Robot Learning using Tree-Based Policy Representation

An Approach Based on Gaussian Mixture Models

M.J.A. Zeestraten

Master of Science Thesis

TU Delft Delft University of Technology

BioMechanical Engineering

Robot Learning using Tree-Based Policy Representation

An Approach Based on Gaussian Mixture Models

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

M.J.A. Zeestraten

July 30, 2013

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The research for this thesis is performed at the Instituto Italiano di Tecnologia (IIT) in Genova.

Delft University of Technology
Department of
Bio Mechanical Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Robot Learning using Tree-Based Policy Representation

by

M.J.A. Zeestraten

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: July 30, 2013

Supervisor(s):

Prof. Dr. Ir. P.P. Jonker

Dr. Ir. W. Caarls

Dr. S. Calinon (IIT)

Reader(s):

Ir. I. Grondman

Acknowledgements

About a year ago I took the coarse Probabilistic Robotics at Delft University of Technology (TU Delft). This course introduced me to the world of Programming by Demonstration (PbD), a very interesting research field in robot learning. I decided I wanted to learn more about PbD and Learning from Exploration (LfE) and contacted my (future) supervisor Sylvain Calinon. He was happy to present me with a challenging research topic for my MSc. Thesis and give me the possibility to do this research at Instituto Italiano di Tecnologia (IIT) in Genova. From November 2012 till April 2013 I spent my days at IIT (and at the mountains in the weekend), to carry out the research that is presented in this Thesis.

First of all I want to thanks Sylvain for his invitation, assignment and feedback on my work. It was nice working with him and learn about his research in the field PbD with a statistical approach. I also want to thank my other colleagues at the IIT, who were always there to answer a question. Special thanks go to Danilo Bruno, who was able to explain to me the math I had to deal with, and to Wouter Caarls who reviewed the first draft of my thesis.

Without a warm place in Italy, I wouldn't have survived. Therefore, I would like to thank Hamza, Isuru and Nawid for their moral support during my stay in Italy. I really enjoyed this international home in the middle of nowhere.

Finally, I want to thank my girlfriend Jana. She supported me while I was doing my work both in the Netherlands and in Italy via Skype or by visiting my Italian home.

Abstract

Learning is an important aspect in creating versatile robots. Pre-programming a robot to acquire a wide variety of skills in an ever changing environment is unfeasible. Robot learning provides a promising alternative. Two well-established learning techniques are Programming by Demonstration (PbD) and Learning from Exploration (LfE). PbD and LfE are often combined to strengthen each other [1–4]. PbD is used because it allows fast learning: with only a few demonstrations, robots are able to reproduce tasks with reasonable performance. After these demonstrations, LfE is used to improve the robot's task performance or to adjust this skills to changing environments.

Robots often use continuous mappings between states and actions to represent a skill. Such mappings are called policies and are represented by function approximators. The shape of the policy is determined by the parameters $\boldsymbol{\theta}$. During learning the robot tries to find optimal $\boldsymbol{\theta}$ for the policy. As the complexity of the skill increases, the number of parameters required to accurately describe the policy for this skill also increases. As the number of parameters increases, the complexity of the solution space increases as well. It is most likely that the LfE algorithm requires more trials to converge for these complex search spaces than simpler search spaces, thus the LfE performance decreases as the complexity of the search space increases.

In this thesis a novel multi-resolution policy representation is investigated. The method, called Tree-based policy representation, creates a multi-resolution model based on demonstration data. After this initialization, LfE can use the structure of the Multi-resolution policy to increase learning performance.

The method is tested in multiple experimental scenarios. The Tree-based policy representation achieves better learning performance compared to conventional 'flat' policy representations, when learning motions that clearly have a multi-resolution aspect. In other cases, the Tree-based movement representation performs equally well or worse compared to standard 'flat' policy representations.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motivation

The prospects of robotics applied in a wide variety of fields are promising. Robots are expected to assist humans in household, health-care and many other fields. However, until today robots are mostly used at factory floors, these robots are pre-programmed to perform one specific task. In such tasks, robots easily outperform humans in both speed and accuracy, because they operate in a well-structured environment. In such environments, robots have almost no uncertainty about the state of their environment. The way in which robots operate should change to enable them to leave factory floors and become the versatile robots of which humans have dreamed for centuries. The control architecture and mechanical design of the robot should be designed such that it allows safe Human-robot interaction. And, even more important, robots should be able to learn from their environment.

Learning is an important aspect in creating versatile robots. Pre-programming a robot to acquire a wide variety of skills in an ever changing environment is unfeasible. Programmers are not able to program robot behavior for every possible scenario. Learning algorithms enable robots to learn new skills and adapt their skills to changing environments.

Related work

Two well-established learning techniques are Programming by Demonstration (PbD) and Learning from Exploration (LfE). In PbD [8, 9], the robot observes several demonstrations and tries to extract the important aspects of the task. After learning the robot should be able to reproduce the task even in different situations. In the field of machine learning, PbD is part of the supervised learning techniques. PbD is also referred to as Learning by Imitation or Teaching by Showing. Actively showing the robot *how* the task is executed is a key element of this technique.

In LfE, the robot tries to learn a task by maximizing a certain performance measure. This performance measure, usually called *reward*, indicates how well the robot executes the task.

Instead of telling the Robot how to perform the task, it is told how good it is performing. In LfE the robot finds the optimal task performance by 'trial-and-error', i.e. exploration.

PbD and LfE are often combined to strengthen each other [1–4]. PbD is used because it allows fast learning. With only a few demonstrations, robots are able to reproduce tasks with reasonable performance. After demonstration, LfE is used to improve the robot's task performance or adjust to his skills to changing environments.

Problem statement and Hypothesis

Robots use continuous state-action spaces. The state can consist of time, the robot joint position, end-effector pose or the position of skill related objects such as a ball. A robot performs actions which change the state of the environment. A skill is represented as continuous mappings between states and actions, called policy π. Given a certain state of the world, the policy outputs an action.

These continuous policies are represented by function approximators. The shape of the policy is determined by the parameters $\boldsymbol{\theta}$. While learning, the robot tries to find optimal $\boldsymbol{\theta}$ for the policy. For LfE this results in parameters that yield the maximum reward. Skills consist of complex motions often require more complex policies with a larger number of policy parameters compared to skills that require only simple motions. The large number of parameters can be problematic for LfE algorithms. LfE algorithms search the parameter space for an optimal solution. The search complexity increases with the number of model parameters. This is also referred to as the *Curse of dimensionality* [10]. As the complexity of the search space increases, the learning performance decreases. The learning performance can be measured by the number of trials required to converge to a solution and the quality of the solution.

As complexity of the search space increases, it is most likely that the LfE algorithm requires more trials to converge compared to simpler search spaces. As a result the algorithm requires more time to learn these complex tasks. Furthermore, although LfE algorithms used for continuous state-action spaces are guaranteed to converge, they are not guaranteed to converge to the global optimum. The increased complexity of the search space also increases the probability of more local optima. Therefore, the risk of converging to (bad) local optima increases, when complexity of search space increases.

Proposed Solution

In this thesis a novel method is proposed to reduce the performance loss for complex search spaces. The method is referred to as a Tree-based policy representation, in short: Tree-model. The Tree-model represents the policy at different levels of refinement, or, in other words, using multiple resolutions. The coarsest level uses only a few parameters to represent the policy in a very rough way. At subsequent levels the policy becomes more refined and the parameter space becomes more complex.

The method relies on both PbD and LfE. Learning consists of two phases; (i) Initialization and (ii) Exploration. In the initialization phase a Tree-model is formed based on demonstration data. Initialization determines the number of layers, the level of refinement at each layer, and

the initial values of the parameters at each layer. In the exploration phase, the policy improves by using a learning algorithm that is based on exploration. Exploration starts at the coarsest level to learn the rough policy. After convergence, exploration will continue at subsequent layers. It is hypothesized that learning using a multi-resolution policy representation will result in a higher learning performance compared to a 'flat'-model representation with only one level of refinement.

Both learning phases provide a challenging research topic. The first phase requires a learning algorithm that has the ability to distinguish different levels of refinement. The second phase requires an efficient exploration algorithm that is able to automatically switch from layer to layer.

The goal of the Tree-based movement representation is to increase learning performance during exploration. The focus of this thesis lies on the development of an exploration algorithm. Development of an algorithm that is able to autonomously determine the Tree-structure is left for future work.

Structure of Thesis

This thesis starts by introducing the concept of learning in robotics in Chapter 2. The Chapter describes policy representations for robots that use continuous state and action spaces. Furthermore, PbD and LfE are discussed in more detail. Chapter 3 introduces the method proposed in this thesis. The chapter is split in two parts: construction of the Tree-model, and exploration of the Tree-model. The Tree-model is evaluated in multiple simulated experiments. These experiments and their results are presented in Chapter 4. Finally, the conclusions are presented in Chapter 5.

Chapter 2

Robot Learning

In this Thesis we describe a novel multi-resolution policy representation for Robot learning. Before introducing this policy representation, this chapter introduces the concept of Robot learning.

