally, we want to be able to combine primitives in order to achieve several tasks simultaneously. This is still a mostly unsolved problem for current movement representations. Here, our Planning Movement Primitives offers new opportunities. For trajectory-based representation we would need to linearly combine two trajectories in order to combine two movements. As many task demands and system dynamics are non-linear such an approach usually fails. However, instead of linearly combining trajectories, our approach can now linearly combine cost functions—which results in a non-linear combination of the policies for the single tasks. The evaluation of this idea for combining several movements is also part of future work.

In this paper we focused on the representation of movement and put less emphasis on learning a movement. Yet, we want to point out again that our method does not depend on the used policy search method (we used the second order stochastic search method CMA), any episode-based exploring policy search method can be used. We also do not want to argue for using episode-based exploring methods for policy search, however, as our experiments show, these methods provide useful alternatives to the more commonly used step-based approaches such as the PI^2 (Theodorou et al., 2010a), the PoWER (Kober and Peters, 2010) or the eNAC algorithm (Peters and Schaal, 2008b). For policy search, future work will concentrate on extending the framework for learning in the case of changing initial conditions (Neumann, 2011) as well as using inference-based methods (Peters et al., 2010) also on the level of learning the parameters.

8.5 Acknowledgments

This paper was written by Elmar Rückert (ER), Gerhard Neumann (GN), Marc Toussaint (MT) and Wolfgang Maass (WM). It will be submitted at the end of november to the Journal of Machine Learning Research. GN provided most of the basic ideas and experimental design, ER implemented the algorithms and conducted most experiments. MT and WM considerably helped to improve the paper by their guidance and comments. Paper writting was mostly done by ER and GN with highly appreciated assistence from MT.

Appendix

Algorithms

In this section we review the evaluated algorithms.

Dynamic Movement Primitives

The most prominent representation for movement primitives used in robot control are the Dynamic Movement Primitives (DMP) (Schaal et al., 2003). We therefore used the DMPs as a baseline in our evaluations and will briefly review this approach in order to clarify differences to our work. For our experiments we implemented an extension of the original DMPs (Pastor et al., 2009), which considers an additional term in the dynamical system which facilitates generalization to different target states. For more details we refer to (Schaal et al., 2003; Pastor et al., 2009).

DMPs generate multi-dimensional trajectories by the use of non-linear differential equations. The basic idea is to a use for each degree-of-freedom (DoF) of the robot a globally stable, linear dynamical system which is modulated by learnable non-linear functions f :

$$\tau\dot{z} \quad = \quad \alpha_z\beta_z(g-y) - \alpha_z z - \alpha_z\beta_z(g-y_0)s + f, \tau\dot{y} = z,$$

where the desired final position of the joint is denoted by g and the initial position of the joint is denoted by y_0. The variables y and $\dot{y}$ denote a desired joint position and joint velocity, which represent our movement plan. The temporal scaling factor is denoted by τ and α_z and β_z are time constants. The non-linear function f directly modulates the derivative of the internal state variable z. Thus, f modulates the desired acceleration of the movement plan. s denotes the phase of the movement.

For each DoF of the robot an individual dynamical system, and hence an individual function f is used. The function f only depends on the phase s of a movement, which represents time, $\tau\dot{s} = -\alpha_s s$. The phase variable s is initially set to 1 and will converge to 0 for a proper choice of τ and α_s. With α_s we can modulate the desired movement speed. The function f is constructed of the weighted sum of K Gaussian basis functions Ψ_i

$$f(s) = \frac{\sum_{i=1}^K \Psi_i(s) w_i s}{\sum_{i=1}^K \Psi_i(s)}, \qquad \Psi_i(s) = \exp(-\frac{1}{2\sigma_i^2}(s-c_i)^2).$$

As the phase variable s converges to zero also the influence of f vanishes with increasing time. Hence, the dynamical system is globally stable with g as point attractor.

In our setting, only the linear weights w_i are parameters of the primitive which can modulate the shape of the movement. The centers c_i specify at which phase of the movement the basis function becomes active and are typically equally spaced in the range of s and not modified during learning. The bandwidth of the basis functions is given by σ_i^2.

Integrating the dynamical systems for each DoF results into a desired trajectory $\langle \mathbf{y}_t, \dot{\mathbf{y}}_t \rangle$ of the joint angles. We will use an inverse dynamics controller to follow this trajectory (Peters et al., 2008). The inverse dynamics controller receives the desired accelerations $\ddot{\mathbf{q}}_{\text{des}}$ as input and outputs the control torques $\mathbf{u}$. In order to calculate the desired accelerations we use a simple decoupled linear PD-controller

$$\ddot{\mathbf{q}}_{\text{des}} = \text{diag}(\mathbf{k}_{\text{pos}})(\mathbf{y}_t - \mathbf{q}_t) + \text{diag}(\mathbf{k}_{\text{vel}})(\dot{\mathbf{y}}_t - \dot{\mathbf{q}}_t).$$

Unfortunately standard inverse dynamics control did not work in our setup because we had to deal with control limits of multi-dimensional systems. Thus, we had to use an inverse dynamics controller which also incorporates control constraints. For this reason we performed an iterative gradient ascent using the difference between the actual (using constrainted controls) and the desired accelerations $\ddot{\mathbf{q}}_{\text{des}}$ as error function. This process was stopped after at most 25 iterations.

For our comparison, we will learn the linear weights $\mathbf{w}$ for each DoF as well as the controller gains $\mathbf{k}_{\text{pos}}$ and $\mathbf{k}_{\text{vel}}$, i.e. $\boldsymbol{\theta} = [\mathbf{w}_1, \ldots, \mathbf{w}_D, \mathbf{k}_{\text{pos}}, \mathbf{k}_{\text{vel}}]$. This results into $KD + 2D$ parameters for the movement representation, where D denotes the number of DoF of the robot.

Approximate Inference Control

The original formulation of the Approximate Inference Control (AICO) method (Toussaint, 2009) does not consider a linear term for the control costs. However, this is needed to encode torque limits, which are important for our dynamic balancing experiments, and hence, we needed to extend AICO.

The introduction of a linear term for the control costs yields not only in a modified cost function but also results in different update equations for the messages and finally in different equations of the optimal feedback controller. For completeness we will first recap the main steps to derive the AICO method and will then discuss the modifications to implement control constraints.

Approximate Inference Control without Torque Limits For motor planning we consider the stochastic process:

$$P(\mathbf{x}_{0:T}, \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) = P(\mathbf{x}_0) \prod_{t=0}^{T} P(\mathbf{u}_t|\mathbf{x}_t) \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \prod_{t=0}^{T} P(z_t|\mathbf{x}_t, \mathbf{u}_t).$$

where $P(\mathbf{u}_t|\mathbf{x}_t)$ denotes the state dependent prior for the controls, $P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ the state transition distribution and $P(\mathbf{x}_0)$ the initial state distribution. Here, we assume that the prior of the controls is independent of the states and thus we will simply use $P(\mathbf{u}_t|\mathbf{x}_t) = P(\mathbf{u}_t)$ for the rest of the appendix. The time horizon is fixed to T time-steps. The binary task variable z_t denotes a reward event, its probability is defined by $P(z_t = 1|\mathbf{x}_t, \mathbf{u}_t) \propto \exp(-c_t(\mathbf{x}_t, \mathbf{u}_t))$, where $c_t(\mathbf{x}_t, \mathbf{u}_t)$ is the intermediate cost function[1] for time step t. It expresses a performance criteria (like avoiding a collision, or reaching a goal).

We want to compute the posterior $P(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}|z_{1:T} = 1)$ over trajectories, conditioned on observing a reward ($z_t = 1$) at each time-step t. This posterior can be

[1]In this paper the immediate cost function is composed of the intrinsic costs and the constraint costs, i.e. $c_t(\mathbf{x}_t, \mathbf{u}_t) = l(\mathbf{x}_t, \mathbf{u}_t, t; \boldsymbol{\theta}) + c_p(\mathbf{x}_t, \mathbf{u}_t)$

computed by using message passing in the given graphical model of Figure 8.1. To simplify the computations we integrate out the controls:

$$P(\mathbf{x}_{t+1}|\mathbf{x}_t) = \int_{\mathbf{u}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) P(\mathbf{u}_t|\mathbf{x}_t) d\mathbf{u}_t, \tag{8.7}$$

The marginal belief $b_t(\mathbf{x}_t)$ of a state at time t is given by:

$$b_t(\mathbf{x}_t) = \alpha_t(\mathbf{x}_t)\beta_t(\mathbf{x}_t)\phi_t(\mathbf{x}_t), \tag{8.8}$$

where $\alpha_t(\mathbf{x}_t)$ is the forward message, $\beta_t(\mathbf{x}_t)$ is the backward message $\phi_t(\mathbf{x}_t)$ is the current task message. The messages are given by:

$$\alpha_t(\mathbf{x}_t) = \int_{\mathbf{x}_{t-1}} P(\mathbf{x}_t|\mathbf{x}_{t-1})\alpha_{t-1}(\mathbf{x}_{t-1})\phi_{t-1}(\mathbf{x}_{t-1})d\mathbf{x}_{t-1}, \tag{8.9}$$

$$\beta_t(\mathbf{x}_t) = \int_{\mathbf{x}_{t+1}} P(\mathbf{x}_{t+1}|\mathbf{x}_t)\beta_{t+1}(\mathbf{x}_{t+1})\phi_{t+1}(\mathbf{x}_{t+1})d\mathbf{x}_{t+1}, \tag{8.10}$$

$$\phi_t(\mathbf{x}_t) = P(z_t|\mathbf{x}_t). \tag{8.11}$$

We consider discrete-time, non-linear stochastic systems with zero mean Gaussian noise

$$P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}|f_{\text{Dyn}}(\mathbf{x}_t, \mathbf{u}_t), \mathbf{Q}_t).$$

The non-linear stochastic system f_{Dyn} is approximated by a Linear dynamics, Quadratic costs and Gaussian noise system (LQG) by Taylor expansion (Toussaint, 2009; Todorov and Li, 2005) :

$$P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{A}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}_t) \tag{8.12}$$

Thus, the system is linearized along a given trajectory $\langle \hat{\mathbf{x}}_{0:T}, \hat{\mathbf{u}}_{1:T} \rangle$ at every point in time. We will use f_t as shorthand for $f_{\text{Dyn}}(\mathbf{x}_t, \mathbf{u}_t)$. Then, the state transition matrices $\mathbf{A}_t$ are given by $\mathbf{A}_t = (\mathbf{I} + \frac{\delta f_t}{\delta \mathbf{x}_t}\Delta t)$, the control matrices $\mathbf{B}_t$ are given by $\mathbf{B}_t = \frac{\delta f_t}{\delta \mathbf{u}_t}\Delta t$ and the linear terms by $\mathbf{a}_t = (f_t - \frac{\delta f_t}{\delta \mathbf{x}_t}\mathbf{x}_t - \frac{\delta f_t}{\delta \mathbf{u}_t}\mathbf{u}_t)\Delta t$.

In the original formulation of AICO the cost function c_t is approximated as :

$$c_t(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T\mathbf{R}_t\mathbf{x}_t - 2\mathbf{r}_t^T\mathbf{x}_t + \mathbf{u}_t^T\mathbf{H}_t\mathbf{u}_t.$$

Note that there is no linear term for the control costs as we only punish quadratic controls. We can now write $P(z_t = 1|\mathbf{x}_t, \mathbf{u}_t) = P(z_t = 1|\mathbf{x}_t)P(\mathbf{u}_t)$ as

$$P(z_t = 1|\mathbf{x}_t) \propto \mathcal{N}[\mathbf{x}_t|\mathbf{r}_t, \mathbf{R}_t] \tag{8.13}$$

$$P(\mathbf{u}_t) = \mathcal{N}[\mathbf{u}_t|\mathbf{0}, \mathbf{H}_t], \tag{8.14}$$

where the distributions in Equation 8.13 and 8.14 are given in canonical form. The canonical form of a Gaussian is used because numerical operations such as products or integrals are easier to calculate in this notation. The canonical form is indicated by the *square* bracket notation and given by

$$\mathcal{N}[\mathbf{x}|\mathbf{a}, \mathbf{A}] = \frac{\exp(-1/2\mathbf{a}^T\mathbf{A}^{-1}\mathbf{a})}{|2\pi\mathbf{A}^{-1}|^{1/2}}\exp(-1/2\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{a}).$$

A Gaussian in normal form can always be transformed into the canonical form by $\mathcal{N}(\mathbf{x}|\mathbf{a},\mathbf{A}) = \mathcal{N}[\mathbf{x}|\mathbf{A}^{-1}\mathbf{a},\mathbf{A}^{-1}]$. For more details we refer to the Gaussian Identities in (Toussaint, 2011).

We can see in Equation 8.14 that our prior for applying the control $\mathbf{u}_t$ is given by the control costs, i.e. $\mathcal{N}[\mathbf{u}_t|\mathbf{0},\mathbf{H}_t]$. By integrating out the controls from our system dynamics we get the following state transition probabilities

$$\begin{aligned} P(\mathbf{x}_{t+1}|\mathbf{x}_t) &= \int_{\mathbf{u}_t} \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{A}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}_t)\mathcal{N}[\mathbf{u}_t|\mathbf{0},\mathbf{H}_t]d\mathbf{u}_t && (8.15)\\ &= \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{A}_t\mathbf{x}_t + \mathbf{a}_t, \mathbf{Q}_t + \mathbf{B}_t\mathbf{H}_t^{-1}\mathbf{B}_t), && (8.16)\end{aligned}$$

where the integral was solved using a reformulation of the *Propagation* rule in (Toussaint, 2011).

As we can see, all distributions in the approximated LQG system in Equation 8.16 are Gaussian, and thus, also all messages are Gaussians and can be calculated analytically. The resulting messages are given in (Toussaint, 2009).

Approximate Inference Control with Torque Limits In order to implement torque and joint limits we introduce an additional cost function c_p which punishes the violation of the given constraints. The function c_p is just added to the current immediate costs. We use separate cost terms for control constraints c_t^u and joint constraints c_t^q, i.e $c_p(\mathbf{x}_t,\mathbf{u}_t) = c_t^q(\mathbf{x}_t) + c_t^u(\mathbf{u}_t)$. Here, we will only discuss how to implement the function $c_t^u(\mathbf{u}_t)$ for the torque constraints, joint constraints are implemented similarly.

The cost function c_t^u is quadratic in $\mathbf{u}$ and punishes leaving the valid control limits of $\mathbf{u}$. In order to implement the upper bound $\mathbf{u}_{\max}$ for the torques, we use the following cost function

$$\begin{aligned} c_t^u(\mathbf{u}_t) &= \mathbf{u}_t^T\mathbf{H}_t\mathbf{u}_t + (\mathbf{u}_t - \mathbf{u}_{\max})^T\mathbf{H}_t^U(\mathbf{u}_t - \mathbf{u}_{\max}),\\ &= \mathbf{u}_t^T\mathbf{H}_t\mathbf{u}_t + \mathbf{u}_t^T\mathbf{H}_t^U\mathbf{u}_t - 2\mathbf{u}_{\max}^T\mathbf{H}_t^U\mathbf{u}_t + \mathbf{u}_{\max}^T\mathbf{H}_t^U\mathbf{u}_{\max},\\ &= \mathbf{u}_t^T\mathbf{H}_t\mathbf{u}_t + \mathbf{u}_t^T\mathbf{H}_t^U\mathbf{u}_t - 2\mathbf{u}_{\max}^T\mathbf{H}_t^U\mathbf{u}_t + \text{const}.\end{aligned}$$

As before, the matrix $\mathbf{H}_t$ denotes the quadratic control costs. The constrained costs are only imposed for the control variable u_i if the torque value exceeds the upper bound $u_{\max,i}$. In order to do so $\mathbf{H}_t^U$ is a diagonal matrix where the ith diagonal entry is zero if $u_i \le u_{\max,i}$ and non-zero otherwise. The lower bound $\mathbf{u}_{\min}$ is implemented likewise using an individual diagonal matrix $\mathbf{H}_t^L$.

We can again implement $c_t^u(\mathbf{u}_t)$ as prior distribution $P(\mathbf{u}_t)$ for the controls.

$$P(\mathbf{u}_t) \propto \mathcal{N}[\mathbf{u}_t|\mathbf{h}_t,\mathbf{H}_t], \tag{8.17}$$

where $\mathbf{h}_t = \mathbf{u}_{\max}^T\mathbf{H}_t^U + \mathbf{u}_{\min}^T\mathbf{H}_t^L$ and the precision $\hat{\mathbf{H}}_t = \mathbf{H}_t + \mathbf{H}_t^U + \mathbf{H}_t^L$. *As we can see, the linear term* $\mathbf{h}_t$ *of the prior distribution* $P(\mathbf{u}_t)$ *is now non-zero. This yields different message equations.*

Joint-limits can be imposed similarly by using additional terms costs for $c_t^q(\mathbf{x}_t)$. However, for joint limits the update equations stay the same because $P(z_t = 1|\mathbf{x}_t)$ has already a non-zero mean denoted by $\mathbf{r}_t$ in Equation 8.13.

To derive the messages we will first integrate out the controls to get the state transition probabilities:

$$\begin{aligned} P(\mathbf{x}_{t+1}|\mathbf{x}_t) &= \int_{\mathbf{u}_t} \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{A}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{B}_t\mathbf{u}_t, \mathbf{Q}_t)\mathcal{N}[\mathbf{u}_t|\mathbf{h}_t, \hat{\mathbf{H}}_t]d\mathbf{u}_t, \\ &= \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{A}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{B}_t\hat{\mathbf{H}}_t^{-1}\mathbf{h}_t, \mathbf{Q}_t + \mathbf{B}_t\hat{\mathbf{H}}_t^{-1}\mathbf{B}_t^T). \qquad (8.18) \end{aligned}$$

Note that, since the cost function $c_t^u(\mathbf{u}_t)$ contains a non-zero linear term $\mathbf{h}_t$, we get a new linear term $\hat{\mathbf{a}}_t = \mathbf{a}_t + \mathbf{B}_t\mathbf{H}_t^{-1}\mathbf{h}_t$ in the transition dynamics. The forward and the backward messages are the same like in (Toussaint, 2009) except that $\mathbf{a}_t$ is replaced by $\hat{\mathbf{a}}_t$ and $\mathbf{H}_t$ by $\hat{\mathbf{H}}_t$.

Like in (Toussaint, 2009) we use the canonical representations for the forward and the backward message:

$$\begin{aligned} \alpha_t(\mathbf{x}_t) &= \mathcal{N}[\mathbf{x}_t|\mathbf{s}_t, \mathbf{S}_t] \\ \beta_t(\mathbf{x}_t) &= \mathcal{N}[\mathbf{x}_t|\mathbf{v}_t, \mathbf{V}_t] \\ \phi_t(\mathbf{x}_t) &= P(z_t|\mathbf{x}_t) = \mathcal{N}[\mathbf{x}_t|\mathbf{r}_t, \mathbf{R}_t]. \end{aligned}$$

The messages are represented by Gaussians in canonical form, for which mathematical operations like products are simply performed by adding the linear terms and the precisions. The mean of the belief is given by $b_t(\mathbf{x}_t) = (\mathbf{S}_t + \mathbf{V}_t + \mathbf{R}_t)^{-1}(\mathbf{s}_t + \mathbf{v}_t + \mathbf{r}_t)$ (multiplying three canonical messages and a subsequent transformation to normal form). Furthermore we use the shorthand $\bar{\mathbf{Q}}_t = \mathbf{Q}_t + \mathbf{B}_t\hat{\mathbf{H}}_t^{-1}\mathbf{B}_t^T$ for the covariance in Equation 8.18.

The messages are computed by inserting the state transition probabilities given in Equation 8.18 in the message passing Equations 8.9 and 8.10. Subsequently the integrals are solved using the *Propagation* rule in (Toussaint, 2011). The final equations in canonical form are:

$$\begin{aligned} \mathbf{S}_t &= (\mathbf{A}_{t-1}^{-T} - \mathbf{K}_s)\mathbf{S}_{t-1}\mathbf{A}_{t-1}^{-1}, && (8.19) \\ \mathbf{s}_t &= (\mathbf{A}_{t-1}^{-T} - \mathbf{K}_s)(\bar{\mathbf{s}}_{t-1} + \mathbf{S}_{t-1}\mathbf{A}_{t-1}^{-1}(\hat{\mathbf{a}}_{t-1} + \mathbf{B}_{t-1}\hat{\mathbf{H}}_{t-1}^{-1}\mathbf{h}_{t-1})), && (8.20) \\ \mathbf{K}_s &= \mathbf{A}_{t-1}^{-T}\mathbf{S}_{t-1}(\mathbf{S}_{t-1} + \mathbf{A}_{t-1}^{-T}\bar{\mathbf{Q}}_{t-1}^{-1}\mathbf{A}_{t-1}^{-1})^{-1}. && (8.21) \end{aligned}$$

And for the backward messages:

$$\begin{aligned} \mathbf{V}_t &= (A_t^T - \mathbf{K}_v)\bar{\mathbf{V}}_{t+1}\mathbf{A}_t, && (8.22) \\ \mathbf{v}_t &= (\mathbf{A}_t^T - \mathbf{K}_v)(\bar{\mathbf{v}}_{t+1} - \bar{\mathbf{V}}_{t+1}(\hat{\mathbf{a}}_t + \mathbf{B}_t\hat{\mathbf{H}}_t^{-1}\mathbf{h}_t)), && (8.23) \\ \mathbf{K}_v &= \mathbf{A}_t^T\bar{\mathbf{V}}_{t+1}(\bar{\mathbf{V}}_{t+1} + \bar{\mathbf{Q}}_t^{-1})^{-1}. && (8.24) \end{aligned}$$

To obtain the same mathematical form as in (Toussaint, 2009) one needs to apply the Woodbury identity and reformulate the equations. In contrast to the update message in normal form (Toussaint, 2009), direct inversions of $\bar{\mathbf{S}}_{t-1}$ and $\bar{\mathbf{V}}_{t+1}$ are not necessary in the canonical form and therefore, the iterative updates are numerically more stable.

Finally, in order to compute the optimal feedback controller we calculate the joint state-control posterior

$$P(\mathbf{u}_t, \mathbf{x}_t) = P(\mathbf{u}_t, \mathbf{x}_t|z_t = 1)$$

$$\begin{aligned}
P(\mathbf{u}_t, \mathbf{x}_t) &= \int_{\mathbf{x}_{t+1}} \alpha_t(\mathbf{x}_t)\phi_t(\mathbf{x}_t)P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)P(\mathbf{u}_t)\beta_{t+1}(\mathbf{x}_{t+1})\phi_{t+1}(\mathbf{x}_{t+1})d\mathbf{x_{t+1}} \\
P(\mathbf{u}_t, \mathbf{x}_t) &= P(\mathbf{x}_t)P(\mathbf{u}_t)\int_{\mathbf{x}_{t+1}} P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)\mathcal{N}[\mathbf{x}_{t+1}|\bar{\mathbf{v}}_{t+1}, \bar{\mathbf{V}}_{t+1}]d\mathbf{x_{t+1}} \\
P(\mathbf{u}_t|\mathbf{x}_t) &= \mathcal{N}(\mathbf{u}_t|\mathbf{M}_t^{-1}(\mathbf{B}_t^T\mathbf{V}_*(\bar{\mathbf{V}}_{t+1}^{-1}\bar{\mathbf{v}}_{t+1} - \mathbf{A}_t\mathbf{x}_t - \hat{\mathbf{a}}_t) + \mathbf{h}_t), \mathbf{M}_t^{-1}),
\end{aligned}$$

where $\mathbf{V}_* = (\mathbf{Q} + \bar{\mathbf{V}}_{t+1}^{-1})^{-1}$ and $\mathbf{M}_t = \mathbf{B}_t^T\mathbf{V}_*\mathbf{B}_t + \hat{\mathbf{H}}_t$. After a reformulation we can obtain an optimal feedback controller of the form $\mathbf{u}_t = \mathbf{o}_t + \mathbf{O}_t\mathbf{x}_t$ with

$$\mathbf{o}_t = \mathbf{M}_t^{-1}(\mathbf{B}_t^T\mathbf{V}_*\bar{\mathbf{V}}_{t+1}^{-1}\bar{\mathbf{v}}_{t+1} - \mathbf{B}_t^T\mathbf{V}_*\mathbf{a}_t + \mathbf{h}_t), \tag{8.25}$$

$$\mathbf{O}_t = -\mathbf{M}_t^{-1}\mathbf{B}_t^T\mathbf{V}_*\mathbf{A}_t. \tag{8.26}$$

Similar to (Toussaint, 2009), we use an iterative message passing approach where we approximate the non-linear system by an Linear dynamics, Quadratic costs and Gaussian noise system (LQG) at the new mode of the trajectory. In (Toussaint, 2009), this is done by using a learning rate on the state beliefs $b(\mathbf{x}_t)$. However, in difference to (Toussaint, 2009), we also need an estimate of the optimal action $\mathbf{u}_t$ in order to impose the control constraints. Using a learning rate on the control action $\mathbf{u}_t$ turned out to be very ineffective because feedback is extenuated. For this reason we will use a learning rate on the feedback controller. We simulate the LQG system (using the linearized model) to get a new mode of the belief of the trajectory. The complete message passing algorithm considering state and control constraints is listed in Algorithm 2. This is a straightforward implementation of Gaussian message passing in linearized systems, similar to an extended Kalman smoother. In (Toussaint, 2009) or (Rawlik et al., 2010) more time efficient methods are presented, where for each time step the belief is updated until convergence in contrast to updating all messages and iterating until the intrinsic costs $L(\tau;\boldsymbol{\theta})$ converge. The computational benefits of such an approach still needs to be evaluated for our messages.

Task settings and parameters

In this section the movement primitive parameters and constants are specified for the one-dimensional via-point task and for the humanoid balancing task.

One-dimensional via-point task

For the one-dimensional via-point task the parameters of the Dynamic Movement Primitives are listed in Table 8.2. The valid configuration space for the policy search algorithm is listed in Table 8.3. The CMA policy search algorithm has just one parameter, the exploration rate. Where the best exploration rate using DMPs for this task found was 0.05.

The limits of the parametrization of the Planning Movement Primitives (see Equation 8.4) is listed in Table 8.4. For the via-point task we choose $N = 2$, where the second via-point $g^{[N]} = g_T$ was given. The exploration rate was set to 0.1 in all experiments.

Algorithm 2: Approximate Inference Control for Constrained Systems

```
Data: initial trajectory $\hat{\mathbf{x}}_{0:T}$, learning rate $\eta$
Result: $\mathbf{x}_{0:T}$ and $\mathbf{u}_{0:T}$
initialize $\mathbf{S}_0 = 1e10 \cdot \mathbf{I}$, $\mathbf{s}_0 = \mathbf{S}_0\mathbf{x}_0$, $k = 0$, $\hat{\mathbf{o}}_{0:T} = \mathbf{0}$, $\hat{\mathbf{O}}_{0:T} = 0 \cdot \mathbf{I}$ ;
while $L(\tau; \boldsymbol{\theta})$ not converged do
    for $t \leftarrow 0$ to $T$ do
        Linearize Model: $\mathbf{A}_t, \mathbf{a}_t, \mathbf{B}_t$ using Equation 8.12
        Compute Costs: $\hat{\mathbf{H}}_t, \mathbf{h}_t, \mathbf{R}_t, \mathbf{r}_t$ using Equation 8.13, 8.17

    for $t \leftarrow 1$ to $T$ do
        Forward Messages: $\alpha_t(\mathbf{x}_t)$ using Equation 8.19 - 8.21

    for $t \leftarrow T - 1$ to $0$ do
        Backward Messages: $\beta_t(\mathbf{x}_t)$ using Equation 8.22 - 8.24

    for $t \leftarrow 0$ to $T$ do
        Feedback Controller: $\mathbf{o}_t, \mathbf{O}_t$ using Equation 8.25, 8.26
        if $k == 0$ then
            $\mathbf{u}_t = \mathbf{o}_t + \mathbf{O}_t\mathbf{x}_t$
        else
            $\hat{\mathbf{o}}_t = (1 - \eta)\hat{\mathbf{o}}_t + \eta\mathbf{o}_t$
            $\hat{\mathbf{O}}_t = (1 - \eta)\hat{\mathbf{O}}_t + \eta\mathbf{O}_t$
            $\mathbf{u}_t = \hat{\mathbf{o}}_t + \hat{\mathbf{O}}_t\mathbf{x}_t$
        $\mathbf{x}_{t+1} = \mathbf{A}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{B}_t\mathbf{u}_t$
    $k = k + 1$
```

Table 8.2: Via-point task: DMP movement primitive parameters

K	α_s	α_z	β_z	τ
10	1	2	0.9	0.1

Dynamic humanoid balancing task

The DMP parameters for the balancing task are listed in Table 8.5. The policy search parameters are the same like for the via-point task, Table 8.3. The exploration

Table 8.3: Via-point task: DMP policy search configuration parameters

	w	$\mathbf{k_{pos}}$	$\mathbf{k_{vel}}$
lower bound	-100	0	0
upper bound	$+100$	100	100

Table 8.4: Via-point task: PMP policy search configuration parameters with $i = 1, 2$

	$d^{[1]}$	$g^{[1]}$	$\mathbf{r}^{[i]}$	$\mathbf{h}^{[i]}$
lower bound	0.05	-2	$[1, 10^{-6}]$	10^{-4}
upper bound	0.4	+2	$[10^{6}, 10^{4}]$	10^{-2}

Table 8.5: Balancing task: DMP movement primitive parameters

$\mathbf{K}$	α_s	α_z	β_z	τ
10	1	5	5	1

rate was set to 0.1.

The PMPs were again evaluated with $N = 2$ via-points, where the second via-point $g^{[N]} = g_T$ (the up-right robot posture) was given and for the first via-point the valid joint angle configuration is shown in Table 8.6. The exploration rate was 0.1 and the policy search algorithm configuration is listed in Table 8.7.

Table 8.6: Joint angle configurations where a robot gets pushed by a specific force F.

Joint	**Init velocities**	**Lower Bound**	**Upper Bound**
arm	$-0.4 \cdot 10^{-2} F$	-0.6	3.0
hip	$+5.1 \cdot 10^{-2} F$	-2.0	0.1
knee	$-7.4 \cdot 10^{-2} F$	-0.05	2.5
ankle	$+1.2 \cdot 10^{-2} F$	-0.8	0.8

Table 8.7: Balancing task: PMP policy search configuration parameters with $i = 1, 2$

	$d^{[1]}$	$\mathbf{r}^{[i]}$	$\mathbf{h}^{[i]}$
lower bound	0.1	$[10^{-2}, 10^{-4}, 10^{-2}, 10^{-4}, 10^{-2}, 10^{-4}, 10^{-2}, 10^{-4}]$	$[10^{-9}, 10^{-9}, 10^{-9}, 10^{-9}]$
upper bound	4.6	$[10^{4}, 10^{2}, 10^{4}, 10^{2}, 10^{4}, 10^{2}, 10^{4}, 10^{2}]$	$[10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}]$

Part III

Policy Search

Chapter 9

Introduction

In this section we briefly discuss policy search methods for motor skill learning. We will put our focus on the temporal movement primitives discussed in Chapter 5 as they are the most widely used. A temporal movement representation uses the duration (or phase) of the movement to encode the state of the robot, i.e. the policy now explicitly depends on the time $\pi(\mathbf{a}|\mathbf{s}, t; \mathbf{w})$, where $\mathbf{w}$ are the parameters of the primitive. Temporal primitives are typically used for episodic tasks, i.e. we always use the same initial conditions for each episode. In this setup we do not necessarily have to estimate a value-function because we can directly search for optimal primitive parameters $\mathbf{w}^*$. This chapter gives a broad overview of existing policy search methods.