To introduce robots into daily activities such as our household the way in which robots obtain new skills should change. To date, most robots are pre-programmed to perform one specific task. Everyday users do not have skills and expertise to program a robot. Therefore, creating robotic platforms that do not rely on the user's programming skills is desired. Instead of programming a robot, one could create a robot that is able to learn new skills. Like humans, robots can learn in two ways; (i) by exploration and (ii) by imitation. Learning has several advantages for robotics. Learning allows robots to adjust to the uncertain and changing environment we live in. Furthermore, learning allows everyday users to teach robots new skills.

To understand what learning a skill actually means, we first need to understand how a robot executes a skill. The robot and the environment are described by their state $\boldsymbol{s}_t$. This state contains all relevant information about the robot's joint angles, angular velocities, and location of skill related objects in the environment, etc. Each time the robot performs an action $\boldsymbol{a}_t$, the action affects the state $\boldsymbol{s}_t$ resulting in a new state $\boldsymbol{s}_{t+1}$. When one *assumes* that the next state $\boldsymbol{s}_{t+1}$ only depends on the current state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$, the environment has the *Markov property*. The assumption of the environment having the Markov property lies at the basis of many learning algorithms.

The robot is able to perform a skill successfully, if it is able to produce the desired sequence of states $\boldsymbol{s}_t \dots \boldsymbol{s}_T$, or reach a terminal state. The robot can only control the state by applying actions $\boldsymbol{a}_t$. Therefore, skill-learning comes down to finding the appropriate mapping between states and actions. This mapping is called a *policy* and is denoted by $\pi(\boldsymbol{s})$.

Robots have a continuous state and action space. Theoretically, this means that a robot has an infinite number of states and actions. Hence, the policy should be a continuous mapping between states and actions. This continuous mapping is described by a function approximator $\pi(\boldsymbol{s}, \boldsymbol{\theta})$. Here $\boldsymbol{\theta}$ represents a vector of parameters that influences the shape of the policy.

The field of robotics has focused on two ways to learn the policy, namely: Programming by Demonstration (PbD) (also referred to as Imitation Learning or Teaching by Showing) [8, 9], and Learning from Exploration (LfE)[1] [10].

In this chapter we first discuss different policy representations that can be used for continuous state and action spaces (Section 2-1). PbD and LfE are discussed in more detail in Section 2-2.

2-1 Policy Representations

Policy representation is an important aspect in robot learning with continuous state-action spaces. One class of policy representations discretizes the complete state/action space of a robot and use additional function approximators to make the policy applicable to robots with continuous state-action spaces. For robots with a large number of degrees of freedom such as humanoids, this approach severely suffers from the curse-of-dimensionality [10, 11]. This type of policy representation can for example be found in classical Value-based RL [12].

Another class of policy representations, used in policy search approaches (see Section 2-2-2), does not cover the complete state-action space. Instead, a policy representation with a smaller policy space describing only a part of the state-action space is used. This policy representation uses a pre-defined structure in the form of a function approximator. The function approximator represents a continuous mapping between state and actions. The general structure of the policy is given by:

$$\boldsymbol{a} = \pi(\boldsymbol{s}, \boldsymbol{\theta}) \tag{2-1}$$

The policy is a function of two parameters, the state $\boldsymbol{s}$ and the policy parameters $\boldsymbol{\theta}$. The use of function approximators combined with state-of-the-art LfE algorithms and PbD, has proven to be successful in a variety of robotic applications [13–16]. In this work we refer to this type of policies as 'model-based' policies.

Model-based policies usually do not directly output a motor command. Instead, the policy outputs a desired position and/or velocity that is tracked by a low-level PD-controller. Learning skills, thus comes down to learning the desired motion that, when executed, results in the desired goal.

Designing a good model-based policy for robotics is challenging. Ijspeert et al. [17] summarized five desirable properties that a policy representation for movements should possess from the viewpoint of PbD:

- Ease of learning goal trajectories;
- Compactness of the policy representation;
- Robustness against perturbation in dynamic environments;
- Ease of reusing movements to modifications of the task;

[1] A more familiar term for LfE might be Reinforcement Learning (RL). However, the concept RL might be restrictive since it does not includes exploration techniques like Black Box Optimization (BBO), Stochastic optimization and evolutionary search techniques.

- Ease of categorization for movement recognition.

More recently, Kormushev et al.[11] describe 15 challenges for designing a model-based policy. Among these challenges are:

- *Smoothness*, the policy should produce smooth movements without sudden accelerations or jerks;
- *Gradual Exploration*, the policy should allow gradual exploration. This is an important aspect to prevent injuries to both robot and human operator;
- *Correlations*, it would be desirable to have a policy representation which inhibits correlation along the different variables similar to the motor synergies found in animals;
- *Multi-resolution*, different parts of the policy should have a different resolution/precision.

In the remainder of this section, model-based policies used in robotic applications are discussed. A distinction is made between policies that use a 'flat', or single resolution, and policies that use a multi-resolution or hierarchical approach.

2-1-1 Flat Policy Representation

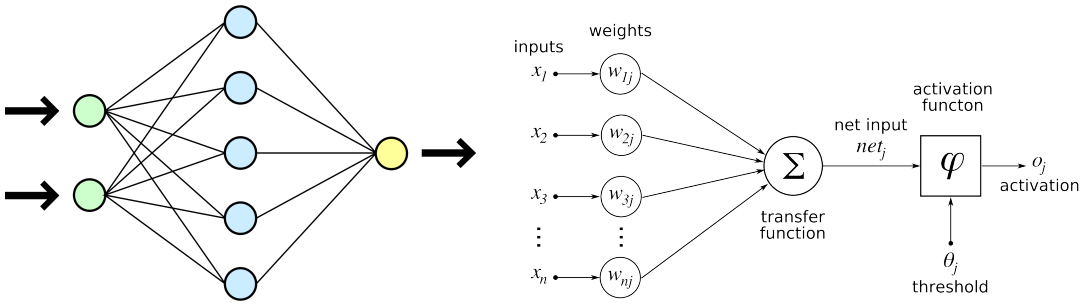
The majority of model-based policy representations use a flat policy representation. In this section a number of this type of policy representations is presented.

Neural Networks

A very generic function approximator used in robotics is a multi-layer Neural Network as shown in Figure 2-1. Neural Networks use a network of artificial neurons to represent a non-linear function. The parameters of the Neural Network are the weights attached to the paths that link the Neurons. By adjusting the weights of artificial Neurons (figure 2-1b), the mapping of the network can be adjusted. Learning the weights of a Neural Network can be done using backward-propagation. Neural Networks have been applied in the field of PbD [8, 9] and LfE. They are generic approximators, but it is hard to relate the parameters to specific parts of the policy. This makes the use of Neural networks less intuitive compared to other the other representations given in this section.

Spline-based Representation

Spline-based trajectories can be used to represent (robot) motion [18]. Trajectories are represented using multiple simple polynomials connected together and running through multiple via-points. The connections of splines are constrained to be smooth, i.e. at the connection point their spatial position and derivative are zero. These splines can be learned both from Demonstration and Exploration [19]. The number of via-points determines complexity of the policy that can be represented. Spline-based methods allow a compact representation of the policy and generate smooth trajectories.



(a) Artifical Network **(b)** Artificial Neuron

Figure 2-1: The structure of a multi-layer Neural network. (a) Shows the different layers from the Neural Network; left the input layer, middle the hidden layer, right the output layer. Each circle represents an artificial neuron. (b) Shows the contents of a neuron: Each input signal weighted and summed. The resulting signal results in the activation of this neuron (usually $0 \leq o_j \leq 1$). Both images are adopted from wikipedia.org.

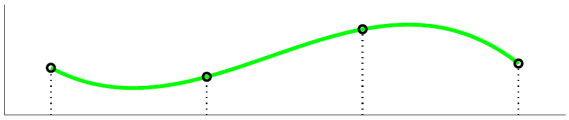


Figure 2-2: A spline-based trajectory consisting of 3 splines and is described by 4 via-points (knots).