In the next chapter we will introduce a new method which is able to generalize policy search to changing several situations. This work has already been published in the paper 'Variational Inference for Policy Search in Changing Situations', appeared in the Proceedings of the International Conference for Machine Learning (ICML), 2011.

9.1 Episodic Policy Search for temporal Movement Primitives

In the episodic setup we can neglect the value-function because the expected reward of using parameter vector $\mathbf{w}$ can be estimated directly by performing roll-outs on the real (or simulated) robot

$$J(\mathbf{w}) \quad = \quad \int_\tau p(\tau; \mathbf{w}) R(\tau) d\tau, \tag{9.1}$$

where a trajectory τ is given by a sequence of state and action vectors, i.e $\tau = \langle \mathbf{s}_0, \mathbf{a}_0, \dots, \mathbf{s}_T \rangle$ and $R(\tau)$ denotes the summed reward associated with this trajectory. Note that this type f performance evaluation is also possible for the non-episodic setup with multiple start states, however, we would have to estimate the costs by averaging over all possible start states, which is very inefficient. The summed reward $R(\tau)$ is typically composed of the summed immediate rewards r_t and a final reward for the last time step ϕ_T, i.e.

$$R(\tau) = \sum_{t=0}^{T-1} r(\mathbf{s}_t, \mathbf{a}_t) + \phi_T(\mathbf{s}_T)$$

Our goal is now to find a parameter vector $\mathbf{w}^*$ which minimizes $J(\mathbf{w})$.

There are many approaches which can be used for this task, some are gradient-based (Williams, 1992; Peters and Schaal, 2006), expectation-maximization (Kober and Peters, 2010) or inference-based (Theodorou et al., 2010b) and some are stochastic optimizers (Heidrich-Meisner and Igel, 2009b; Hansen et al., 2003). These methods can coarsely be divided into episode-based exploring and step-based exploring approaches. Step-based exploring methods apply exploration at each time step by performing noisy actions while Episode-based exploring approaches explore the parameter space by using different parameter vectors for each episode.

9.2 Step-based Exploring Approaches

Step-based exploring algorithms include traditional policy gradient methods such as episodic REINFORCE (Williams, 1992) and the episodic Natural Actor Critic algorithm (Peters and Schaal, 2006, 2008a), expectation-maximization based algorithms such as PoWER (Kober and Peters, 2010) and algorithms based on path integrals such as PI^2 (Theodorou et al., 2010b). PoWER and PI^2 are currently considered to be state of the art.

The main principle of step-based approaches is that an exploration-noise $\boldsymbol{\varepsilon}_t$ for the action $\mathbf{a}_t$ is used for each time step to search for trajectories with low costs. Subsequently the parameters $\mathbf{w}$ are adapted such that the (noisy) trajectories with lower costs are more likely to be reproduced again. Step-based exploring approaches only work for policies which depend linearly on the parameters $\mathbf{w}$ as we have to be able to easily fit the parameters to the noisy trajectories. Of all the discussed movement representations in Chapter 5, the linear policy representation applies only for the Dynamic Movement Primitive (DMP) approach. The DMPs typically use a Gaussian policy for the desired acceleration where the mean $\boldsymbol{\mu}$ depends linearly on the parameters $\mathbf{w}$ (see Section 5.2.1).

The exploration scheme is quite simple. All algorithms either add the noise term ε directly to the action, i.e

$$\mathbf{a}_t = \boldsymbol{\Phi}_t^T \mathbf{w} + \varepsilon_t^a, \qquad \pi(\mathbf{a}_t|\mathbf{s}_t, t; \mathbf{w}) = \mathcal{N}\left(\mathbf{a}_t|\boldsymbol{\Phi}_t^T \mathbf{w}, \boldsymbol{\Sigma}_a\right) \tag{9.2}$$

or to the linear parameters $\mathbf{w}$, i.e.

$$\mathbf{a}_t = \boldsymbol{\Phi}_t^T (\mathbf{w} + \varepsilon_t^w), \qquad \pi(\mathbf{a}_t|\mathbf{s}_t, t; \mathbf{w}) = \mathcal{N}\left(\mathbf{a}_t|\boldsymbol{\Phi}_t^T \mathbf{w}, \boldsymbol{\Phi}_t^T \boldsymbol{\Sigma}_w \boldsymbol{\Phi}_t\right), \tag{9.3}$$

Adding noise to the parameters has become more accepted in recent algorithms like PoWER (Kober and Peters, 2010) and PI^2 (Theodorou et al., 2010b).

The noise $\boldsymbol{\varepsilon}_t$ itself is always sampled from a multivariate normal distribution with a diagonal covariance matrix $\boldsymbol{\Sigma}_\varepsilon = \mathrm{diag}(\boldsymbol{\sigma})$. Thus, there is no correlation in the exploration noise, and consequently exploration is always undirected. In addition, the uncorrelated exploration noise is usually not adapted during learning. Another problem of step-based exploration techniques is that we add noise at each time step which often results in high variance of the estimate of the value of a specific parameter vector $\mathbf{w}$, in particular if the time horizon T is large.

9.2.1 Policy Gradient Methods

The principle of policy gradient (PG) methods is simple. The parameters are updated in the direction of the gradient of our expected reward function

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha \nabla_{\mathbf{w}} J(\mathbf{w}),$$

where α is a learning rate which has to be specified by the user.

Typically PG-methods use Gaussian noise directly added to the current action, i.e. $\mathbf{a}_t = \mathbf{\Phi}_t^T \mathbf{w} + \varepsilon_t^a$. PG algorithms only differ how they estimate the gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$.

REINFORCE

REINFORCE is one of the first policy gradient algorithms (Williams, 1992). The algorithm calculates the gradient of the expected reward (Equation 9.1) by using the well-known 'log-ratio trick' $\nabla_{\mathbf{w}} p(\tau; \mathbf{w}) = p(\tau; \mathbf{w}) \nabla_{\mathbf{w}} \log p(\tau; \mathbf{w})$, i.e.

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \int_\tau \nabla_{\mathbf{w}} p(\tau; \mathbf{w}) R(\tau) d\tau = \int_\tau p(\tau; \mathbf{w}) \nabla_{\mathbf{w}} \log p(\tau; \mathbf{w}) R(\tau) d\tau$$

Now, if we use the identity

$$p(\tau; \mathbf{w}) = P(\mathbf{s}_0) \prod_{t=0}^{T-1} P(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t | \mathbf{s}_t; \mathbf{w})$$

, we can reduce the product to a sum as we are using the log term of $p(\tau; \mathbf{w})$. Consequently all terms drop out which do not depend on $\mathbf{w}$ for the derivative. The REINFORCE gradient is therefore given by

$$\nabla_{\mathrm{RF}} J(\mathbf{w}) = \int_\tau R(\tau) \sum_{t=0}^{T-1} \nabla_{\mathbf{w}} \log \pi(\mathbf{a}_t | \mathbf{s}_t; \mathbf{w}) d\tau.$$

Fortunately, the gradient only depends on the derivative of the policy, which is usually known to the experimenter. No knowledge of the transition or reward model is needed.

The expectation over the trajectories can be replaced by sample trajectories. It has been also shown that the substraction of a reward baseline b from the trajectory reward $R(\tau)$ can improve the performance significantly (Greensmith et al., 2004). Further advancements of REINFORCE are the GPOMDP (Baxter and Bartlett, 1999) and the Policy Gradient Theorem algorithm (Sutton et al., 1999). Both algorithms use the simple observation that rewards in the past are not affected by actions in the future, which results in a gradient calculation with reduced variance in comparison to REINFORCE.

9.2.1.1 Episodic Natural-Actor Critic

The episodic Natural-Actor Critic (eNAC) (Peters and Schaal, 2008a) algorithm is one of the most efficient policy gradient methods. The algorithm is inspired by the use of natural gradients for supervised learning (Amari, 1998). Here, the standard gradient is projected onto more efficient update directions by the use of the inverse Fisher Information matrix. The Fisher Information matrix measures the amount of 'information' that the trajectory distribution $p(\tau; \mathbf{w})$ carries about the parameter vector $\mathbf{w}$, it is given by

$$F(\mathbf{w}) \quad = \quad \mathbb{E}_\tau \left[\nabla_{\mathbf{w}} p(\tau; \mathbf{w}) \nabla_{\mathbf{w}} p(\tau; \mathbf{w})^T \right].$$

Using the inverse Fisher Information matrix to project the standard REINFORCE gradient has the effect that the change of each parameter w_i has now the same influence on the trajectory distribution. Thus, the algorithm is invariant to the choice of the parametrization (e.g. scale) if two parametrizations have the same representational power. The resulting gradient estimation is given by

$$\nabla_{\mathrm{NAC}} J(\mathbf{w}) \quad = \quad F(\mathbf{w})^{-1} \nabla_{\mathrm{RF}} J(\mathbf{w}).$$

Experiments showed that due to the scaling of the REINFORCE gradient the eNAC converges an order of magnitude faster than the standard REINFORCE algorithm.

9.2.2 Inference-based algorithms

There are 2 types of inference-based algorithms, Monte-Carlo Expectation-Maximization (MC-EM) algorithms such as PoWER (Kober and Peters, 2010) and MC-EM Policy Search (Vlassis et al., 2009) and algorithms based on path-integrals such as PI^2 (Theodorou et al., 2010b). The exact derivation of these algorithms is out of the scope of this introduction, we will only briefly sketch some interesting properties. The methods apply the noise term $\boldsymbol{\varepsilon}_t^w$ directly to the parameter vector $\mathbf{w}$ at each time step (see Equation 9.3). The parameter update $\Delta\mathbf{w}$ is subsequently determined by weighting each noise term $\boldsymbol{\varepsilon}_t^w$, i.e.

$$\Delta\mathbf{w} = \mathbb{E}\left[\sum_{t=0}^{T} S(R_t, t) \boldsymbol{\varepsilon}_t^w \right],$$

where $R_t = \sum_{j=t}^T r_j$ is the future reward for time step t and the function $S(R_t, t)$ determines the weighting of $\boldsymbol{\varepsilon}_t^w$. The main idea is that noise terms $\boldsymbol{\varepsilon}_t^w$ which resulted in high returns R_t have a higher influence in the parameter update. Thus, the weighting typically depends on the point in time when the noise has been taken and on the return R_t after applying the noise. Even so the PoWER and the PI^2 are derived quite differently, they only differ in the way they calculate $S(R_t, t)$. We refer to the corresponding papers for an exact definition of $S(R_t, t)$. The main advantage of both algorithms is that they do not require a user-specified learning rate. The learning speed of both methods is comparable (Theodorou et al., 2010b), although PoWER has some restrictions on the reward function which can be used. Both methods are known to outperform the (step-based) policy gradient methods introduced in Section 9.2.1.

9.3 Episode-based Exploration Approaches

Episode-based exploring approaches apply the exploration noise to the parameter vector $\mathbf{w}$ before executing the whole roll-out, during the execution no further exploration is applied. In this setup we typically use a deterministic policy $\mathbf{a}_t = \pi(\mathbf{s}_t, t; \mathbf{w})$ and therefore, we only have system noise to deal with. As a result, the expected reward of a single parameter vector $\mathbf{w}$ is easier to estimate (with reduced variance, (Sehnke et al., 2010)).

The big advantage of episodic-exploring approaches is that they do not depend on a linear parametrization of the movement representation. In order to use episodic-exploring approaches we just have to be able to estimate $J(\mathbf{w})$ by performing roll-outs on the real system. In addition, sophisticated second order optimizers can be used in this setup which tend to show very good performance on many problems (Heidrich-Meisner and Igel, 2009b). For example, the CMA-ES algorithm estimates the full covariance matrix $\pm_w$ of a Gaussian distribution which is used for the exploration noise. This can be seen as second-order information of the cost-function. As a consequence, the exploration noise correlated and thus more directed than the uncorrelated exploration noise used by many other approaches. We will now briefly discuss the most relevant episode-based exploration approaches.

9.3.1 Gradient-based Methods

Similar to the standard step-based exploring PG methods, episode-based PG-methods try to estimate the gradient of the expected reward $J(\mathbf{w})$ in order to apply gradient descent. However, the gradient is now estimated by perturbing the parameter vector $\mathbf{w}$ before performing the roll-outs, during execution no additional exploration is applied (i.e. a deterministic policy is used).

9.3.1.1 Finite Difference Policy Gradients

The most simple policy gradient method is to use the Finite Difference Policy Gradient. Here, J rollouts [1] with small, random perturbations $\Delta\mathbf{w}_j$ of the current parameter vector $\mathbf{w}$ are performed on the real system. The gradient is calculated by using a first order Taylor-Approximation of the cost function and subsequently applying the least-square solution

$$\nabla_{\mathrm{FD}} J(\mathbf{w}) = (\Delta\mathbf{W}^T \Delta\mathbf{W})^{-1} \Delta\mathbf{W}^T \Delta\mathbf{J},$$

with $\Delta\mathbf{W} = [\Delta\mathbf{w}_1^T, \Delta\mathbf{w}_2^T \dots \Delta\mathbf{w}_J^T]^T$ and $\Delta\mathbf{J} = [J(\mathbf{w} + \Delta\mathbf{w}_1), J(\mathbf{w} + \Delta\mathbf{w}_2), \dots, J(\mathbf{w} + \Delta\mathbf{w}_J)] - J(\mathbf{w})$. The finite difference method has been used in (Kohl and Stone, 2003) as one of the first policy gradient methods applied to a real robot. It is also often used as baseline for comparison (Peters and Schaal, 2006), (Kober and Peters, 2010).

[1] typically J has to be twice the number of parameters to accurately estimate the gradient

9.3.1.2 The Policy Gradient with Parameter-Based Exploration algorithm

In difference to the Finite-Difference PG algorithm, the Policy Gradient with Parameter-Based Exploration (PG-PE) algorithm (Sehnke et al., 2010) calculates the likelihood gradient of the expected reward. It is therefore the equivalent to REINFORCE for the step-based-exploring approaches. However, the likelihood gradient is now calculated for the whole trajectory instead of calculating it for each time step separately. In order to do so, we introduce a distribution over the policy parameters $p(\mathbf{w}|\boldsymbol{\theta}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_w, \text{diag}(\sigma_i^2)$ and calculate the likelihood gradient with respect to $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$. The parameters $\boldsymbol{\theta}$ now define a distribution over the parameterspace $\mathbf{w} \in \mathcal{W}$. The expected reward is then given by

$$J(\boldsymbol{\theta}) = \int_{\mathbf{W}} \int_{\tau} p(\tau|\mathbf{w}) p(\mathbf{w}|\boldsymbol{\theta}) R(\tau) d\tau d\mathbf{w}$$

If we now apply again the 'log-ratio trick' and replace the integral with samples, the gradient of the expected reward $J(\boldsymbol{\theta})$ is given by

$$\nabla_{\text{PGPE}} J(\boldsymbol{\theta}) = \sum_j \nabla_{\boldsymbol{\theta}} \log p(\mathbf{w}_j|\boldsymbol{\theta}) R(\tau_j)$$

In the original algorithm (Sehnke et al., 2010), an uncorrelated Gaussian distribution is used as model for $\mathbf{w}$. Thus the exploration noise for each dimension of the parameter vector is adapted, but the correlations between the parameters are still neglected.

9.3.2 Stochastic Optimizers

By now we have only discussed algorithms which originated from Reinforcement Learning, but in fact any stochastic optimizer can be used to find the optimal parameters $\text{argmin}_{\mathbf{w}} J(\mathbf{w})$. These stochastic optimizers have shown impressive performance on many tasks so they should not be neglected. Here, we briefly present two promising approaches, the Covariance Matrix Adaption - Evolutionary Strategy (CMA-ES) [2] (Heidrich-Meisner and Igel, 2009b) and Cross-Entropy search (Mannor et al., 2003; de Boer et al., 2005). The strategy of both approaches is quite similar. They sample a certain number of offsprings $\mathbf{w}_j$ from a Gaussian distribution $\mathcal{N}(\mathbf{w}|\Updownarrow_k, \pm_k)$. The evaluated reward $J(\mathbf{w}_j)$ of the samples is used to weight each of the offsprings. Subsequently, the weighted offsprings are used to update the Gaussian distribution. CMA-ES and Cross-Entropy search only differ in the way they update the distribution, i.e. how they calculate $\Updownarrow_{k+1}$ and $\pm_{k+1}$ from the offsprings $\mathbf{w}_l$. One big advantage of both methods is that they estimate a full-covariance matrix $\pm_{k+1}$ for exploration, thus, both algorithms use correlated noise resulting in more directed exploration strategies than most other algorithms which simply use a diagonal covariance matrix. Although there is no direct comparison of CMA-ES and Cross-Entropy search, CMA-ES has become more popular in recent literature (Heidrich-Meisner and Igel, 2009b; Neumann, 2011).

[2] CMA-ES is often denoted as genetic algorithm, however, it can also be seen as stochastic second order optimizer.

9.4 Generalizing Temporal Representations to Multiple Situations

One disadvantage of temporal representations is that they are only valid locally, i.e. they can only be used for constant initial conditions of the robot (including a small neighborhood). For different initial conditions $\mathbf{s}_0'$, the parameter vector $\mathbf{w}$ has to be re-estimated. In order to avoid exhaustive relearning, we need to learn a hierarchic policy $\pi_{\mathbf{w}}(\mathbf{w}|\mathbf{s}_0;\boldsymbol{\theta})$ which can generalize between different initial situations $\mathbf{s}_0$ of the robot. We will denote the parameter vector of the hierarchic policy as $\boldsymbol{\theta}$. In order to assess the expected reward for a parameter vector $\boldsymbol{\theta}$, we now have to evaluate multiple roll-outs starting from different initial conditions, and thus, solving this problem directly with standard parameter-exploring policy search methods is highly inefficient. For a discussion of more efficient methods and also a new method which has been introduced in our work in the paper (Neumann, 2011) we refer to Chapter 10.

Chapter 10

Variational Inference for Policy Search in Changing Situations

Contents

Many policy search algorithms minimize the Kullback-Leibler (KL) divergence to a certain target distribution in order to fit their policy. The commonly used KL-divergence forces the resulting policy to be 'reward-attracted'. The policy tries to reproduce all positively rewarded experience while negative experience is neglected. However, the KL-divergence is not symmetric and we can also minimize the the reversed KL-divergence, which is typically used in variational inference. The policy now becomes 'cost-averse'. It tries to avoid reproducing any negatively-rewarded experience while maximizing exploration.

Due to this 'cost-averseness' of the policy, Variational Inference for Policy Search (VIP) has several interesting properties. It requires no kernel-bandwith nor exploration rate, such settings are determined automatically by the inference. The algorithm meets the performance of state-of-the-art methods while being applicable to simultaneously learning in multiple situations.

We concentrate on using VIP for policy search in robotics. We apply our algorithm to learn dynamic counterbalancing of different kinds of pushes with human-like 2-link and 4-link robots.

10.1 Introduction

Variational inference is a widely used approximate inference method. While there exists first applications of variational inference for discrete reinforcement learning (Furmston and Barber, 2010), it has never been used for policy search in high dimensional parameter spaces. Variational inference introduces an approximate distribution q and iteratively minimizes the Kullback-Leibler divergence $\mathrm{KL}(q||p)$ between

q and the target distribution p. This minimization is also known as I(nformation)-projection of distribution p.

In policy search, many algorithms also apply approximate inference. However, all these algorithm use the M(oment)-projection, which is given by the reversed KL-divergence $\mathrm{KL}(p||q)$ to estimate their policy. While at the first glance this might only be a minor difference, it turns out that the resulting policies may differ considerably. Policies calculated by the M-projection try to reproduce all experience with high reward, but neglect information coming from negative experience. We will therefore call these policies 'reward-attracted'. The I-projection forces the resulting policy to be 'cost-averse'. Here, the focus of the policy is to avoid reproducing negative experience, while exploration is maximized.

Which projection is better suited for policy search? We argue for the I-projection. When using a common Gaussian policy, the M-projection averages over all positively rewarded experience seen so far. However, in the case of a multi-modal or non-concave target distribution taking the average might be a bad choice. The I-projection always tries to exclude negative experience from the resulting distribution, and thus, concentrates at one mode of the target distribution. Non-concave target distributions typically occur if we want to apply policy search for multiple situations. The I-projection can be applied with ease in this context. The 'cost-averseness' also comes with additional advantages. The algorithm automatically determines the optimal kernel bandwidth for a new situation and adapts its exploration rate and used search directions.

In difference to the M-projection, the I-projection can't be minimized in closed form. We have to rely on non-linear optimization methods like gradient descent. Here, we present a new method where gradient descent is performed on meta-parameters of the approximate distribution q.

We will apply our new Variational Inference for Policy Search (VIP) algorithm to learn complex motor skills with robots. In robotics we often need to search for parametrized movement plans in related, but different scenarios. These movement plans, also called Dynamic Movement Primitives (Ijspeert and Schaal, 2003; Schaal et al., 2007), Motion Templates (Neumann et al., 2009) or Muscle Synergies (Bizzi et al., 2008) are often only valid locally, and hence, need to be adjusted for a new situation.

For example, a tennis playing robot has to adapt its movement to the trajectory of the ball or a humanoid robot has to react differently to counter-balance different kinds of pushes. Hence, we need to find a policy $\pi(\mathbf{w}|\mathbf{s}^0)$ which is able to choose good parametric descriptions $\mathbf{w} \in \mathcal{W}$ of the movement plan given the initial conditions $\mathbf{s}^0$. Learning such a policy π is very challenging due to the high-dimensionality of parameter-space $\mathcal{W}$. Our algorithm is well suited for such tasks.

Many policy search algorithm like the CMA-ES (Heidrich-Meisner and Igel, 2009b), Cross-Entropy search (Mannor et al., 2003) or the PoWER (Kober and Peters, 2010) algorithm are limited to the single-situation setting. Only few algorithms exist for learning in multiple initial conditions. Here, we can use Reward-Weighted Regression (RWR)(Kober et al., 2010) or Cost-Regularized-Kernel Regression (CRKR) (Kober et al., 2010), which is the kernelized version of RWR. Both algorithms use locally weighted linear regression methods to interpolate between different initial states $\mathbf{s}_i^0$. In addition to the local weighting, the data points are weighted by their corresponding rewards. The reward-weighted linear regression represents an M-projection of the reward distribution (see Section 10.3.1), therefore these algorithms suffer from the previously mentioned limitations of the M-projection.

Both algorithms require that the user specifies the shape or bandwidth of the receptive fields or kernels. This shape is not only kept constant during the learning phase, it is also constant in the whole state space. Therefore the user always has to make a tradeoff between fast learning speed and good quality of the final performance. Because the I-projection always wants to exclude samples with low reward, the 'kernel shape' automatically adapts to the data density as well as to the shape of the target distribution.

Note that CRKR and RWR have only been used to learn meta-parameters of the motion (Kober et al., 2010) (like the duration or the end-point of the motion). The remaining (typically higher-dimensional) parametrization for the shape of the trajectory was kept fixed. Therefore, the application is limited to similar shapes of the movement. The VIP approach allows learning with the full-parametric representation of a movement for multiple scenarios, and therefore, can find completely different movements for different subregions of the state space.

We will apply our method to a 2-link and a 4-link dynamic robot balancing task where the robot has to counterbalance different kinds of pushes.

10.2 Kullback Leibler (KL) Divergences

We quickly review concept of KL-divergences because it is of great importance for this paper. The KL divergence between two probability distributions q and p is defined as

$$\mathrm{KL}(q||p) = -\int_{\mathbf{X}} q(\mathbf{X}) \log \frac{p(\mathbf{X})}{q(\mathbf{X})} d\mathbf{X}$$

It is zero if and only if the two distributions are equal. Since the KL-divergence is *not* symmetric, there are 2 kinds of KL-divergences which we can minimize in order to approximate a target distribution p with an approximate distribution q.

- The **M-projection** $q = \mathrm{argmin}_q \mathrm{KL}(p||q)$: The M-projection forces the approximate distribution q to have high probability everywhere where p has high probability. Therefore, if distribution q is a Gaussian, the M-projection tries to average over all modes of p.

- The **I-projection** $q = \mathrm{argmin}_q \mathrm{KL}(q||p)$: It forces the approximate distribution q to be zero everywhere where p is zero. Can not be calculated in closed form for the most distributions. When using a Gaussian distribution q, the I-projection typically concentrates on a single mode of the target distribution.

These differences between the projections are well known (Bishop, 2006), however, the effect of these difference for policy search have never been evaluated.

10.3 Inference for policy search

Many policy search algorithms (Kober and Peters, 2010; Vlassis et al., 2009; Heidrich-Meisner and Igel, 2009b) use inference or inference related methods to iteratively optimize the policy.

In order to use inference for policy search we define a binary reward event $R = 1$ as observed variable. To simplify notation we will always write R when we mean $R = 1$. The probability of this reward event is given by $p(R|\tau) \propto \exp(-C(\tau))$, where τ is a trajectory and $C(\tau)$ are the associated costs. This is a common method to transform an optimization problem into an inference problem (Toussaint, 2009). We want to find parameter vectors $\boldsymbol{\theta}$ with high evidence

$$p(R; \boldsymbol{\theta}) = \int_\tau p(R|\tau) p(\tau; \boldsymbol{\theta}) d\tau,$$

where τ is a trajectory and $p(\tau; \boldsymbol{\theta})$ is the parametric model of the trajectory distribution. The policy π is contained in this model.

We can now introduce a variational distribution $q(\tau)$ which is used to decompose the log-evidence

$$\log p(R; \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q||p_R), \tag{10.1}$$

where

$$\mathcal{L}(q, \boldsymbol{\theta}) = \int_\tau q(\tau) \log \frac{p(R|\tau) p(\tau; \boldsymbol{\theta})}{q(\tau)} d\tau$$

is the lower bound of the log evidence and

$$\mathrm{KL}(q||p_R) = -\int_\tau q(\tau) \log \frac{p(\tau|R; \boldsymbol{\theta})}{q(\tau)} d\tau \tag{10.2}$$

is the KL-divergence between the q and the *reward-weighted* trajectory distribution

$$p_R(\tau) = p(\tau|R; \boldsymbol{\theta}) = \frac{p(R|\tau) p(\tau; \boldsymbol{\theta})}{p(R; \boldsymbol{\theta})} \tag{10.3}$$

The correctness of Equation (10.1) can be easily verified by substituting Equation (10.3) into Equation (10.2). Note that this decomposition is the same as used in expectation-maximization (EM) and variational inference algorithms. It has also already been used in (Furmston and Barber, 2010) for using variational inference for learning the model of discrete MDPs.

The lower bound $\mathcal{L}(q, \boldsymbol{\theta})$ is now iteratively improved by an expectation (E-) and a maximization (M-) step. In the E-step, we minimize $\mathrm{KL}(q||p_R)$ with respect to q. Since $\log p(R; \boldsymbol{\theta})$ is fixed, the lower bound has to increase. In the M-step we maximize the lower bound $\mathcal{L}(q, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

10.3.1 M-Projection: Monte-Carlo EM-based Policy Search Algorithms

Monte-Carlo (MC) EM-based algorithms (Kober and Peters, 2010; Vlassis et al., 2009) use a sample based approximation for q, i.e. in the E-step they minimize the KL-divergence $\mathrm{KL}(q||p_R)$ by setting $q(i) \propto p(R|\tau_i)p(\tau_i;\boldsymbol{\theta})$ for a discrete set of samples τ_i. Subsequently, the $q(i)$ are used to replace the integral in the lower bound $\mathcal{L}(q,\boldsymbol{\theta})$ by a sum. The lower bound therefore reads

$$\begin{aligned}\mathcal{L}(q,\boldsymbol{\theta}_{\text{new}}) &= \sum_{\tau_i} p(R|\tau_i)p(\tau_i;\boldsymbol{\theta}_{\text{old}})\log\frac{p(\tau_i;\boldsymbol{\theta}_{\text{new}})}{p(\tau_i;\boldsymbol{\theta}_{\text{old}})} \\ &= -\mathrm{KL}(p_R(\tau)||p(\tau;\boldsymbol{\theta}_{\text{new}})) + \text{const}\end{aligned}$$

As we can see maximizing the lower bound with respect to the new parameter vector $\boldsymbol{\theta}_{\text{new}}$ is equivalent to calculating the *M-projection* of $p_R(\tau)$. Note that this is exactly the same lower bound as given in (Kober and Peters, 2010) for the PoWER and RWR algorithm. Thus, these algorithms are special cases of the decomposition shown in Equation 10.1.

10.3.2 I-projection: Variational Inference for Policy Search

In the variational approach, a parametric representation of q is used instead of a sample-based approximation. We choose $q(\tau;\boldsymbol{\omega})$ to be from the same family of distributions as $p(\tau;\boldsymbol{\theta})$. Now, we will use a sample-based approximation to replace the integral in the KL-divergence $\mathrm{KL}(q||p_R)$ needed for the E-step. Thus we need to minimize

$$\mathrm{KL}(q||p_R) = -\sum_{\tau_i} q(\tau_i;\boldsymbol{\omega})/Z_q \log\frac{p_R(\tau_i)/Z_p}{q(\tau_i;\boldsymbol{\omega})/Z_q}, \tag{10.4}$$

with respect to $\boldsymbol{\omega}$, which is equivalent to the *I-projection* of $p_R(\tau)$. The terms Z_q and Z_p are used to normalize the sample-based approximations. The M-step now trivially reduces to setting the new parameter vector $\boldsymbol{\theta}_{\text{new}}$ to $\boldsymbol{\omega}$.

Both algorithms only differ in the used projections of $p_R(\tau)$. As the projections are in general different, they converge to a different (local) maximum of the lower bound $\mathcal{L}(q,\boldsymbol{\theta})$. When using a Gaussian model distribution, the I-projection concentrates on a single mode. This is not a problem if all modes are almost equally good, however, the I-projection might also choose a sub-optimal mode (which has lower reward probability). In our evaluations we could not observe this problem. The M-projection always averages over all modes and therefore might also include large areas of low reward in the distribution. Hence, we consider the use of the I-projection to be less harmful. If the target distribution is concave, both projections yield almost the same solutions, however, using the I-projection is computationally more demanding.

The discussed projections are applicable for any kind of policy search problems, however, in this paper we will focus on single-step decision problems with high dimensional action spaces because these problems are of high importance for motor skill learning with motion primitives.