GMM-GMR

Another approach uses the Gaussian Mixture Model (GMM) combined with Gaussian Mixture Regression (GMR), for example [20]. The movement of the robot is encoded by defining a joint Probability Density Function (pdf) $\mathcal{P}(\cdot)$ over the state-action space using multivariate Gaussian pdf. Although one Gaussian, $\mathcal{N}(\boldsymbol{x}|\mu, \Sigma)$, can only define a linear relation, multiple Gaussians can define non-linear relations:

$$\mathcal{P}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)., \tag{2-2}$$

where π_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ define the prior, mean and covariance of Gaussian k, respectively. Based on the GMM, one can estimate a Gaussian distribution of the output variable $\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{a}$ given an input $\boldsymbol{\xi}^{\mathcal{I}} = \boldsymbol{s}$ using GMR. As described in appendix A-2, this is done by calculating the conditional probability of $\boldsymbol{\xi}^{\mathcal{O}}$ given $\boldsymbol{\xi}^{\mathcal{I}}$ for each Gaussian separately. Then the different Gaussians are mixed according to a weight h_k. Hereby assuming that the mixed distribution is Gaussian with the mean and covariance defined as:

$$\hat{\boldsymbol{\xi}^{\mathcal{O}}}(\boldsymbol{\xi}^{\mathcal{I}}) = \sum_{k=1}^{K} h_k(\boldsymbol{\xi}^{\mathcal{I}}) \left(\boldsymbol{\mu}_k^{\mathcal{O}} + \boldsymbol{\Sigma}_k^{\mathcal{OI}} \left(\boldsymbol{\Sigma}^{\mathcal{II}} \right)^{-1} \left(\boldsymbol{\xi}^{\mathcal{I}} - \boldsymbol{\mu}^{\mathcal{I}} \right) \right), \tag{2-3}$$

$$\hat{\boldsymbol{\Sigma}}(\boldsymbol{\xi}^{\mathcal{I}}) = \sum_{k=1}^{K} h_k^2(\boldsymbol{\xi}^{\mathcal{I}}) \left(\boldsymbol{\Sigma}_k^{\mathcal{OO}} - \boldsymbol{\Sigma}_k^{\mathcal{OI}} \left(\boldsymbol{\Sigma}_k^{\mathcal{II}} \right)^{-1} \boldsymbol{\Sigma}_k^{\mathcal{IO}} \right), \tag{2-4}$$

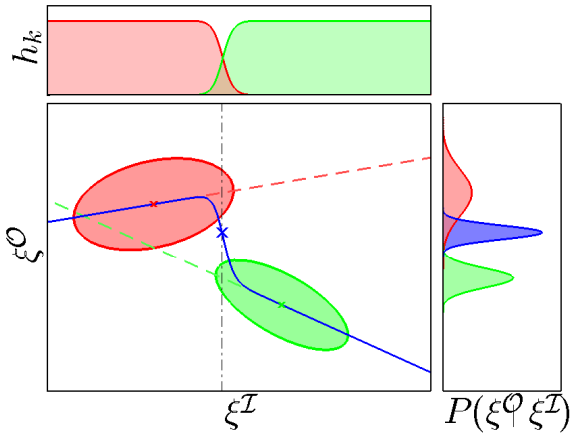


Figure 2-3: Regression of a GMM by means of GMR. For each point on the $\boldsymbol{\xi}^{\mathcal{I}}$ axis, the conditional probability $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{O}}|\boldsymbol{\xi}^{\mathcal{I}})$ is calculated for each Gaussian of the GMM. The results are mixed according to mixing coefficient h_k to estimate the mean and covariance of the regressed distribution at $\boldsymbol{\xi}^{\mathcal{I}}$ (displayed at the right). The blue line represents the mean, and is the regressed trajectory.

respectively. The weight h_k can be defined as:

$$h_k(\boldsymbol{\xi}^{\mathcal{I}}) = \frac{\pi_k \mathcal{N}(\boldsymbol{\xi}^{\mathcal{I}}|\boldsymbol{\mu}_k^{\mathcal{I}}, \boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}})}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\boldsymbol{\xi}^{\mathcal{I}}|\boldsymbol{\mu}_i^{\mathcal{I}}, \boldsymbol{\Sigma}_i^{\mathcal{I}\mathcal{I}})}. \tag{2-5}$$

The GMR process is visualized in Figure 2-3.

An advantage of learning joint probability of the input and output variables is that one also encodes the synergies between the different variables. These synergies are encoded in the correlation coefficients of the covariance matrix. Having this information, allows switching input and output variables on the fly, and handle missing data due to, for example, sensor errors, while keeping reasonable performance.

Learning joint probability requires learning full covariance matrices. The number of parameters required to define a covariance matrix grows quadratically with the number of dimensions. This makes the approach less compact compared to spline representation. However, this correlation information can be very useful for real robotic applications, e.g. [21].

Dynamic Movement Primitives

A movement representation especially designed for robotics operating in continuous state-action spaces is Dynamic Movement Primitives (DMP) [17]. In DMP, the acceleration of the robot is controlled as if it was pulled from the current position towards a goal position by a virtual spring-damper system:

$$\tau\ddot{x} = k^{\mathcal{P}}(x_g - x) - k^v\dot{x} + f(t), \text{ with } f(t) = \sum_{i=1}^{K} \Phi_i(t) f_i. \tag{2-6}$$

Here $k^{\mathcal{P}}$ and k^v are the virtual stiffness and damping, respectively. The spring-damper system is perturbed by a forcing function $f(t)$ that enforces a certain motion onto the end-affector.

This forcing function is defined by a number of activation functions $\Phi(t)$ with corresponding force components f_i. Learning is achieved by adjusting force components of the forcing functions. The number and location of the activation functions in the state-space is predefined. Hence, the parameter vector $\boldsymbol{\theta}$ is formed by force components f_i. This method can be generalized and combined with GMM-GMR, resulting in Dynamical System GMR (DS-GMR) [22]. The main difference between DS-GMR and DMP is the fact that DS-GMR uses multiple attractor points along which the end-affector of the robot is pulled, while DMP only has one attractor point. Hence, when under perturbation, DS-GMR is better capable of following the desired trajectory and reach the final position than DMP. DMP, due to its open-loop forcing function, is not able to recover towards the desired trajectory after perturbation. DMP is only able to recover through the end point attractor.

2-1-2 Multi-resolution Policy Representation

So-far methods have been discussed that represent a motion at one single level of refinement. In this thesis we aim to develop a multi-resolution policy representation suitable for policy search methods discussed in Section 2-2-2. The underlying idea is to simplify the search space. An alternative way of calling this a multi-resolution approach would be a hierarchical approach. Each 'resolution' level would then be equal to a level of hierarchy.

In the field of RL where the complete state/action space is considered, various techniques are proposed that try to simplify the policy by using hierarchical approaches. Most of these methods are intended for Value function-based Reinforcement Learning [23].

In the field of policy search using Policy parameterization, hierarchical or multi-resolution approaches are not widely explored. Kormushev et al.[16] propose an 'Evolving Policy Parameterization' based on Splines. This policy starts using a spline representation using only few via-points. As the learning process evolves, more via points are added to increase resolution of the policy. The method is empirically verified using a trajectory learning task. The Evolving policy parameterization outperforms single resolution spline parameterization.

2-2 Learning Techniques

As stated in the introduction of this chapter, learning comes down to finding a policy $\pi : \boldsymbol{s} \mapsto \boldsymbol{a}$. When using parameterized policies, such as the one introduced in section 2-1, learning comes down to finding appropriate policy parameters $\boldsymbol{\theta}$.

In this section we focus on two learning techniques that enable robots to learn $\boldsymbol{\theta}$: Programming by Demonstration and Learning from Exploration.

2-2-1 Programming by Demonstration

In Programming by Demonstration (PbD), robots learn the policy by observing one or multiple demonstrations of the skill to be learned. The learning process consists of two steps.

First, the robot observes one or multiple demonstrations of a successful skill execution. The robot records the demonstration using vision, data from sensors attached to the demonstrator

or from their own sensors in case of tele-operation or kinesthetic teaching. Depending on the demonstration type, a mapping should be created that projects the demonstration to the robot's body. This mapping solves the *correspondence problem*; a mismatch between the physical appearance of the demonstrator and the physical appearance of the robot. Using this mapping the data can be projected to the robot joint or task space.

Second, the data obtained from the demonstration should be transformed into a policy. This policy represents a mapping between input and output variables, i.e. state and action, respectively. In most PbD applications, a mapping is made between time and joint position or end-affector position and orientation. An alternative approach is to create a dynamical system from the demonstration data, i.e. create a mapping between position and velocity.

Learning is achieved by tuning the policy parameters $\boldsymbol{\theta}$ such, that the policy most closely resembles the relation between chosen input and output variables given by the demonstration data. The algorithm used to find $\boldsymbol{\theta}$, depends on the type of policy that is chosen. For example, the parameters of a neural network can be learned using backwards propagation, the parameters of a DMP can be learned using Locally Weighted Regression [24], and the Parameters of a GMM can be learned using Expectation Maximization (EM) [25].

Using GMM-GMR in PbD offers several advantages. The statistical origin of this method provides a proper foundation to handle high variability along different demonstrations of the same skill. Since GMM is a well-studied probabilistic model, a wide variety of tools exist that can be useful for robotics. In [26], Lee et al. use the Bayesian Information Criterion to estimate the number of Gaussians required to encode the demonstration data effectively; this provides autonomous selection of an appropriate model parameterization. In [27], Calinon et al. use the Gaussian product to generalize a task by combining demonstration data observed from different frames of reference.

2-2-2 Learning from Exploration

PbD provides a method for fast approximation of a policy. However, this approximation is not always good enough to successfully reproduce skills. For skills in which successful execution depends heavily on the movement dynamics, it is hard to present the robot with a consistent set of demonstrations. Examples of such skills are the *ball-in-cup* task [15] and the *pancake-flipping* task [28]. When demonstration sets are generalized acceleration peaks that are critical for successful execution might be washed out due to the variance in the demonstration set.

Robots that use LfE, learn a skill by trying to maximize task performance. LfE is a semi-supervised learning technique. LfE is not supervised because, unlike PbD, the robot is not told *how* to execute a certain skill. Instead, the robot evaluates *how well* it performs the skill. The skill performance is formalized by means of a reward. The reward is most commonly a function of both state and action: $\mathrm{r}(\boldsymbol{s}, \boldsymbol{a})$. The reward can be assigned at each time-step and/or after one full execution cycle of the skill.