10.4 Policy Search in multiple situations

In this paper we concentrate on policy search in multiple situations. Thus, we want to learn a policy $\pi(\mathbf{w}|\mathbf{s}^0;\boldsymbol{\theta})$ for choosing the parametric description $\mathbf{w}$ of our movement plan when being in situation $\mathbf{s}^0$.

We will treat the policy search problem as 1-step reinforcement learning problem and neglect any sequential nature of the decision problem. The agent chooses its desired trajectory description $\mathbf{w}$ in the initial state $\mathbf{s}^0$ and then observes the whole trajectory τ and the associated costs $C(\tau)$ as one big step. The trajectory τ_i itself is therefore determined by the state-action pair $\langle \mathbf{s}_i^0, \mathbf{w}_i \rangle$ and its associated costs $C(i)$. All derivations from Section 10.3 are still valid, we just replace the trajectories τ_i with the state-action pairs $\langle \mathbf{s}_i^0, \mathbf{w}_i \rangle$.

As samples we will always use the whole history of the agent, i.e. we will use samples from all situations $\mathbf{s}_i^0$ experienced so far. For the sake of simplicity, we neglected any importance weights in Equation 10.4 which should be used to compensate for the fact that the history of the agent is usually not sampled uniformly from the state-action space. In the subsequent discussion we assume that each dimension of the parameter vector has been scaled to the interval $[0;1]$.

If the reward weighted probability $p_R(\tau_i)$ is very close to 0 we can't use the log function. Instead, we use a penalty term of $-P_z$ for $\log p_R(\tau_i)$. It turned out that reasonable settings of this value have to scale exponentially with the number of dimensions of the parameter space to account for the increasing volume of the search space.

10.4.1 Approximate Distribution

For representing $p(\mathbf{s}_i^0, \mathbf{w}_i;\boldsymbol{\theta})$ we use Gaussian distributions $\mathcal{N}([\mathbf{s}^0;\mathbf{w}]|\boldsymbol{\mu},\boldsymbol{\Sigma})$. Since a Gaussian is a rather simple representation we re-estimate the Gaussian for a new, currently active situation $\mathbf{s}_t^0$. For every re-estimation, the state components of $\boldsymbol{\mu}$ are clamped at $\mathbf{s}_t^0$ by putting a sharply peaked prior on these components (see next section).

In Figure 10.1 and 10.2, we illustrated the difference of policy search with the M- and the I-projection for bimodal and non-concave target distributions. For the bi-modal distribution, the M-projection concentrates on both modes while the I-projection only tries to cover one mode. For the non-concave target distribution we assumed that the first variable represents a state variable which is observed. Therefore, we clamped this dimension of the mean of the Gaussian to be the observed value. Again, the M-projection tries to average over the non-concave function, and hence also includes regions of low reward, while the I-projection nicely approximates the desired distribution.

10.4.2 Minimization of the I-projection

The I-projection $\mathrm{KL}(q||p_R)$ is difficult to use because it can't be calculated in closed form. We have to rely on non-linear optimization methods, i.e. gradient descent. However, optimizing directly the parameters of a Gaussian is difficult because of the quadratic number of parameters needed to represent the covariance matrix.

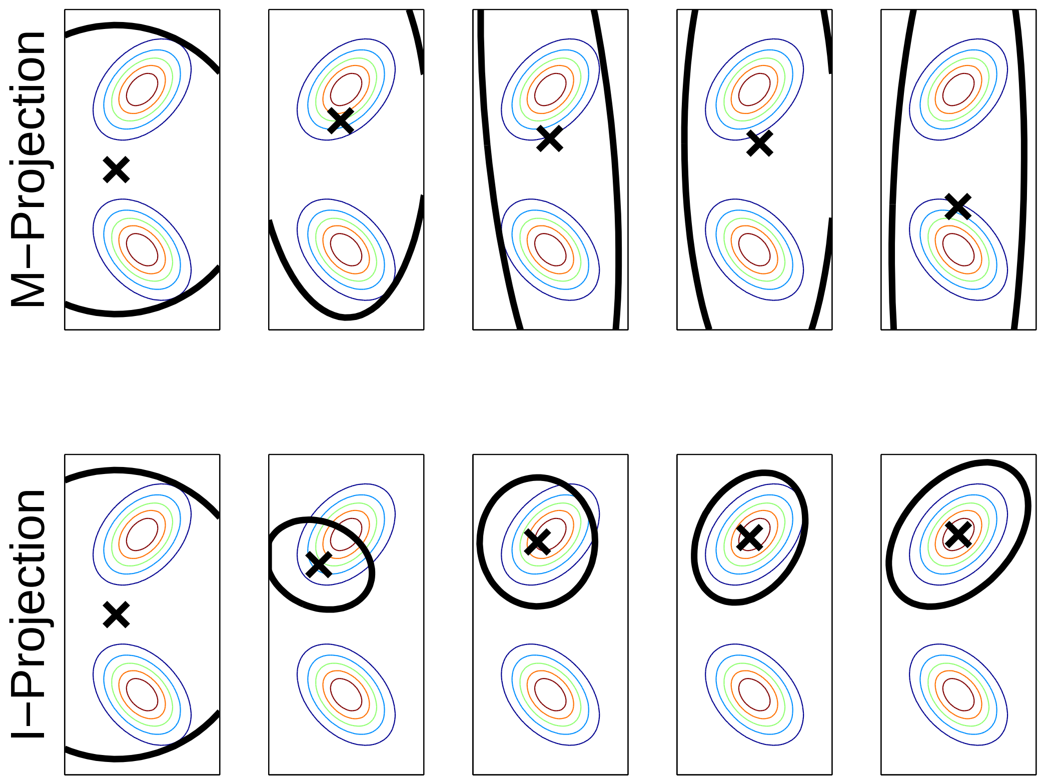


Figure 10.1: Comparison of the Variational Policy Search algorithm using the I-projection against the M-projection on a bi-modal reward function. A simple Gaussian distribution was used as model distribution. The M-projection tries to average over both modes, while the I-projection concentrates on a single mode.

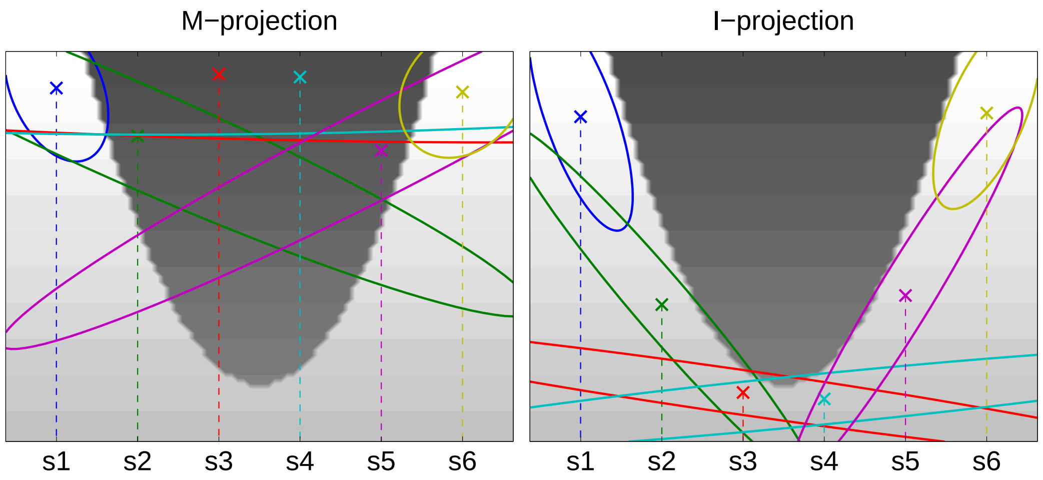
Hence, we propose a sample oriented approach which is computationally more tractable. For each sample we introduce a weighting v_i. These weightings are used to calculate the weighted maximum likelihood (ML) estimate from the data-points. We will denote the weighted sample mean as $\mathbf{m}$ and the weighted sample covariance matrix as $\mathbf{S}$. The weights v_i are normalized such that $\max_i v_i = 1$.

In order to clamp the state-space part of the mean $\boldsymbol{\mu}$ at the current initial state $\mathbf{s}_t^0$, we combine the ML-estimate $\mathbf{m}$ with a Gaussian prior distribution $P(\boldsymbol{\mu}|\mathbf{s}_t^0) = \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{\mu}_0, \boldsymbol{S}_0)$ with $\boldsymbol{\mu}_0 = \left[\mathbf{s}_t^0, \mathbf{0.5}\right]^T$ and $\boldsymbol{S}_0$ is a diagonal matrix which is set such that the prior is sharply peaked for the state variables $\mathbf{s}^0$ and almost flat in the action space. The mean $\boldsymbol{\mu}$ of our Gaussian distribution is then given by

$$\boldsymbol{\mu} = (\boldsymbol{S}_0^{-1} + \mathbf{S}^{-1})^{-1}(\boldsymbol{S}_0^{-1}\boldsymbol{\mu}_0 + \mathbf{S}^{-1}\mathbf{m})$$

For the covariance matrix $\boldsymbol{\Sigma}$ of our model, we also use a combination of a prior covariance matrix $\boldsymbol{C}_0$ and the weighted sample covariance S.

$$\boldsymbol{\Sigma} = \frac{\sum_i v_i \mathbf{S} + \alpha \boldsymbol{C}_0}{\sum_i v_i + \alpha}, \quad \boldsymbol{C}_0 = k \cdot \operatorname{diag}([\sigma_i^2]) + \textstyle\sum_{j=1}^{l-1} c_j \boldsymbol{\Sigma}_{\boldsymbol{j}},$$



(a) M-projection (b) I-projection

Figure 10.2: Comparison of I-projection and M-projection on a non-concave reward function. Dark background indicates negative reward. The model distribution is a Gaussian of which the mean (indicated by 'x') of the state variable (x-axis) has been clamped at different locations. The M-projection again tries to average over the non-concave function while the I-projection nicely approximates the desired policy.

where $\boldsymbol{\Sigma_j}$ are the covariance matrices of the previous iterations of VIP. The $\boldsymbol{\Sigma_j}$ are used to incorporate previous search directions into the current search. The parameters k, $\sigma^2_{1:d}$ and $c_{1:l-1}$ are also optimized by gradient descent.

After calculating $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ we can evaluate the KL-divergence $\mathrm{KL}(q||p_R)$ on our sample points by the use of Equation 10.4. The gradient with respect to v_i, α, k, σ_i^2 and c_j is calculated numerically by finite differences. Subsequently we apply standard gradient descent augmented by a line search algorithm to estimate the optimal learning rate. The algorithm always runs for 10 iterations.

We also use a slight modification of the original variational algorithm. Instead of using the model distribution $p(\mathbf{s}_i^0, \mathbf{w}_i; \boldsymbol{\theta})$ for calculating the reward weighted trajectory distribution $p_R(i)$ we use the sample weights v_i found by the previous KL-divergence minimization, i.e $p_R(i) = v_i p(R|\mathbf{s}_i^0, \mathbf{w}_i)$. This turned out to be numerically more stable in high dimensional parameter spaces.

10.4.3 Reward Transformation

Instead of using the standard reward transformation $p(R|\mathbf{s}_i^0, \mathbf{w}_i) = \exp(-C(\mathbf{s}_i^0, \mathbf{w}_i))$, we will use a baseline $V(\mathbf{s}_i^0)$ and also introduce a scaling factor ρ to the costs, i.e. $p(R|\mathbf{s}_i^0, \mathbf{w}_i) = \exp\left(-(C(\mathbf{s}_i^0, \mathbf{w}_i) - V(\mathbf{s}_i^0))/\rho\right)$. Both mechanisms help to improve accuracy of the algorithm as well as to reduce the number of required iterations.

As baseline we use an estimate of the value $V(\mathbf{s}_i^0) = \int_{\mathbf{w}} C(\mathbf{s}_i^0, \mathbf{w}) p(\mathbf{w}|\mathbf{s}_i^0; \boldsymbol{\theta}) d\tau$ at state $\mathbf{s}_i^0$. In order to do so, we use the tuples $\langle \mathbf{s}_i^0, C_i \rangle$ as data points to estimate a Gaussian cost model. Each data point gets again weighted by the weights v_i found by the previous KL-minimization. Subsequently, we condition this Gaussian cost model on the scenario states $\mathbf{s}_i^0$ of our samples. This results in a linear Gaussian

Algorithm 3: Variational Policy Search

```
Input: History of the agent $H = \langle \mathbf{s}_i^0, \mathbf{w}_i, C_i \rangle$, current scenario $\mathbf{s}^0$, initial
        covariance $\mathbf{\Sigma}_0$
Initialize $\boldsymbol{\mu}_0 = [\mathbf{s}_0; \mathbf{1}/2]$ and $v_i = \mathcal{N}([\mathbf{s}_i^0; \mathbf{w}_i] | \boldsymbol{\mu}_0, \mathbf{\Sigma}_0)$ for all $i$
for $l = 1$ to $L$ do
    Estimate $V(\mathbf{s}_i^0)$ and $\rho$ by calculating a Gaussian cost model using $v_i$.
    Calculate cost weighted trajectory distribution
      $p_R(i) = v_i \exp\left(-\left(C_i - V(\mathbf{s}_i^0)\right)/\rho\right)$
    Check effective number of examples, eventually reduce sharpness of $p_R$
    while $\sum_i p_R(i) / \max_j p_R(j) < n_{act}$ do
        $p_R(i) = p_R(i)^{0.9}$ for all $i$
    Acquire new $v_i$, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ (minimize $\mathrm{KL}(q||p_R)$)
      $[v_i, \boldsymbol{\mu}_l, \mathbf{\Sigma}_l] = \text{I-project}(p_R, H, \{\mathbf{\Sigma}_0, \cdots \mathbf{\Sigma}_{l-1}\})$
    Set new model distribution...
      $p(\mathbf{s}^0, \mathbf{w}; \boldsymbol{\theta}) = \mathcal{N}([\mathbf{s}^0; \mathbf{w}] | \boldsymbol{\mu}_l, \mathbf{\Sigma}_l)$
Calculate policy (conditional Gaussian)
$\pi(\mathbf{w}|\mathbf{s}^0; \boldsymbol{\theta}) = p(\mathbf{s}^0, \mathbf{w}; \boldsymbol{\theta}) / p(\mathbf{s}^0; \boldsymbol{\theta})$
```

model from which we use the (state-dependent) mean as baseline $V(\mathbf{s}_i^0)$.

The scaling factor ρ regulates the greediness of our distribution $p(R|\mathbf{s}^0, \mathbf{w})$. We use the standard deviation of the conditioned Gaussian cost model to determine ρ.

If the effective number of activations of our target distribution $p_R(i)$ gets too small (i.e. $\sum_i p_R(i) / \max_j p_R(j) < n_{\text{act}}$) we do not have enough data-points to reliably estimate the Gaussian models. Hence, we iteratively reduce the sharpness of $p_R(i)$ by setting all $p_R(i)$ to $p_R(i)^{0.9}$ until the effective number of samples is larger than n_{act}. The parameter n_{act} has to be specified by the user and depends on the dimensionality of the state-space (in our experiments we varied the value between 5 and 15).

10.4.4 Estimating the policy

So far we have estimated a model which describes the probability of whole trajectories, i.e. in our case a probability distribution over the state *and* action space. In order to determine the policy $\pi(\mathbf{w}|\mathbf{s}_t^0; \boldsymbol{\theta})$ we just have to condition on the current state $\mathbf{s}_t^0$. This is again a linear Gaussian model which can be easily calculated.

The whole algorithm is summarized in Algorithm 3. The number of iterations L was always set to 10. For performance reasons we only use the last N examples (between 100 and 10000) from the history. The initial covariance $\mathbf{\Sigma}_0$ as given in Algorithm 3 is typically almost flat in the action space and state space. The method is almost invariant to this setting.

In difference to MC-EM based algorithms like RWR or CRKR we use several iterations to estimate the model distribution. Additionally, the introduced scaling factor ρ of the reward function helps to set the greediness of the resulting distribution correctly. If we would use the M-projection and only apply one iteration ($L = 1$) without the scaling factor ρ and the baseline $V(\mathbf{s}_i^0)$, VIP reduces to RWR.

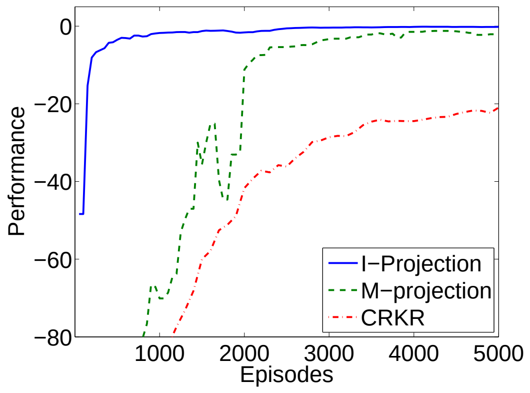


Figure 10.3: Evaluation of VIP on Cannon-Toy task. We compared our algorithm using the I and the M-projection. The I-projection converges much faster and also produces a final policy with higher quality. The competing algorithm CRKR could not find as good solutions.

10.5 Experiments

In our evaluations of the algorithms we always use the median over 20 trials. The median is used to get rid of outliers, 1 or 2 trials out of 20 usually did not find good solutions.

We first evaluate our algorithm on a Cannon Toy Task. Here, the task is to hit a target located at distance d with a cannon ball. The controls are the launching angle α and the launching velocity v of the cannon ball. The angle was restricted to $[0; \pi/2]$ and the velocity to $[0; 10] m/s$. The cannon-ball was modelled as 1-kg point mass, gravity and a horizontal wind force f act on the ball. The wind force f can be in the range of $[0; 1]$ and the target locations were also restricted to $[0; 10]$. This results in a 2-dimensional state space $\mathbf{s}^0 = [d, w]$ and a two dimensional parameterspace $\mathbf{w} = [\alpha, v]$. As reward function we used 20 times the negative squared distance of the impact position to the target. Note there are several solutions to hit a target at a certain distance, rendering the reward function multi-modal. We compared our algorithm using the I-projection and M-projection against the CRKR algorithm. Every 50 episodes we evaluated the policy at 20 randomly chosen states (which were fixed for every evaluation). The parameter n_{act} was set to 5 and P_z to 10.

The result can be seen in Figure 10.3. The I-projection clearly outperformed the M-projection in learning speed as well as in the quality of the learned policy. The final average distance to the target was $0.08m$ with the I-projection while the final policy of the M-projection missed the target at a average distance of $0.26m$. The learning speed of CRKR matched the speed of the M-projection, but could not find as good solutions. We also compared both approaches with the finite difference policy gradient algorithm using a fixed set of basis functions (Kober et al., 2010) in the state space. The algorithm did converge after approximately 10^5 episodes, which is not shown in Figure 10.3 due to the bad performance.

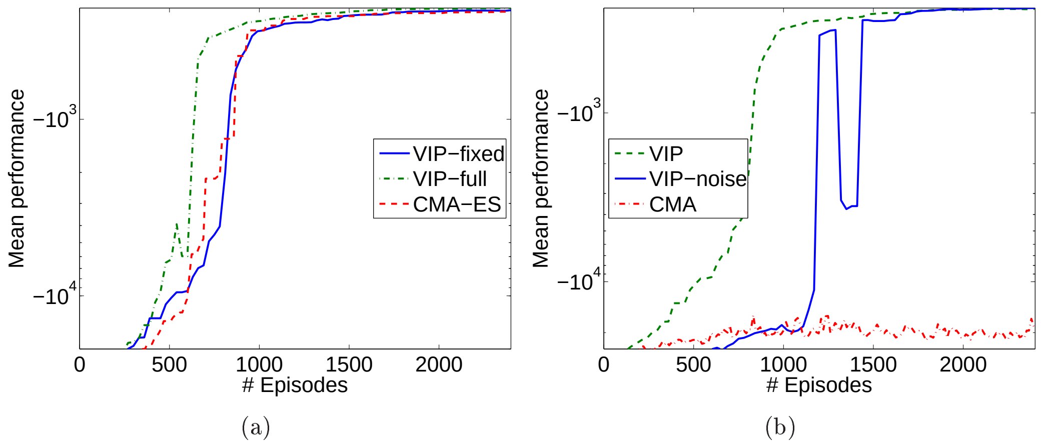


Figure 10.4: (a)Comparison of the VIP with full representation of the covariance (VIP-full) and the fixed representation (VIP-diag) with the CMA-ES algorithm. In order to compare our algorithm to CMA-ES, we only used a single force $F = 25$Ns. We use the maximum value seen so far for the plot of CMA-ES. (b) In this experiment we added a uniformly distributed noise $\varepsilon \in [-2.5; 0]$ to the force F. As the CMA-ES is unaware of this noise it could not cope with this setting. VIP was only sightly disturbed and could find solutions of the same quality as in (a).

10.5.1 2 and 4 Link Humanoid Balancing

Here we use a 2-link and a 4-link model to learn dynamic humanoid balancing strategies. The masses and lengths of the links as well as the maximum torques were chosen to crudely match a human.

The joints of the 2-link model resemble the ankle and the hip joints. For a more exact description of the model please refer to (Atkeson and Stephens, 2007). The robot is pushed with a certain force $0 \le F \le 25$Ns which results in an immediate jump in the joint velocities. The robot has to learn to keep balance. This requires completely different strategies for different forces (Atkeson and Stephens, 2007). If the joints leave the intervals $\phi_1 \in [-0.4; 0.8]$ or $\phi_2 \in [-0.1; 1.6]$ the robot has fallen and the episode is terminated. An episode is considered as successful if the robot has managed to keep balance for $5s$. The state space is defined by the applied (one dimensional) force F. We used the following reward function

$$C(\tau) = -2000(T-5)^2 - 0.01 \sum_{t=1}^{T} \mathbf{a}_t^T \mathbf{a}_t,$$

where T is the point in time the robot falls over (or 5s if the robot keeps balance).

The whole movement representation consisted of 19 parameters. Since the exact representation of the movement is of minor importance for this paper we refer to the supplementary material for further information. For performance reasons, we always create 30 samples from the currently estimated policy. The parameter n_{act} was again set to 5 and the punishment for including samples with zero probabilities to $P_z = 300$ In our first experiment we compared our algorithm to CMA-ES, which is a highly competitive stochastic optimizer, in a single situation setup with $F =$

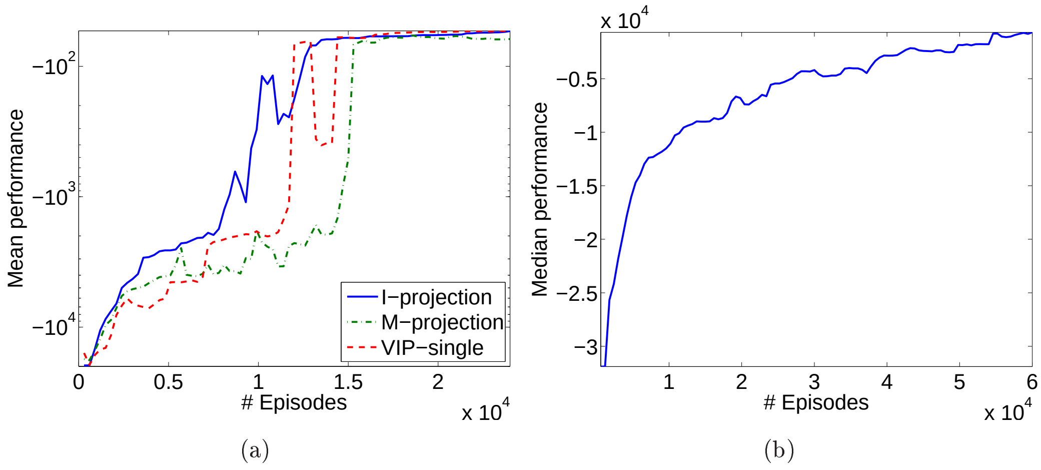


Figure 10.5: (a)Comparison of the I-projection and M-projection on the multi-force setup. VIP-single denotes learning for each force separately. The I-projection could outperform the M-projection and also slightly the VIP-single setup. (b) Learning curve of the 4-link balancing experiment with random forces.

25Ns. We evaluated VIP one time with learning all diagonal entries σ_i^2 of the covariance matrix and VIP when keeping these factors fixed. As we can see in Figure 10.4(a), VIP with the full representation performed best. VIP with the fixed diagonal entries showed similar performance as the CMA-ES algorithm. Because of the huge computational requirements of the full representation (one trial runs for $10h$) we will only use the fixed diagonal representation (one trial runs for 90min) for the remaining experiments. In the next experiment (Figure 10.4(b)) we used a small noise for our force F which was uniformly sampled from interval $[-2.5; 0]$Ns. This noise was known to the VIP, however, as CMA-ES is inherently unaware of the state $\mathbf{s}^0$, it could not learn a useful policy. The VIP algorithm was only slightly affected by the noise. The final performance was similar as learning without noise.

Next, we evaluated the VIP algorithm once with the M-projection and the I-projection on the multi-force setup. The force was chosen uniformly from the interval $[0, 25]$Ns. We also compared our algorithm to the noisy single situation setting. Here, we used 10 different forces from 2.5 to 25Ns and performed individual learning trials for each force (we again added a noise of $[-1.25, 1.25]$ to the force). The result can be seen in Figure 10.5, again, the I-projection outperformed the M-projection, however, the difference was not that extreme as in the Cannon task. Still, the final performance of the I-projection (-51.2) was better than the M-projection (-62.1) by 20%. We can also see that learning with all forces at once could slightly improve the learning speed in comparison to the average of the noisy single-force setup.

The 4-link model consisted of an ankle, a knee, a hip and a shoulder joint. In this experiment the force $\mathbf{F}$ was a 4-dimensional vector, denoting the force value applied to each body part. Thus, our state space is 4 dimensional. The movement representation for this task had 39 parameters. We always normalized the force vector $\mathbf{F}$, such that $|\mathbf{F}| = 25$Ns. In this experiment we used 16 randomly chosen force vectors, which were additionally perturbed by a uniformly sampled noise in

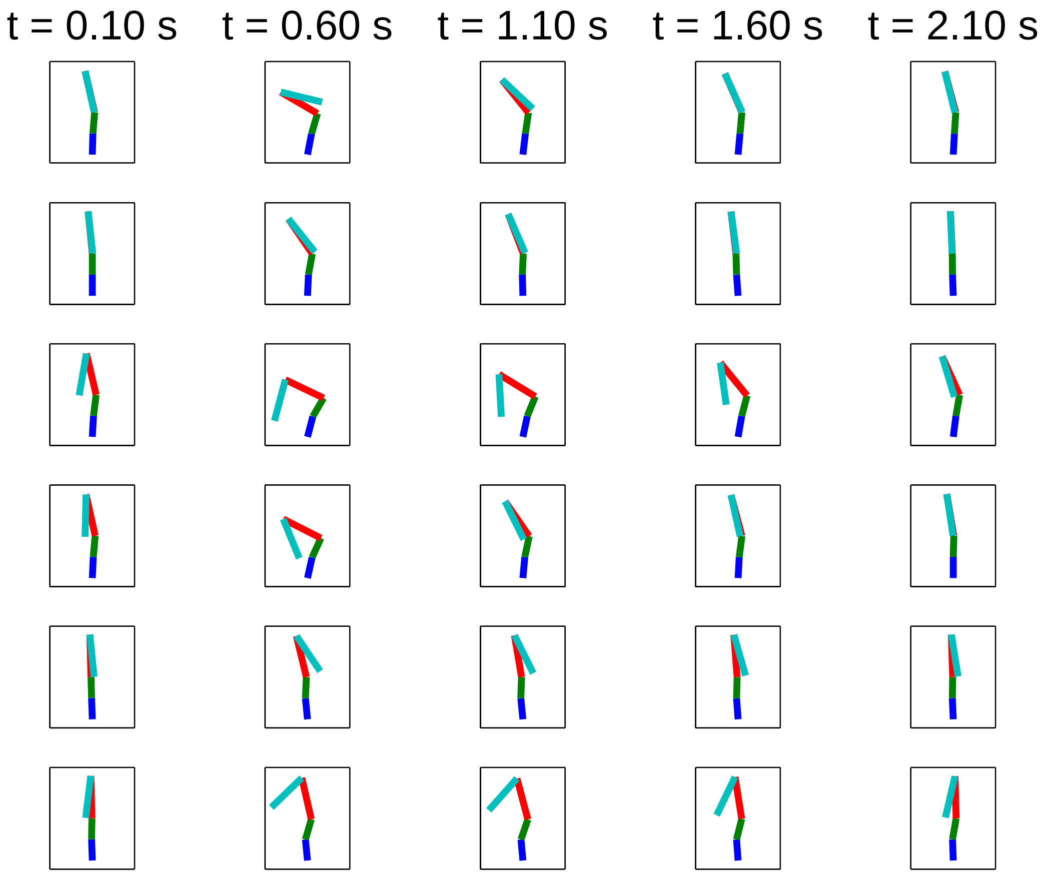


Figure 10.6: Learned balancing strategies for different random forces (with $|\mathbf{F}| = 25$Ns). The robot has learned to apply completely different strategies in different situations.

the interval ± 2.5Ns. The parameter n_{act} was set to 10 and P_z to 10^5. The learning curve for this experiment can be seen in Figure 10.5(b). After 60000 episodes the agent was able to balance almost all experienced forces. The resulting balancing strategies for different forces can be seen in Figure 10.6. As we can see, the robot has learned to apply completely different strategies in different situations.

10.6 Conclusion and future work

Existing policy search algorithms typically approximate the policy by using the M-projection to the reward-weighted trajectory distribution. In this paper we proposed to use the I-projection of the reward-weighted trajectory distribution as interesting alternative. The I-projection alleviates many problems connected to the M-projection. While the I-projection is computationally a much more difficult operation, the 'cost-averse' policy resulting from the I-projection comes along with several advantages. Because the I-projection always wants to exclude negative examples, the algorithm does not suffer from problems which occur by averaging over non-concave or multi-modal target distributions. Consequently, it shows an increased learning speed, improved performance of the final policy and it can also be applied with ease to the learning in multiple situations simultaneously.

The main restriction of VIP is the computation time. In future, we plan to use mixture of Gaussian models to alleviate this problem. This should give us considerable speed up because we do not have to re-estimate our distributions over and over again. Furthermore, a more efficient method for calculating the I-projection is needed.

VIP is not limited to the single step reinforcement learning setup. In the future we plan to use the algorithm also for sequential decision tasks. In this case, message passing algorithms like the one presented in (Toussaint, 2009) could extend our framework.

10.7 Acknowledgments

This chapter is based on the paper 'Variational Inference for Policy Search in Changing Situations' published at the International Conference for Machine Learning (ICML) 2011. Gerhard Neumann was the only author of this paper.