One can distinguish three classes of exploration algorithms namely: value-function methods, policy search methods and Actor-critic methods [29]. Value-function approaches model a relation between the state/action combinations and expected reward. The policy is derived from the value-function by selecting the action for which the value-function indicates the

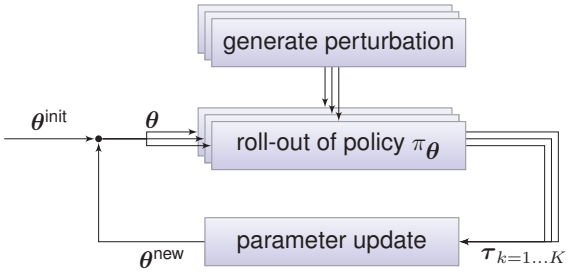


Figure 2-4: The policy improvement loop used for policy search. Perturbed versions of the initial policy parameters $\boldsymbol{\theta}^{init}$ are used to generate roll-outs. The resulting trajectories $\boldsymbol{\tau}_{k=1...K}$ are analyses to update the policy parameters. Adopted from [5].

highest expected return given a state. Value-function based approaches work with discrete state and actions. In discrete state-action spaces value-function approaches are guaranteed to converge to the global optimal solution. To scale value-function approaches to continuous state/action spaces, additional function approximators are required. No convergence proof exists for value-function approaches that use function approximators.

Policy search methods try to find the optimal policy by directly changing the policy without an additional value-function. Policy search methods scale well to continuous state/action spaces but are only guaranteed to converge to a *local* optimum.

Actor-Critic algorithms [30] split the process of optimizing and executing the policy in two pieces. The actor executes the current policy and the critic evaluates the performance of the actor by observing rewards. The critic consist of a value-function. It uses the same type of algorithms utilized by value-function approaches to update the value-function.

In this work we would like to initialize LfE based on demonstration data. The demonstration data provides an initial estimate of the policy. The initialization of the value-function from the initial policy is not possible without a model. This model is required to calculate the expected values for each state/action combination. In robotics a model free approach is desired because it is hard to obtain a good model of the environment and environments can change in some situations [10]. Therefore, we use policy search methods in this work.

Policy Improvement Loop

Policy search methods improve the policy by repetitive trials. The general policy improvement scheme utilized by policy search methods is displayed in Figure 2-4. Given the initial policy parameters $\boldsymbol{\theta}^{Init}$ a number of *roll-outs* are created. A roll-out is the execution of the policy. Each roll-out results in a sequence of states and actions together with a reward. This roll-out information is indicated by $\boldsymbol{\tau}_k$.

The roll-out is performed under perturbation. This perturbation is usually a zero mean Gaussian noise $\mathcal{N}(0, \boldsymbol{\Sigma})$ with covariance $\boldsymbol{\Sigma}^{Noise}$. Perturbations can be applied on the actions, i.e. $\tilde{a} = \pi(\boldsymbol{s}, \boldsymbol{\theta}) + \mathcal{N}(0, \boldsymbol{\Sigma})$, or perturbations can be applied on policy parameters, i.e. $\boldsymbol{a} = \pi(\boldsymbol{s}, \tilde{\boldsymbol{\theta}})$ with $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \mathcal{N}(0, \boldsymbol{\Sigma})$. Several disadvantages for using action perturbation in robotics are reported: (i) although exploration can lead to more successful roll-outs from time-to-time, it is hard to improve the policy due to the high variance in the gradient estimation. In

other words, it is hard to determine new policy parameters from roll-outs because it is not clear which actions were responsible for performance improvement [31]. (ii) The dynamics of the robotic system act as a low-pass filter and 'mask' the high-frequent components of the perturbations. (iii) Consecutive action perturbations may cancel each other and thus are washed out [5].

After performing one or multiple roll-outs a parameter update is calculated. Multiple options for this calculation exist. Early update algorithms were based on gradient estimation; for example Finite Difference-methods and REINFORCE [19]. More recently, algorithms based on reward weighted averaging are widely applied. Examples of these algorithms are PoWER [32], CMA-ES [33], CEM [34] and PI^2 [35]. These algorithms use a weighting mechanism, that weights the perturbed policy parameters $\tilde{\boldsymbol{\theta}}_i$ with their return to calculate a parameter update:

$$\boldsymbol{\theta}^{new} = \sum_i \tilde{\boldsymbol{\theta}} w_i \ , \quad \text{with} \ \sum_i w_i = 1 \tag{2-7}$$

State-of-the-art algorithms also adjust the exploration noise $\boldsymbol{\Sigma}^{Noise}$ during the update step. By adjusting exploration noise, the algorithm shapes the sampling distribution in such a way that samples are drawn in the direction in which the highest return is expected.

Reinforcement Learning and Black Box Optimization

Within policy search we distinguish two types of algorithms; Black Box Optimization (BBO) and Reinforcement Learning (RL). The difference between BBO and RL lies in the amount of information that is available to calculate a parameter update. BBO assumes that the process of creating a roll-out is a Black Box. This Black Box accepts the policy parameters $\boldsymbol{\theta}$ and results in a single scalar reward. RL uses the complete set of roll-out information $\boldsymbol{\tau}$ to calculate the parameter update. RL algorithms assume that the process that generated the roll-out information is a Markov Process. This means that the transition from state s_t to s_{t+1} only depends on action a_t. In other words, one assumes that all information required to estimate s_{t+1} is given by s_t and a_t.

Using BBO influences how the parameters can be perturbed. For BBO it is useless to perform a perturbation at every time step. These type of perturbations are performed inside the Black Box, thus the BBO algorithm is unaware of these perturbations. When using perturbation at each time step inside the Black Box, inserting the same set of policy parameters multiple times will result in different returns. Without information about state/action-reward relations it not possible to calculate a sensible parameter update that results in a higher expected return.

Togelius et al. [36] state that BBO algorithms generally outperform RL algorithms in continuous state spaces. Togelius et al. do not specify the metric of performance, but generally this is based on speed of convergence and the convergence value. It might seem counter intuitive that BBO can outperform RL because BBO has less information available. Rueckstiess et al. [31] argue that parameter perturbation at each time-step, which is used in RL, leads to a high variance in calculated parameter updates. This high variance leads to slower convergence of RL compared to BBO.

Stulp et al. [5] created RL and BBO algorithms that resulted in the same variance in parameter updates. By removing the difference in variance a better comparison could be made

between BBO and RL. They showed that the difference in performance was not due to the superiority of BBO, but because BBO is based on Reward Weighted averaging. In contrast, until recently most RL methods generally used gradient estimation. When BBO and RL algorithms both use reward weighted averaging, the difference in performance is much smaller. But still BBO seems to outperform RL in multiple empirical evaluations. Stulp et al. leave the explanation of this difference as an open research question.

2-3 Summary

Robot learning enables robots to obtain new policies without old-fashion programming skills. Instead of pre-programming a policy into a robot, the robot is able to obtain the policy from human demonstration or by exploration (self-teaching). Programming by Demonstration (PbD) allows fast learning of a policy because a human actively shows the robot how to perform a skill. Learning from Exploration (LfE) enables robots to improve their policy, modify their policy to unseen context or learn a policy outside human-capabilities.

Learning in robotics comes down to finding an appropriate mapping between states and actions. This mapping can be learned by PbD, LfE or a combination of both. Recent advances in the field of policy search combined with PbD have shown promising results on real robotic applications. In this field of learning, the state-action space of a robot is represented by a Model-based policy of the form $\boldsymbol{a} = \pi(\boldsymbol{s}, \boldsymbol{\theta})$. The policy space is represented by the policy parameters $\boldsymbol{\theta}$. These parameters can be learned using PbD and LfE.

In PbD, robots learn new skills by actively observing demonstrations and generalizing them into a policy. In LfE, robots learn new skills using a trial-and-error approach. The learning cycle is as follows: Given an initial policy, the robot makes small perturbations of this policy and creates roll-outs (policy executions). Each roll-out results in a reward or a reward sequence. The LfE algorithm updates the policy parameters in such a way that the expectation of the new policy receiving high rewards is maximized. Different update algorithms exist to achieve this. State-of-the-art algorithms are: PoWER[32], PI^2 [35], CMA-ES [33] and CEM [34]. One can distinguish two types of algorithms within LfE, namely: Black Box Optimization (BBO) and Reinforcement Learning (RL). RL uses the complete roll-out history (all state/action transitions and rewards) to calculate a new policy. In contrast, BBO only uses a single reward to calculate a new policy. Hence, BBO does not rely on the Markov assumption.

Different types of model-based policies have been used in combination with robotic applications: Neural-networks, Spline-based representations, Dynamic Movement Primitives (DMP) and GMM-GMM.

The use of multi-resolution or hierarchical policy representations has not yet widely been explored. The use of multi-resolution methods could be advantageous to achieve faster learning performance and thus provides an interesting research topic. In addition, the use of GMM-GMR have shown to be useful in robotic applications. In PbD it has proven to be useful in both generalization and reproduction. Furthermore, being well understood and documented, GMM provides interesting tools that can be useful for future applications.