Appendix

Motion Templates used for balancing

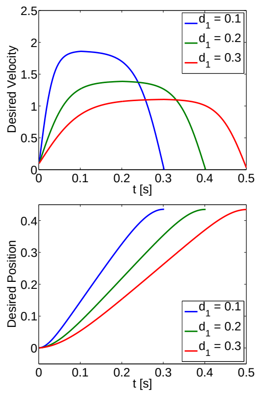
The motion templates (or movement plans) used for the balancing tasks define desired velocity profiles. The profiles are intergrated to get a desired trajectory. The policy of the motion templates is then defined by a linear PD-trajectory tracking controller. The motion is divided into 2 motion templates. Template m_1 drives the robot to a set-point of each joint, Template m_2 tries to stabilize the agent at the upright position. Template m_1 consists of an acceleration phase and a deceleration phase. The acceleration phase takes d_1 seconds and has the following velocity profile :

$$v_D(t) = -k_1(1 - \frac{2}{1 + \exp(-6/d_1 c_1 t)}) + v(0),$$

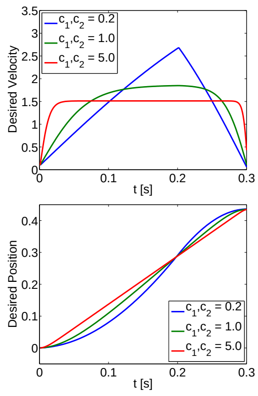
where k_1 is a constant depending on the desired set point p_1 (the exact dependence is not shown here but can be easily obtained via integrating the velocity profiles) and c_1 is a constant which specifies slope of the velocity profile. The deceleration profile has the following form :

$$v_D(t) = \tilde{k}_1(1 - \frac{2}{1 + \exp(-6c_2(1 - \frac{t-d_1}{d_2}))}),$$

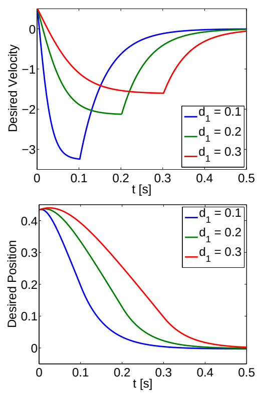
where d_2 is again the duration of the deceleration phase, c_2 sets the slope of the acceleration and $\tilde{k}_1$ is again chosen depending on the desired set point p_1. k_1 and $\tilde{k}_1$ is chosen such that the integral over the acceleration and the deceleration phase equals p_1, and that there is no jump in the velocity at the transition of acceleration to deceleration phase. At the end of the deceleration phase the desired velocity $v_D(d_1 + d_2)$ is always zero. d_1 and d_2 are shared for all joints while the remaining



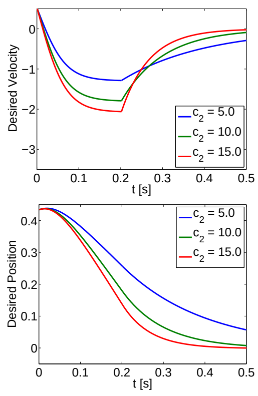
(a) m_1, d_1



(b) m_1, $c_1 + c_2$



(c) m_2, d_1



(d) m_2, c_2

Figure 10.7: The plots show the desired velocity profiles (top row) and positions trajectories (bottom row) for different parameter settings of template m_1 and m_2.

parameters can be chosen independently for each joint. Thus we have 3 parameters per joint (c_1, c_2 and p_1) which describe the shape of the velocity profile. The velocity profile is then integrated and used in combination with a PD trajectory tracking controller to define the policy of the template. The 2 PD-controller gains are also chosen by the template, resulting in 5 parameters per joint. Thus, including the 2 timing parameters d_1 and d_2, template m_1 has 12 parameters for the 2-link pendulum task and 22-parameters for the 4-link pendulum problem. The velocity profile and the resulting trajectories for different parameter settings can be seen in Figure 10.7(a) and (b).

After executing template m_1 the agent should ideally have reached the set-points p_i with zero velocity. However, since we deal with a highly non-linear system, the PD-controller is usually not able to track the trajectories perfectly and these conditions might be violated.

Motion template m_2 is used to stabilize the robot at the upright position. Thus, the desired setpoint is already given ($\phi_i = 0$ and $\phi_4 = \pi$). m_2 runs until the episode is terminated. The template uses the same acceleration profile than m_1. For stabilizing the robot at the upright position, the deceleration profile resembles a slowly decaying exponential function. The deceleration profile is given by

$$v_D(t) = \tilde{k}_2 \exp(-c_2(t - d_1))$$

Due to the tracking errors this profile is more suitable for stabilization at a setpoint than the deceleration profile of m_1. Again k_2 and $\tilde{k}_2$ are chosen such that the desired joint position converges to zero (or pi for ϕ_4) and that there is no jump in the velocity at the transition from the acceleration to the deceleration phase. The velocity profile and the resulting trajectory for different parameter settings can be seen in Figures 10.7(c) and (d). Template m_2 has 4 parameters per joint (c_1, c_2 and the 2 PD-controller gains), resulting in 9 (including d_1) parameters for the 2-link and 17 parameters for the 4-link balancing task.

Appendix A

List of Publications

1. G. Neumann *Batch-Mode Reinforcement Learning for Continuous State Spaces: A Survey*, In Proceedings of Oesterreichische Gesellschaft fuer Artificial Intelligence (OGAI) Journal, 2008
2. G. Neumann, M. Pfeiffer, and W. Maass. *Efficient Continuous-Time Reinforcement Learning with Adaptive State Graphs.*, In Proceedings of the 18th European conference on Machine Learning (ECML) 2007, pp. 250-261, Springer, 2007
3. G. Neumann and J. Peters. *Fitted Q-Iteration by Advantage Weighted Regression.* In Proceedings of Advances in Neural Information Processing Systems 22 (NIPS 2008), pp. 1177-1184. MIT Press, 2008.
4. H. Hauser, G. Neumann, A. Ijspeert and W. Maass. *Biologically Inspired Kinematic Synergies Provide a New Paradigm for Balance Control of Humanoid Robots.*, In Proceedings of the 7th IEEE RAS/RSJ Conference on Humanoids Robots (HUMANOIDS07), pp. 73-80, 2007
5. H. Hauser, G. Neumann, A. Ijspeert and W. Maass. *Biologically Inspired Kinematic Synergies enable Linear Balance Control of a Humanoid Robot.*, Biological Cybernetics, volume 104, pp. 235-249, 2011
6. G. Neumann, W. Maass and J. Peters. *Learning Complex Motions by Sequencing Simpler Motion Templates.*, In Proceedings of the 26th International Conference on Machine Learning (ICML 2009), pp. 753-760, 2009
7. E. Rueckert, G. Neumann, M. Toussaint and W. Maass. *Planning Movement Primitives.*, in preparation.
8. G. Neumann. *Variational Inference for Policy Search in Changing Situations.*, In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 817-824, 2011

A.1 Comments and Contributions to Publications

The first publication, *Batch-Mode Reinforcement Learning for Continuous State Spaces: A Survey*, is a survey paper written for the OGAI Journal. As it is only a review and was not peer reviewed it is not included in this thesis.

The paper *Efficient Continuous-Time Reinforcement Learning with Adaptive State Graphs* was written by myself (GN), Michael Pfeiffer (MP) and Wolfgang

Maass (WM). I developed the graph-based RL algorithm and conducted most of the experiments while MP implemented the reward prediction mechanism and did most of the paper writting. The paper was selected for oral presentation at the 18th European Conference for Machine Learning (ECML). Chapter 3 of this thesis is based on this paper.

The paper *Fitted Q-Iteration by Advantage Weighted Regression* was written by myself (GN) and Jan Peters (JP). I developed the theory and implemented the Advantage-Weighted Regression algorithms. I also conducted the experiments while JP provided the basic ideas and very helpful guidance for this paper. I also wrote the paper with significant improvements provided by JP. The paper was selected for a poster including spotlight presentation at the *22th Annual Conference on Neural Information Processing Systems* (NIPS) 2008. The Chapter 4 is based on this paper.

The paper *Biologically Inspired Kinematic Synergies Provide a New Paradigm for Balance Control of Humanoid Robots.* was written by Helmut Hauser (HH), myself, Auke Jan Ijspeert (AI) and my supervisor Wolfgang Maass (WM). HH conducted the experiments and implemented the linear control laws while GN implemented the synergies including the inverse kinematics optimization. HH and WM did most of the paper writting. AI provided many useful ideas for the design of the experiments. The paper got the best paper award at the 7th IEEE RAS/RSJ Conference on Humanoids Robots (HUMANOIDS07). It was subsequently extended to the journal version *Biologically Inspired Kinematic Synergies enable Linear Balance Control of a Humanoid Robot.* Chapter 6 of this thesis is based on this paper.

The paper *Learning Complex Motions by Sequencing Simpler Motion Templates* was written by myself, Wolfgang Maass (WM) and Jan Peters (JP). The algorithm design, implementation and the experiments have been conducted by GN while the initial basic idea was provided by WM. JP greatly helped to improve the paper writting and also provided useful guidance.

Chapter 8 is based on joint work with Elmar Rückert (ER), Marc Toussaint (MT) and my supervisor Wolfgang Maass (WM). While I provided the basic idea and guidance, ER did a great job in the realization of these ideas. Paper writting was done by ER and myself, with additional very useful input by MT and WM. While this work is not yet submitted, I included it in my thesis because I think it has great potential. It will hopefully be finished for submission in December 2011.

The paper *Variational Inference for Policy Search in Changing Situations* was completely written by myself.

Bibliography

Alexander, R. M. (1997). A Minimum Energy Cost Hypothesis for Human Arm Trajectories. *Biological Cybernetics*, 76:97–105. 28

Alexandrov, A., Frolov, A., and Massion, J. (1998). Axial Synergies during Human Upper Trunk Bending. *Experimental Brain Research*, 118(2):210–220. 52

Amari, S. (1998). Natural Gradient works efficiently in Learning. *Neural Computation*, 10:251–276. 126

Antos, A., Munos, R., and Szepesvari, C. (2008). Fitted Q-Iteration in continuous Action-Space MDPs. In *Advances in Neural Information Processing Systems 20*, pages 9–16. MIT Press, Cambridge, MA. 29

Arbib, M. A. (1981). Perceptual Structures and distributed Motor Control. *Handbook of physiology, section 2: The nervous system vol. ii, motor control, part 1*, pages 1449–1480. 76

Astrom, K. J. and Wittenmark, B. (1995). *Adaptive Control*. Addison Wesley Longman, second edition. Basics on adaptive control. 62, 70

Atkeson, C. and Stephens, B. (2007). Multiple Balance Strategies from one Optimization Criterion. In *Proceedings of the 7th International Conference on Humanoid Robots*, pages 57–64, Pittsburgh, USA. 86, 105, 141

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally Weighted Learning for Control. *Artificial Intelligence Review*, 11:75–113. 32, 33, 98

Baerlocher, P. and Boulic, R. (1998). Task-Priority Formulations for the Kinematic Control of highly Redundant Articulated Structures. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 1, pages 323–329. 51

Baerlocher, P. and Boulic, R. (2004). An Inverse Kinematics Architecture enforcing an arbitrary Number of strict Priority Levels. *The Visual Computer*, 20(6):402–417. 51

Baird, L. C. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. In *International Conference for Machine Learning (ICML)*. 8, 14

Baxter, J. and Bartlett, P. (1999). Direct Gradient-Based Reinforcement Learning: I. Gradient Estimation Algorithms. Technical report. 125

Bertsekas, D. P. and Tsitsiklis, J. N. (1998). *Neuro Dynamic Programming*. Athena Scientific. 7, 8

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York. 133

Bizzi, E., Cheung, V., d'Avella, A., Saltiel, P., and M., T. (2008). Combining Modules for Movement. *Brain Research Reviews*, 57:125–133. 43, 46, 93, 132

Boyan, J. A. (1999). Least-Squares Temporal Difference Learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. 10

Boyan, J. A. and Moore, A. W. (1995). Generalization in Reinforcement Learning: Safely Approximating the Value Function. In *Advances in Neural Information Processing Systems 7*, pages 369–376. MIT Press. 9, 14, 28

Bradtke, S. J. and Duff, M. O. (1995). Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In *Advances in Neural Information Processing Systems 7*, volume 7, pages 393–400. 77

Chand, S. and Doty, K. (1985). Online Polynomial Trajectories for Robot Manipulators. *International Journal of Robotic Research*, 4:38–48. 47

Chhabra, M. and Jacobs, R. A. (2006). Properties of Synergies Arising from a Theory of Optimal Motor Behavior. *Neural Computation*, 18:2320–2342. 46, 93

Dautenhahn, K. and Nehaniv, C. L. (2002). *Imitation in Animals and Artifacts*. MIT Press, Campridge. 76

d'Avella, A. and Bizzi, E. (2005). Shared and Specific Muscle Synergies in Natural Motor Behaviors. *Proceedings of the National Academy of Sciences (PNAS)*, 102(3):3076–3081. 41, 42, 46, 51, 58

d'Avella, A., Saltiel, P., and Bizzi, E. (2003). Combinations of Muscle Synergies in the Construction of a Natural Motor Behavior. *Nature*, 6(3):300–308. 51, 89, 92, 93

de Boer, P.-T., Kroese, D., Mannor, S., and Rubinstein, R. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1):19–67. 29, 128

Deisenroth, M. P., Rasmussen, C. E., and Peters, J. (2009). Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7-9):1508–1524. 8, 9, 10, 11

Doya, K., Samejima, K., Katagiri, K., and Kawato, M. (2000). Multiple Model-based Reinforcement Learning. *Neural Computation*, 14:1347–1369. 14

Ernst, D., Geurts, P., and Wehenkel, L. (2003). Iteratively Extending Time Horizon Reinforcement Learning. In *European Conference on Machine Learning (ECML)*, pages 96–107. 8

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Resource*, 6:503–556. 8, 10, 11, 29, 34, 37, 77, 78, 80

Ernst, D., Stan, G.-B., Goncalves, J., and Wehenkel, L. (2006). Clinical Data based Optimal sti Strategies for hiv; a Reinforcement Learning Approach. In *Machine Learning Conference of Belgium and The Netherlands (Benelearn)*, pages 65–72. 11

Freitas, S. M. S. F., Duarte, M., and Latash, M. L. (2006). Two Kinematic Synergies in Voluntary Whole-Body Movements during Standing. *Jorunal of Neurophysiology*, 95(2):636–645. 52

Furmston, T. and Barber, D. (2010). Variational Methods for Reinforcement Learning. In *Proceedings of the Thirteenth Conference on Artificial Intelligence and Statistics (AISTATS)*. 131, 134

Ghavamzadeh, M. and Mahadevan, S. (2003). Hierarchical Policy Gradient Algorithms. In *International Conference for Machine Learning (ICML)*, pages 226–233. AAAI Press. 76

Goswami, A. (1999). Postural stability of biped robots and the foot rotation indicator (FRI) point. *The International Journal of Robotics Research*, 18(6):523–533. 50

Goswami, A. and Kallem, V. (2004). Rate of Change of Angular Momentum and Balance Maintenance of Biped Robot. In *Proceedings of the 2004 IEEEE International Conference on Robotics and Automation ICRA*, volume 4, pages 3785–3790. 50

Greensmith, E., Bartlett, P., and Baxter, J. (2004). Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal Machine Learning Resource*, 5:1471–1530. 125

Guestrin, C. E. and Ormoneit, D. (2001). Robust Combination of Local Controllers. In *Proc. UAI*. 15, 17

Guez, A., Vincent, R. D., Avoli, M., and Pineau, J. (2008). Adaptive Treatment of Epilepsy via Batch-Mode Reinforcement Learning. In *Proceedings of the 20th national conference on Innovative Applications of Artificial Intelligence - Volume 3*, pages 1671–1678. AAAI Press. 11

Hansen, N., Muller, S., and Koumoutsakos, P. (2003). Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18. 94, 124

Hauser, H., Neumann, G., Ijspeert, A. J., and Maass, W. (2007). Biologically Inspired Kinematic Synergies Provide a New Paradigm for Balance Control of Humanoid Robots. In *Proceedings of the 7th IEEE RAS/RSJ Conference on Humanoids Robots (HUMANOIDS07)*, Pittsburgh, PA. 25, 52, 62, 63, 68