Chapter 3

The Tree-model

In this chapter a novel movement representation is introduced based on GMM-GMR. The method is referred to as the Tree-based movement representation, or short; Tree-model. Inspiration for the method presented in this Chapter comes from Yamane et al. [37], who uses similar hierarchical Tree to store and recover motions from a movement database. However, instead of using the Tree-model only for movement recognition purposes, we use it for actual reproduction and improvement of movements.

As discussed in Section 2-1, GMM-GMR has several beneficial properties that are useful in robotics. This motivates the choice for using the GMM-GMR in this approach. However, the relatively large number of parameters to define a Gaussian Mixture Model (GMM) makes this method less suitable for exploration purposes. In the proposed method, we explore only a minimum part of the GMM parameter space, while keeping full control over the Gaussian Mixture Regression (GMR) process.

In the remainder of this introduction we present the global framework of our method. After this global introduction, the algorithm is discussed in more detail in the first two Sections of this Chapter. First, Section 3-1 describes how the Tree-model is constructed, and how the Tree-structure can be initialized based on demonstration data. Section 3-2 describes the proposed exploration algorithm and the GMM parameterization. This Chapter ends with a discussion on the proposed approach in Section 3-3.

The proposed method consists of two learning phases. In the first phase - the Programming by Demonstration (PbD) phase - the Tree-model is constructed based on demonstration data. In the second learning phase - the Learning from Exploration (LfE) phase - the performance of the Tree-model is optimized to meet a pre-defined performance criterion, i.e. to maximize the reward function.

The framework of the method is given by means of pseudo code in Algorithm 1. In the first learning phase (lines 2-7) the Tree-model, indicated by the object TM, is constructed based on data obtained by demonstration. The Tree-model is created bottom-up: starting at most coarse level and increasing resolution in each subsequent level. Alternatively, we could have decided to create the Tree top-down (starting at the most refined layer and then creating the

more coarse layers). However, using a bottom-up method seems more intuitive: first focus on the rough policy and then subsequently adding refinements. For simplicity, it is assumed that the number of Gaussians in the first layer, $K_{l=1}$ and the number of Layers L are given to the algorithm as prior knowledge. Future work could focus on learning these values autonomously.

A distinction is made between creating the bottom layer (Line 2), i.e. the layer with lowest resolution, and the subsequent child layers (Line 4-6). This is caused by the fact that the child layers are based on previous layers, while the first layer does not have a previous layer.

In the second phase an exploration algorithm is used to improve the Tree-based policy (Lines 9-17). This is done by starting exploration at the bottom layer, i.e. the layer with lowest resolution, and, after convergence, continuing by exploring the subsequent layers. By using the bottom-up approach, first the global policy is optimized in a low dimensional parameter space and subsequently more refinement can be achieved in the finer layers.

par_{expl} contain all the exploration settings, these are discussed in detail in Section 3-2-5.

Algorithm 1 Pseudocode describing the complete Tree-model framework.

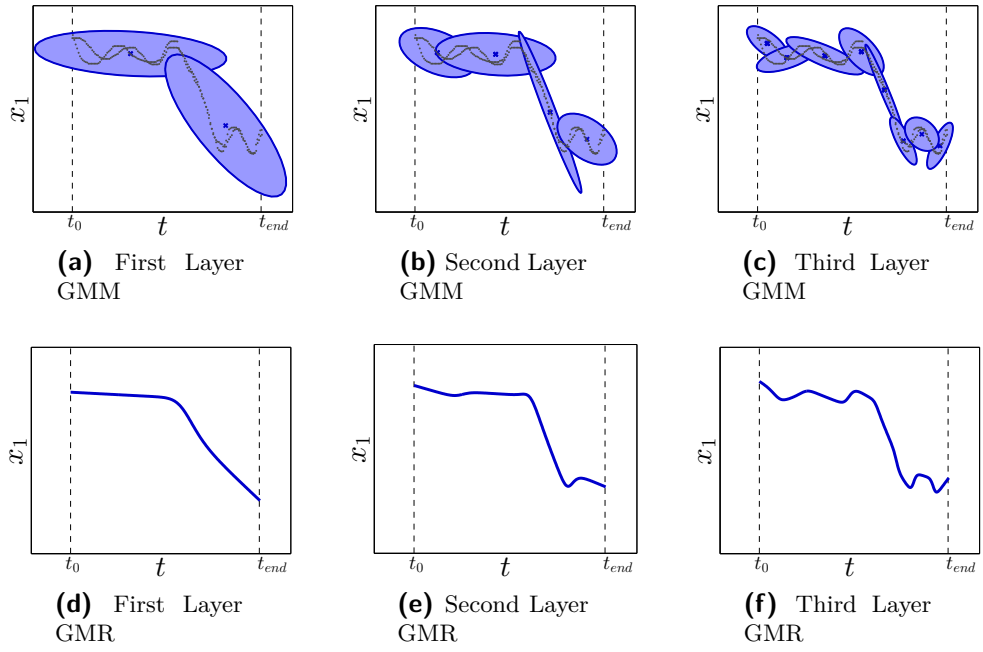
Require: $DemData$, L, $K_{l=1}$, par_{LfE}

1: //Phase I: Creating the Tree-model
2: TM . CreateBottomLayer($DemData, K_{l=1}$)
3:
4: **for** $i = 2$ to L **do**
5: TM . addChildLayer ($DemData$)
6: **end for**
7:
8: //Phase II: Optimization of Tree-model
9: **for** $i = 1$ to L **do**
10: $\boldsymbol{\theta}^i$ = TM . getTheta(i)
11: **repeat**
12: $\tilde{\boldsymbol{\theta}}^i$ = perturbParameter($\boldsymbol{\theta}^i, par_{LfE}$)
13: r = RollOut(TM, $\boldsymbol{\theta}^i$)
14: $\boldsymbol{\theta}^i$ = Update($\tilde{\boldsymbol{\theta}}^i, r$)
15: **until** Convergence
16: TM . updateTree($i, \boldsymbol{\theta}^i$)
17: **end for**

3-1 Construction of Tree

The objective of the Tree-model is to represent a movement at different, discrete levels of refinement. In this section the construction of the Tree-model, and its initialization is formalized.

This section starts with an example of the construction of the Tree-model. This example is intended to give the reader a global overview of the method. Afterwards, a more detailed description of the different steps involved is given.



(a) First Layer GMM **(b)** Second Layer GMM **(c)** Third Layer GMM

(d) First Layer GMR **(e)** Second Layer GMR **(f)** Third Layer GMR

Figure 3-1: An example of a Tree-based movement representation. Figure 3-1a - Figure 3-1c show how the data points - displayed in grey - are encoded in GMM of 2, 4 and 8 Gaussians, respectively. Figure 3-1d - Figure 3-1f displays the GMR result of the three different layers.

Figure 3-1 shows a 3-layer Tree-model. The model is fitted on a data set obtained from demonstration. The data set consists of two demonstrations of a 1-dimensional motion in time. The first layer is formed by fitting two Gaussians on the demonstration data. Thus, Layer 1 consist of a GMM with $K_1 = 2$ Gaussians. The creation of the first layer consists of two steps. First the parameters of the Gaussians, $\{\pi_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^{K_1}$, are initialized using an initialization algorithm, this could for example be done by a K-means algorithm. Then parameters of the GMM are learned using Expectation Maximization (EM) [38]. A good initial guess is required because EM is not guaranteed to converge to the global optimum. By combining EM with K-means, the means of the Gaussians are already placed at a good starting point. The GMM of layer 1 is displayed in Figure 3-1a.

Using GMR, the GMM can be regressed into a mapping from time to position. The regressed motion at layer l is depicted in Figure 3-1d. The regressed trajectory shows a coarse estimation of the demonstration data.

To increase the resolution of the model, a second layer is created. Each Gaussian from the first layer produces two child Gaussians by splitting itself in two new Gaussians. The second layer thus consist of a GMM with $K_2 = 4$ Gaussians.

The process of creating child Gaussians at layer l from a parent Gaussian in layer $l-1$ consists of three steps:

1. Create a local frame of reference based on the principal axis of the parent.
2. Split the parent Gaussian in two new child Gaussians.

3. Fit the child Gaussians on the data represented by their parent Gaussian.

We choose to use local frames of reference since this allows us to easily store and maintain parent-child relations as will be shown later on. This work focuses on investigating the advantages of a multi-resolution policy in LfE. Finding the optimal number of Gaussians after splitting is left as a challenge for later work. For simplicity, we assume that each Gaussian is split into two child Gaussians.

The GMM and the GMR results of the second layer, expressed in the global frame of reference, are shown in Figure 3-1b and Figure 3-1e, respectively. The trajectory obtained in the second layer is a better approximation of the demonstration data. In a similar fashion the third layer is created. The GMM and GMR results are displayed in Figure 3-1c and Figure 3-1f respectively. The regression result obtained from the third layer again shows a better fit of the demonstration data.

The remainder of this Section will explain the different functions used to construct the Tree-model. Section 3-1-1 describes how one can define a new frame of reference based on a parent Gaussian. The Split algorithm that splits a Parent Gaussian in two Child Gaussians is discussed in Section 3-1-2. The adapted EM-algorithm, used to fit the child Gaussians on the demonstration data, is discussed in Section 3-1-3.

3-1-1 Defining a Frame of Reference Based on a Gaussian

The Tree-model representation requires a formalization of the relation between child and parent in the policy space. This relation should be in such a way that any changes to the location or orientation of the parent Gaussian affect the children but not vice versa. To this end, location and orientation of child Gaussians are described with respect to their parent.

A Gaussian is defined by a covariance $\boldsymbol{\Sigma}$, and a mean $\boldsymbol{\mu}$ expressed in a frame of reference Ψ^g. Figure 3-2a shows a multivariate Gaussian representing the joint probability of two random variables. The covariance is visualized by an ellipse that indicates a line of equal probability. A Covariance matrix can be decomposed in its principal components:

$$\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T, \tag{3-1}$$

where $\boldsymbol{V}$ is a matrix of eigenvectors, and $\boldsymbol{\Lambda}$ a diagonal matrix of which the diagonal entries correspond to the eigenvalues λ_i. The eigenvectors of $\boldsymbol{\Sigma}$ are shown in Figure 3-2b and represent the mirror axis of the symmetric Covariance. The eigenvectors can be used to define a new frame of reference Ψ^l as shown in Figure 3-2c.

The relation between two frames of reference Ψ^j and Ψ^i is composed of a translation of origin $\boldsymbol{o} \in \mathbb{R}^{d\times 1}$ combined with a rotation $\boldsymbol{R} \in \mathrm{SO}(d)$. The transformation of a point $\boldsymbol{p}$ described in the frame of reference Ψ^j to the frame of reference Ψ^i can be described by an affine transformation matrix $\boldsymbol{H}^i_j$ that is defined as:

$$\boldsymbol{H}^i_j = \begin{bmatrix} \boldsymbol{R}^i_j & \boldsymbol{o}^i_j \\ 0 & 1 \end{bmatrix} \tag{3-2}$$

Here $\boldsymbol{R}^i_j$ represents the rotation of Ψ^j with respect to Ψ^i and $\boldsymbol{o}^i_j$ represents the origin of frame Ψ^j expressed in frame Ψ^i. When using the properties of a Gaussian, the translation

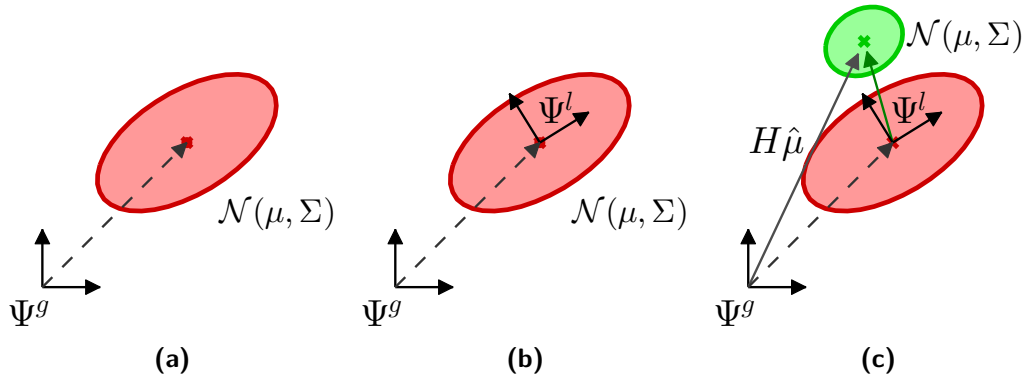


Figure 3-2: (a) A Gaussian defined in a frame of reference. (b) The principal components of a Gaussian can be used to define a local frame of reference. (c) The local frame of reference can be used to define a new GMM. The relation with the global frame of reference can be described by means of a Homogeneous matrix H.

and rotation corresponds to $\boldsymbol{\mu}$ and $\boldsymbol{V}$, respectively. Projection of the point $\boldsymbol{p}^j$ expressed in frame j to frame i is achieved by:

$$\hat{\boldsymbol{p}}^i = \boldsymbol{H}^i_j \hat{\boldsymbol{p}}^j \ , \quad \text{with } \hat{\boldsymbol{p}} = \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} \tag{3-3}$$

The concept of expressing Gaussians in a frame of reference is used to formalize the Tree-structure. Each layer l consists of a number of frames of reference which are uniquely defined by $\Psi^{l,p}_i$. i indicates the number of the frame of reference in layer l. p indicates the parent frame of reference which, by convention, is always defined in layer $l-1$.

Figure 3-3 gives an example of the notation for a 3-layer Tree-model. The first layer has only one frame of reference $\Psi^{1,0}_1$ and has no parent. The frame of reference used to define the first layer is the global frame of reference, i.e. $\Psi^g := \Psi^{1,0}_1$.

The global frame of reference is the space in which the policy is executed. For all subsequent layers $l > 1$, a local frame of reference is defined for each Gaussian defined in layer $l-1$. Note that layers $l > 1$ thus have multiple local frames of reference.

The policy required for execution should be expressed in the global frame of reference. To project the GMM defined in layer l onto the global frame of reference, all Gaussians must be projected back from their local frame of reference to the global frame of reference. This projection is defined as:

$$\hat{\boldsymbol{\mu}}^g = \boldsymbol{H}^g_j \hat{\boldsymbol{\mu}}^j \ , \quad \boldsymbol{\Sigma}^g = \boldsymbol{R}^g_j \boldsymbol{\Sigma}^j \left(\boldsymbol{R}^g_j\right)^T \tag{3-4}$$

with

$$\boldsymbol{H}^g_j = \boldsymbol{H}^g_i \cdots \boldsymbol{H}^i_j. \tag{3-5}$$

3-1-2 Gaussian Split Algorithm

When creating a layer $l+1$ from layer l, the Gaussians in the previous layer are split. In this work it is assumed that each Gaussian is split in two components. Splitting a Gaussian is

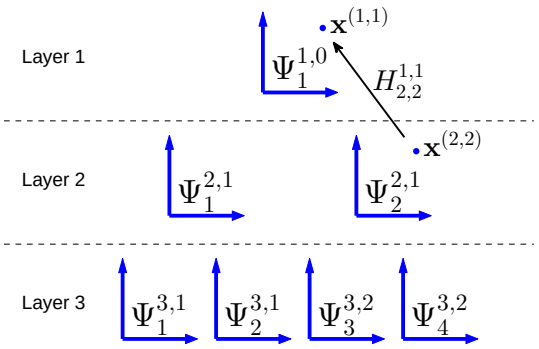


Figure 3-3: Visualisation of the Notation used to formalize the Tree-model. $\Psi_i^{l,p}$ is the i-th frame of reference in layer l with a parent with index p in layer $l-1$.

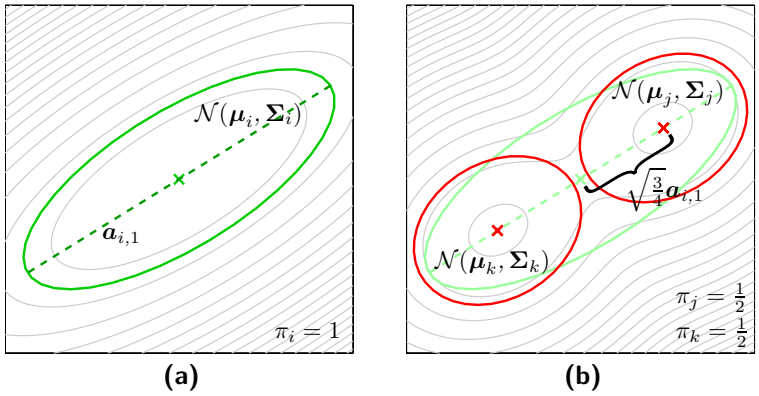


Figure 3-4: A Gaussian being split in two Gaussians. Illustration obtained from [6].

an ill-posed problem since it has more unknowns than equations. Zhang et al.[39] proposed a split-and-merge method based on Singular Value Decomposition (SVD). We use this split algorithm to initialize the Child Gaussian before using the adapted EM-algorithm (discussed in Section 3-1-3).