Hauser, H., Neumann, G., Ijspeert, A. J., and Maass, W. (2011). Biologically Inspired Kinematic Synergies enable Linear Balance Control of a Humanoid Robot. *Biological Cybernetics*, 104:235–249. 41, 73

Heidrich-Meisner, V. and Igel, C. (2009a). Hoeffding and Bernstein Races for Selecting Policies in Evolutionary Direct Policy Search. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408. ACM. 102

Heidrich-Meisner, V. and Igel, C. (2009b). Neuroevolution Strategies for Episodic Reinforcement Learning. *Journal of Algorithms*, 64(4):152–168. 94, 98, 124, 127, 128, 132, 134

Huber, M. and Grupen, R. A. (1998). Learning Robot Control—Using Control Policies as Abstract Actions. In *In NIPS'98 Workshop: Abstraction and Hierarchy in Reinforcement Learning*. 76

Ijspeert, A. J. and Schaal, S. (2003). Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems 15*, (NIPS 2003), pages 1523–1530. MIT Press, Cambridge, MA. 76, 132

Jong, N. K. and Stone, P. (2007). Model-Based Function Approximation for Reinforcement Learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*. 9

Kagami, S., Kanehiro, F., Tamiya, Y., Inaba, M., and Inoue, H. (2001). AutoBalancer: An Online Dynamic Balance Compensation Scheme for Humanoid Robots. In Donald, B., Lynch, K., and Rus, D., editors, *Algorithmic and Computational Robotics: New Directions*, pages 329–340. A K Peters Ltd. 51

Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., and Hirukawa, H. (2001). The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking patttern generation. In *Proceedings of the 2001 IEEEE/RSJ International Conference on Intelligent Robots and Systems, Maui*, pages 239–246. 51

Kajita, S., Yamaura, T., and Kobayashi, A. (1992). Dynamic walking control of a biped robot along a potential energy conserving orbit. *Robotics and Automation, IEEE Transactions on*, 8(4):431–438. 51

Kappen, H. J. (2007). An Introduction to Stochastic Control Theory, Path Integrals and Reinforcement Learning. In *Cooperative Behavior in Neural Systems*, volume 887 of *American Institute of Physics Conference Series*, pages 149–181. 91, 94

Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Trans. on Robotics and Automation (ICRA)*, 12(4). 14, 17

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning Stable Non-Linear Dynamical Systems with Gaussian Mixture Models. *IEEE Transaction on Robotics*, 27(5):943–957. 93

Kietzmann, T. C. and Riedmiller, M. (2009). The Neuro Slot Car Racer: Reinforcement Learning in a Real World Setting. In *International Conference on Machine Learning and Applications (ICMLA)*, pages 311–316. 11

Kober, J., Mohler, B. J., and Peters, J. (2008). Learning Perceptual Coupling for Motor Primitives. In *Intelligent Robots and Systems (IROS)*, pages 834–839. 41, 42, 44, 70

Kober, J., Oztop, E., and Peters, J. (2010). Reinforcement Learning to adjust Robot Movements to New Situations. In *Proceedings of the 2010 Robotics: Science and Systems Conference (RSS 2010)*. 44, 90, 92, 132, 133, 140

Kober, J. and Peters, J. (2010). Policy Search for Motor Primitives in Robotics. *Machine Learning Journal*, online first:1–33. 2, 44, 76, 90, 94, 110, 124, 126, 127, 132, 134, 135

Kohl, N. and Stone, P. (2003). Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *International Conference for Robotics and Automation (ICRA)*. 127

Kolter, J. Z. and Ng, A. Y. (2009a). Regularization and Feature Selection in Least-Squares Temporal Difference Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 521–528, New York, NY, USA. ACM. 10

Kolter, Z. and Ng, A. (2009b). Task-Space Trajectories via Cubic Spline Optimization. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 2364–2371, Piscataway, NJ, USA. IEEE Press. 43, 47

Kuffner, J. and LaValle, S. (2000). RRT-Connect: An efficient Approach to Single-Query Path Planning. In *Proceedings of the 2000 IEEEE International Conference on Robotics and Automation ICRA, San Francisco,CA*, pages 995–1001. 14, 95

Kuo, B. C. and Golnaraghi, F. (2002). *Automatic Control Systems*. John Wiley & Sons, Inc., 8th edition. 62

Lagoudakis, M. G. and Parr, R. (2003). Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149. 10

Lee, S.-H. and Goswami, A. (2007). Reaction Mass Pendulum (RMP): An Explicit Model for Centroidal Angular Momentum of Humanoid Robots. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4667–4672. 51

Lin, L.-J. (1992). Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8:293–321. 8

Mannor, S., Rubinstein, R., and Gat, Y. (2003). The Cross Entropy method for Fast Policy Search. In *Proceedings of the 20th International Conference on Machine Learning*, (ICML 2003), pages 512–519, Washington, DC, USA. 128, 132

Mansard, N. and Chaumette, F. (2007). Task Sequencing for High-Level Sensor-Based Control. *Robotics, IEEE Transactions on*, 23(1):60–72. 51

Mason, C. R., Gomez, J. E., and Ebner, T. J. (2001). Hand Synergies during Reach-to-Grasp. *J Neurophysiol*, 86(6):2896–2910. 52

Michel, O. (2004). Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems*, 1:39–42. 63

Mitchell, J. S. B. and Papadimitriou, C. H. (1991). The Weighted Region Problem: Finding Shortest Paths through a Weighted Planar Subdivision. 13

Mitrovic, D., Klanke, S., and Vijayakumar, S. (2010). Adaptive Optimal Feedback Control with Learned Internal Dynamics Models. In *From Motor Learning to Interaction Learning in Robots*, pages 65–84. 95

Moore, A. W. and Atkeson, C. G. (1993). Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time. *Machine Learning*, 13:103. 9, 15, 22

Moore, A. W. and Atkeson, C. G. (1995). The Parti-game Algorithm for Variable Resolution RL in Multidimensional State-spaces. *Machine Learning*, 21. 15

Mülling, K., Kober, J., and Peters, J. (2010). A Biomimetic Approach to Robot Table Tennis. pages 1921–1926, Piscataway, NJ, USA. Max-Planck-Gesellschaft, IEEE. 2

Munos, R. and Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning*, 49. 15

Mussa-Ivaldi, F. A. (1999). Modular Features of Motor Control and Learning. *Current Opinion in Neurobiology*, 9:713–717. 51

Nakanishi, J., Morimoto, J., Endo, G., Cheng, G., Schaal, S., and Kawato, M. (2004). Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47:79?–91. 42

Neumann, G. (2011). Variational Inference for Policy Search in Changing Situations. In *Proceedings of the 28th International Conference on Machine Learning*, (ICML 2011), pages 817–824, New York, NY, USA. ACM. 42, 43, 44, 45, 110, 128, 129

Neumann, G., Maass, W., and Peters, J. (2009). Learning Complex Motions by Sequencing Simpler Motion Templates. In *Proceedings of the 26th International Conference on Machine Learning*, (ICML 2009), pages 753–760, Montreal, Canada. 43, 44, 45, 48, 77, 78, 79, 89, 92, 132

Neumann, G. and Peters, J. (2009). Fitted Q-Iteration by Advantage Weighted Regression. In *Advances in Neural Information Processing Systems 22 (NIPS 2008)*. MA: MIT Press. 7, 10, 11, 37, 87

Neumann, G., Pfeiffer, M., and Maass, W. (2007). Efficient Continuous-Time Reinforcement Learning with Adaptive State Graphs. In *Proceedings of the 18th European conference on Machine Learning*, ECML '07, pages 250–261, Berlin, Heidelberg. Springer-Verlag. 7, 9, 25

Nguyen-Tuong, D., Peters, J., Seeger, M., and Schölkopf, B. (2008a). Learning Inverse Dynamics: A Comparison. In *16th European Symposium on Artificial Neural Networks*, (ESANN 2008), pages 13–18, Bruges, Belgium. 91, 98

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2008b). Local Gaussian Process Regression for Real Time Online Model Learning and Control. In *Proceedings of 22nd Annual Conference on Neural Information Processing Systems*, (NIPS 2008), pages 1193–1200, Vancouver, BC, Canada. 91, 98

Oppenheim, A. V. and Willsky, A. S. (1992). *Signal and Systems*. Prentice-Hall Inc., Englewood Cliffs. 61

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and Generalization of Motor Skills by Learning from Demonstration. In *International Conference on Robotics and Automation (ICRA 2009)*. 90, 92, 100, 103, 111

Peters, J., Mistry, M., Udwadia, F. E., Nakanishi, J., and Schaal, S. (2008). A Unifying Methodology for Robot Control with Redundant DOFs. *Autonomous Robots*, (1):1–12. 42, 47, 112

Peters, J., Mülling, K., and Altun, Y. (2010). Relative Entropy Policy Search. In *Proceedings of the 24th National Conference on Artificial Intelligence*, (AAAI 2010), Physically Grounded AI Track., pages 1607–1612. AAAI Press. 110

Peters, J. and Schaal, S. (2006). Policy Gradient methods for Robotics. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS)*, Beijing, China. 2, 42, 44, 124, 127

Peters, J. and Schaal, S. (2007a). Policy Learning for Motor Skills. In *Proceedings of 14th International Conference on Neural Information Processing (ICONIP)*. 29

Peters, J. and Schaal, S. (2007b). Reinforcement Learning by Reward-Weighted Regression for Operational Space Control. In *Proceedings of the International Conference on Machine Learning (ICML)*. 30, 32, 33

Peters, J. and Schaal, S. (2008a). Natural Actor-Critic. *Neurocomputation*, 71(7-9):1180–1190. 124, 126

Peters, J. and Schaal, S. (2008b). Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, (4):682–97. 92, 94, 110

Popovic, M., Goswami, A., and Herr, H. (2005). Ground Reference Points in Legged Locomotion: Definitions, Biological Trajectories and Control Implications. *International Journal of Robotics Research*. 50

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley. 16

Rawlik, K., Toussaint, M., and Vijayakumar, S. (2010). An Approximate Inference Approach to Temporal Optimization in Optimal Control. In *Proceedings of 24nd Annual Conference on Neural Information Processing Systems*, (NIPS 2010), pages 2011–2019, Vancouver, BC, Canada. 116

Riedmiller, M. (2005). Neural fitted Q-Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *Proceedings of the European Conference on Machine Learning (ECML)*. 10, 11, 28, 29, 34, 78

Riedmiller, M., Gabel, T., Hafner, R., and Lange, S. (2009). Reinforcement Learning for Robot Soccer. *Auton. Robots*, 27:55–73. 11

Sabatini, A. M. (2002). Identification of Neuromuscular Synergies in Natural Upper-Arm Movements. *Biol Cybern*, 86(4):253–262. 52

Schaal, S., Mohajerian, P., and Ijspeert, A. (2007). Dynamics Systems vs. Optimal Control: a Unifying View. *Progress in Brain Research*, 165:425–445. 41, 43, 132

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. J. (2003). Learning Movement Primitives. In *International Symposium on Robotics Research*, (ISRR 2003), pages 561–572. 41, 43, 44, 89, 90, 92, 105, 109, 111

Sciavicco, L. and Siciliano, B. (2005). *Modelling and Control of Robot Manipulators (Advanced Textbooks in Control and Signal Processing)*. Advanced textbooks in control and signal processing. Springer, 2nd edition. 47, 52, 57, 70

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-Exploring Policy Gradients. *Neural Networks*, 23(4):551–559. 94, 127, 128

Sutton, R. (1996). Generalization in Reinforcement Learning: Successful Examples using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press. 1, 8, 10, 28, 35

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Boston, MA. 10, 14, 16, 19, 27, 28, 31, 35

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems (NIPS)*. 14, 76, 125

Theodorou, E., Buchli, J., and Schaal, S. (2010a). Reinforcement Learning of Motor Skills in High Dimensions: a Path Integral Approach. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2397–2403. 2, 92, 94, 102, 104, 110

Theodorou, E., Tassa, Y., and Todorov, E. (2010b). Stochastic Differential Dynamic Programming. In *Proceedings of the 29th American Control Conference*, (ACC 2010), Baltimore, Maryland, USA. 44, 124, 126

Timmer, S. and Riedmiller, M. (2007). Fitted Q-Iteration with CMACs. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL*, pages 1–8. 10, 29

Todorov, E. and Jordan, M. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5:1226–1235. 90, 101, 110

Todorov, E. and Li, W. (2005). A generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems. In *Proceedings of the 24th American Control Conference*, volume 1 of *(ACC 2005)*, pages 300 – 306, Portland, Oregon, USA. 91, 94, 95, 113

Torres-Oviedo, G. and Ting, L. H. (2007). Muscle Synergies Characterizing Human Postural Responses. *J Neurophysiol*, 98(4):2144–2156. 52

Toussaint, M. (2009). Robot Trajectory Optimization using Approximate Inference. In *Proceedings of the 26th International Conference on Machine Learning*, (ICML 2009), pages 1049–1056, Montreal, Canada. 91, 94, 95, 99, 112, 113, 114, 115, 116, 134, 144

Toussaint, M. (2011). Lecture Notes: Gaussian Identities. Technical report, Freie Universität Berlin. 114, 115

Tricon, V., Pellec-Muller, A. L., Martin, N., Mesure, S., Azulay, J.-P., and Vernazza-Martin, S. (2007). Balance Control and Adaptation of Kinematic Synergy in aging Adults during Forward Trunk Bending. *Neuroscience Letter*, 415(1):81–86. 52

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). Incremental Online Learning in High Dimensions. *Neural Computation*, 17(12):2602–2634. 91, 98

Viviani, P. and Flash, T. (1995). Minimum-Jerk, Two-Thirds Power Law, and Isochrony: Converging Approaches to Movement Planning. *Journal of Experimental Psychology: Human Perception and Performance*, 21(1):32–53. 28

Vlassis, N., Toussaint, M., Kontes, G., and Piperidis, S. (2009). Learning Model-Free Robot Control by a Monte Carlo EM Algorithm. *Autonomous Robots*, 27(2):123–130. 126, 134

Vukobratović, M. and Borovac, B. (2004). Zero-Moment Point - Thirty Five Years of its Life. *International Journal of Humanoid Robotics*, 1:157–173. 50

Wang, Y., Zatsiorsky, V. M., and Latash, M. L. (2005). Muscle Synergies Involved in Shifting the Center of Pressure while making a first Step. *Exp Brain Res*, 167(2):196–210. 52

Wawrzynski, P. (2009). Real-time Reinforcement Learning by sequential Actor-Critics and Experience Replay. *Neural Netw.*, 22:1484–1497. 2, 8, 11

Whitney, D. (1969). Resolved Motion Rate Control of Manipulators and Human Prostheses. *Man-Machine Systems, IEEE Transactions on*, 10(2):47 –53. 51

Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. (2008). Episodic Reinforcement Learning by Logistic Reward-Weighted Regression. In *Proceedings of the 18th international conference on Artificial Neural Networks, Part I*, (ICANN 2008), pages 407–416, Berlin, Heidelberg. Springer-Verlag. 94

Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256. 94, 124, 125

Winter, D. A. (1995). Human Balance and Posture Control during Standing and Walking. *Gait and Posture*, 3(4):193–214. Review article. 50

Wolpert, D. (1998). Internal Models in the Cerebellum. *Trends in Cognitive Sciences*, 2(9):338–347. 28

Xu, X. and Antsaklis, P. (2002). An Approach to Optimal Control of Switched Systems with Internally Forced Switchings. In *Proceedings of american control conference*, pages 148–153, Achorage, USA. 76