The algorithm works as follows: Given $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, one can obtain $\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ and $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ of which the priors, mean and Covariance are given by:

$$\begin{aligned}
&\pi_j = \pi_i\alpha, && \pi_k = \pi_i\alpha,\\
&\boldsymbol{\mu}_j = \boldsymbol{\mu}_i - \sqrt{\frac{\pi_k}{\pi_j}}u\boldsymbol{a}_{i,l}, && \boldsymbol{\mu}_k = \boldsymbol{\mu}_i - \sqrt{\frac{\pi_j}{\pi_k}}u\boldsymbol{a}_{i,l},\\
&\boldsymbol{\Sigma}_j = \frac{\pi_k}{\pi_j}\boldsymbol{\Sigma}_j + \left(\beta - \beta u^2 - 1\right)\frac{\pi_i}{\pi_j}\boldsymbol{a}_{i,l}\boldsymbol{a}_{i,l}^T + \boldsymbol{a}_{i,l}\boldsymbol{a}_{i,l}^T, && \boldsymbol{\Sigma}_k = \frac{\pi_j}{\pi_k}\boldsymbol{\Sigma} + \left(\beta - \beta u^2 - u^2\right)\frac{\pi_i}{\pi_k}\boldsymbol{a}_{i,l}\boldsymbol{a}_{i,l}^T + \boldsymbol{a}_{i,l}\boldsymbol{a}_{i,l}^T.
\end{aligned} \tag{3-6}$$

$\boldsymbol{\Sigma}_i = \boldsymbol{A}_i\boldsymbol{A}_i^T$ represents the ordered eigencomponents decomposition with $\boldsymbol{A}_i = [\boldsymbol{a}_{i,1}, \boldsymbol{a}_{i,2}, \ldots, \boldsymbol{a}_{i,d}]$. The split is controlled by the parameters $\alpha \in (0,1)$, $\beta \in (0,1)$, $u \in (0,1)$ and $l \in \{1,2,\ldots,d\}$. By setting $\alpha = \beta = \frac{1}{2}$ the split Gaussians have the same 'shape' and prior. u controls the

'Classical' Expectation Maximization	Adapted Expectation Maximization
E-step:	E-step:
$h_{n,k} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})}.$	$h_{n,k} = h_{n,p} \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})}.$
M-step:	M-step:
$\pi_k = \frac{\sum_{n=1}^{N} h_{n,i}}{N},$	$\pi_k = \frac{\sum_{n=1}^{N} h_{n,i}}{N},$
$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} h_{n,k} \boldsymbol{x}_n}{\sum_{n=1}^{N} h_{n,k}}$	$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} h_{n,k} \boldsymbol{x}_n}{\sum_{n=1}^{N} h_{n,k}}$
$\boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^{N} h_{n,k} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} h_{n,k}}$	$\boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^{N} h_{n,k} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} h_{n,k}}$

Table 3-1: The classical EM-algorithm (left) is adapted to enable a soft-clustering approach required for the Tree-Model. The adapted EM-algorithm (right) has an extra weight $h_{n,p}$ in the Expectation step. This weight represents the parent Gaussian responsibility $\mathcal{P}(p|\boldsymbol{x}_n)$.

placement of the means along the selected principal axis l. By setting $l = 1$ the Gaussian is split along the axis on which the demonstration data has the largest variance.

3-1-3 Adapted EM-algorithm

For each layer the parent Gaussian is split in two child Gaussians. After the split, EM is required to fit the child Gaussians on the demonstration data. Standard EM considers all data points as equally important. However, the child Gaussians should increase the level of detail of the demonstration data covered by their parent. This type of clustering is not taken into account in the standard EM-algorithm.

To take into account the parent-child relation, the Expectation step of the EM-algorithm is modified. The Expectation step calculates the probability of Gaussian k being responsible for data point $\boldsymbol{x}_n$, i.e. the responsibility $h_{n,k} = \mathcal{P}(k|\boldsymbol{x}_n)$. A child Gaussian should model the data represented by the parent with an increased level of detail. Therefore, EM for the child Gaussians should focus on data that lies near the parent Gaussian p, i.e. data with a high responsibility $\mathcal{P}(p|\boldsymbol{x}_n)$. Higher weight is achieved by adding the parent responsibility to the expectation step as shown in Table 3-1.

3-2 Exploring the Tree

In this section the exploration algorithm used to explore the Tree-model is presented. After creating the Tree-model we try to exploit its structure during exploration. The method relies on the hypothesis that search in a parameter space of low dimensionality is faster than search in a parameter space of high dimensionality. The aim is to first learn the rough policy in a relatively low dimensional space and subsequently learn refinements in higher dimensional space.

First, the general Tree-model exploration algorithm is introduced to give the reader a clear view of the global concept. Afterward, the different elements of the exploration algorithm are treated in detail.

Algorithm 2 shows pseudo-code of the Tree-model exploration algorithm. The general concept is equal to the policy improvement loop introduced in Section 2-2-2. The policy improvement loop is contained inside a for-loop that iterates through the different layers of the Tree-model. Thus, instead of performing all roll-outs in one layer, the exploration algorithm starts at the coarsest layer and subsequently optimizes the more refined layers until convergence. To determine when to switch from layer l to $l+1$, a convergence criterion is required.

Optimization of one layer is done as follows. First the parameters vector $\boldsymbol{\theta}^l$ at layer l is constructed (line 2) and $\boldsymbol{\Sigma}_{noise}$ is initialized (line 3). $\boldsymbol{\theta}^l$ does not contain all the parameters of the GMM, but only parameters that influence the GMR process. This will be discussed in more detail in Section 3-2-2. Then the algorithm enters the policy improvement loop (lines 4 - 10). For each parameter update nbE perturbations and roll-outs are created (lines $6-7$). The perturbation is created by adding Gaussian noise to the current policy parameters. Then a roll-out is made using the perturbed parameters. When nbE roll-outs are created, a parameter update is calculated using an update algorithm. After a layer has converged, the Tree-model is updated according to the obtained parameter vector $\boldsymbol{\theta}^l$ (line 11).

Exploration is done in global frame of reference, i.e. $\boldsymbol{\theta}^l$ is obtained from the GMM at layer l in the global frame of reference. By doing this, it becomes more intuitive to specify the noise since the noise can then be set in the same space as the policy is executed. As a consequence, the values of $\boldsymbol{\theta}^i$ $i > l$ will change when exploring layer l because the child Gaussians of a parent will move in the global reference frame when their parents' position is changed.

3-2-1 Convergence criteria

The convergence detection is assumed to be an important aspect in the Tree-model exploration. When convergence is detected too early, the algorithm will not optimally use the advantage of searching a parameter space of lower complexity. Detecting convergence too late will result in consuming time-expensive roll-outs without gaining much performance.

For the exploration algorithm online convergence detection is needed, i.e. convergence should be detected while the algorithm is running. Convergence detection is for example studied in the field of Multi Objective Evolutionary Algorithms [40].

We propose convergence detection based on Least Squares regression of the reward obtained, similar to [40] and [41]. By analyzing the reward history, one can fit a linear line $y = \alpha x + \beta$.

Algorithm 2 Optimization of Tree-model

```
for l = 1 to L do
    θ^l = TM.getTheta(l)
    Init Σ_noise
    repeat
        for i = 1 to nbE do
            θ̃_i^l = θ^l + N(0, Σ_noise)
            r_i = rollOut(TM, θ^i)
        end for
        [θ, Σ_noise] = update(θ̃^i, r_i)
    until Convergence
    TM.updateTree(i, θ^i)
end for
```

Given r_i as the reward obtained at roll-out i the regression is described as:

$$\underbrace{\begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix}}_{\boldsymbol{r}} = \underbrace{\begin{bmatrix} 1 & 1 \\ \vdots & \vdots \\ N & 1 \end{bmatrix}}_{\boldsymbol{A}} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{3-7}$$

The slope α is obtained by performing the operation $\boldsymbol{A}^{\dagger}\boldsymbol{r}$ where $(\cdot)^{\dagger}$ is Moore-Penrose pseudo inverse. The slope α indicates the increase in reward over the N samples and can be used as a convergence criteria. When $\alpha < \epsilon_{conv}$ it is assumed that the algorithm has converged.

However, the regression must be performed locally instead of over the complete reward history. Therefore, the rewards are regressed within a floating 'reward-window' which will represent the last M results. When M is low, the regression process is sensitive to reward drops inherent to the stochastic exploration process leading to pre-mature convergence. Setting M too large will result in late detection of convergence.

3-2-2 Tree-model parameterization

State of the art exploration algorithms such as PoWER[32] and PI^2 [35], use linear combinations of the policy parameters to calculate the parameter update:

$$\boldsymbol{\theta}^{new} = \sum_{i}^{n} w_i \tilde{\boldsymbol{\theta}}_i, \tag{3-8}$$

To use GMM in such a setting, the parameters of the GMM, $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$, should be transformed in a vector. A function is required to transform the parameters defining the GMM, into the parameter vector $\boldsymbol{\theta}$:

$$\{\boldsymbol{\theta}, \boldsymbol{\phi}\} = \mathrm{f}(\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}). \tag{3-9}$$

Besides the policy parameters $\boldsymbol{\theta}$, a set of parameters $\boldsymbol{\phi}$ is generated that are not considered during exploration. These parameters are required to reconstruct the covariance matrices of

the GMM. The parameterization of the covariance matrix will be discussed in more detail later on in this section.

The generation of roll-outs (line 7 in Algorithm 2) consists of the reconstruction of a temporary GMM based on the perturbed parameters $\tilde{\boldsymbol{\theta}}_i^l$ and the 'static' parameters $\boldsymbol{\phi}$. Afterwards, the GMM is evaluated by generating a reproduction of the policy and observing the reward function of this reproduction.

The design of function f$(\cdot)$ has focused on two aspects: (i) minimizing the number of parameters $\boldsymbol{\theta}$, (ii) finding an unconstrained parameterization of $\boldsymbol{\Sigma}$.

Reduction of dimensionality

To keep the search space as simple as possible, it is desirable to keep the dimensionality of $\boldsymbol{\theta}$ low. A K state GMM is defined by the parameters $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ with $k = 1 \dots K$. The number of parameters that uniquely define the GMM is given by:

$$K\left(1 + \frac{d(d+3)}{2}\right), \tag{3-10}$$

where d is dimensionality of the random variable encoded by the GMM. Compared to other policy representations like for example Dynamic Movement Primitives (DMP) [24], this number of parameters is large.

The number of parameters required to define a GMM is relatively large because a GMM encodes the joint Probability Density Function (pdf) of input and output variables. In contrast, other policy representations encode only a mapping between input and output variables. The covariance matrix, required to define the joint pdf, is responsible for $\frac{d(d+1)}{2}$ parameters.

By exploiting the knowledge about the GMR process, the number of parameters can be reduced. It is shown in Appendix A, that the trajectory generated by GMR can be completely controlled by only changing the matrix describing the correlation between input and output elements $\boldsymbol{\Sigma}^{\mathcal{I}\mathcal{O}}$, and the mean $\boldsymbol{\mu}$. Hence, the number of parameters required for exploration can be reduced to $K\left(d^{\mathcal{O}}\left(2d^{\mathcal{I}}+1\right)\right)$ where $d^{\mathcal{I}}$ and $d^{\mathcal{O}}$ indicate the dimensionality of the input and output variables, respectively. This approach does not optimize the joint pdf but only the elements that influences the GMR process for given input and output parameters.

Following the derivation described in Appendix A-3, the parameterization of a GMM is given by:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \vdots \\ \boldsymbol{\theta}_K \end{bmatrix}, \quad \text{with } \boldsymbol{\theta}_k = \begin{bmatrix} \boldsymbol{\theta}_{\boldsymbol{\mu}} \\ \boldsymbol{\theta}_{\Sigma^{\mathcal{O}\mathcal{I}}} \end{bmatrix}, \; k = 1 \dots K \tag{3-11}$$

$\boldsymbol{\theta}_{\boldsymbol{\mu}} = \boldsymbol{\mu}_k$ are the parameters relating to the means of the k-th Gaussian. $\boldsymbol{\theta}_{\Sigma^{\mathcal{O}\mathcal{I}}}$ are the parameters relating to the covariance matrix of the k-th Gaussian. We assume that the Gaussians are well separated over the input space. Hence, exploration of the priors π_k can be omitted because they do not influence the regression process in this case.

Parameterization of Covariance

During exploration the covariance matrices of the GMM must remain positive definite. This requires a suitable parameterization of the covariance matrix. In Appendix A-4 multiple parameterizations of the covariance matrix are discussed. In this work we use the covariance parameterization based on the Cholesky decomposition [42]. A covariance matrix can be decomposed as:

$$\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T \tag{3-12}$$

With $\boldsymbol{L}$ as a lower triangular matrix. The non-zero elements of L form the parameterization of $\boldsymbol{\Sigma}$. This are $\frac{d(d-1)}{2}$ parameters.

When taking into account the fact that we use GMR, the list of parameters is split in two: namely a list of parameters that influence the regression result and a list of parameters which does not influence the regression result. Let Chol$(\cdot)$ be a function that converts a Covariance into the Cholesky parameterization:

$$\beta = \text{Chol}(\boldsymbol{\Sigma}) \tag{3-13}$$

β is the list of parameters describing the non-zero elements of the $\boldsymbol{L}$ matrix. β can be split into two groups; $\beta = \{\boldsymbol{\phi}, \boldsymbol{\theta}_{\Sigma^{\mathcal{IO}}}\}$. $\boldsymbol{\theta}_{\Sigma^{\mathcal{IO}}}$ are the parameters that are used during exploration. $\boldsymbol{\phi}$ are the remaining parameters required only for reconstruction of the covariance matrix.

3-2-3 Update Rule

As discussed in Section 2-2-2, a variety of update algorithms exists. Any of these algorithms can be combined with the proposed approach. In this work a Black Box version of the PoWER algorithm used in Calinon et al. [43] is used. This algorithm was selected because it was reported as well-performing and it is easy to understand and implement. The algorithm, like PoWER, uses reward weighted averaging. Similar to CEM or CMA-ES the exploration noise is recalculated with every update. In addition, it uses importance sampling to select only the best N samples from the history of roll-outs to calculate the parameter update. The algorithm is Black Box because it calculates the reward over the complete roll-out.

The parameter update is given by:

$$\boldsymbol{\theta} \leftarrow \frac{\sum_{i=1}^{I} \tilde{\boldsymbol{\theta}}_{w(i)} r_{w(i)}}{\sum_{i=1}^{I} r_{w(i)}} \tag{3-14}$$

And noise covariance update:

$$\Sigma^{noise} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\theta}}_{w(1)} & \dots & \tilde{\boldsymbol{\theta}}_{w(I)} \end{bmatrix} \begin{bmatrix} r_{w(1)} \\ \dots \\ r_{w(I)} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\theta}}_{w(1)} \\ \dots \\ \tilde{\boldsymbol{\theta}}_{w(I)} \end{bmatrix} + \Sigma^{min} \tag{3-15}$$

The function $w(i)$ outputs the indices of the perturbed parameters by descending reward. $w(1)$ will output the index of the perturbed parameter with the highest reward and $w(2)$ will output the index of the perturbed parameters with the second highest reward. The variable I indicates the number of $\boldsymbol{\theta}$ considered for a parameter update.

Σ^{min} is the minimum exploration noise. This exploration noise prevents premature convergence to bad local optima. In addition, the minimum exploration can keep the exploration noise full rank. Keeping the matrix full-rank is required when one uses a number of samples I which is less than the dimension of $\boldsymbol{\theta}$. The minimum exploration noise is set as a fraction of the initial exploration noise, i.e. $\Sigma^{min} = \Sigma^{Noise}_{init}\epsilon_{Noise}$.

3-2-4 Reusing Experience

The exploration noise determines the direction in which the exploration algorithm explores for better solutions. When the number for roll-outs increases, the exploration noise is optimized, i.e. the exploration noise contains 'experience' or knowledge about the local solution space. When switching from layer l to $l+1$ the exploration noise should be reinitialized. It is not possible to use the exploration noise from layer l. The exploration noise at layer l is of dimension $n \times n$, while the exploration noise at layer $l+1$ is of dimension $m \times m$ with $m > n$. When initializing the exploration noise at layer $l+1$ with a pre-defined initial exploration noise, the algorithm would omit all 'experience' obtained so-far. In this section a method is proposed to re-use the experience from layer l to initialize the exploration noise at layer $l+1$.

For convenience the following notation for the Tree-model is adopted; TM $\left(\boldsymbol{\theta}^1_{init}, \dots, \boldsymbol{\theta}^L_{init}\right)$. Here $\boldsymbol{\theta}^l_{init}$ indicates the vector of parameters considered for exploration at layer l. The initial values of $\boldsymbol{\theta}^l$ follow from the GMM at layer l which was initialized using EM. The subscript $\cdot_{init}$ indicates that the parameters have not yet been explored and thus are in the initial state. When the subscript is dropped, the parameters at layer l have been explored.

Given the sets of parameters and corresponding reward $\{\tilde{\boldsymbol{\theta}}^l_i, r_i\}$ $i \in 1 \cdots RO$ with RO the number of roll-outs made so-far, the objective is to find $\tilde{\boldsymbol{\theta}}^{l+1}_i$ that yield to the same policy as $\tilde{\boldsymbol{\theta}}^l_i$; i.e. find $\tilde{\boldsymbol{\theta}}^{l+1}_i$ such that TM $\left(\boldsymbol{\theta}^1, \dots, \tilde{\boldsymbol{\theta}}^l_i, \boldsymbol{\theta}^{l+1}_{init}, \dots\right) \sim$ TM $\left(\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^l, \tilde{\boldsymbol{\theta}}^{l+1}_i, \dots\right)$. When parameters $\tilde{\boldsymbol{\theta}}^{l+1}_i$ are found, they can be combined with obtained rewards r_i to calculate the exploration noise using Eq. (3-15).

After layer l has converged, the algorithm calculates the final parameter update $\boldsymbol{\theta}^l$ and switches to $l+1$. The initial state of the Tree-model is TM $\left(\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^l_i, \boldsymbol{\theta}^{l+1}_{init}, \dots\right)$. The parameters $\boldsymbol{\theta}^{l+1}_{init}$ are obtained by calculating the GMM parameterization at layer $l+1$ as described in appendix A. This parameterization results in two parameter sets; $\boldsymbol{\theta}^{l+1}$ and $\boldsymbol{\phi}^{l+1}$. Both $\boldsymbol{\theta}^{l+1}$ and $\boldsymbol{\phi}^{\boldsymbol{l+1}}$ are required to recreate the GMM, but only $\boldsymbol{\theta}^{l+1}$ is changed during exploration.

The same operation can be executed for the perturbed parameters $\tilde{\boldsymbol{\theta}}^l_i$ at layer l to obtain $\tilde{\boldsymbol{\theta}}^{l+1}$ and $\tilde{\boldsymbol{\phi}}^{l+1}$. But $\tilde{\boldsymbol{\phi}}^{l+1} \neq \boldsymbol{\phi}^{l+1}$, i.e. combining $\boldsymbol{\phi}^{l+1}$ with $\tilde{\boldsymbol{\theta}}^{l+1}_i$ will not result in the same GMM as $\tilde{\boldsymbol{\theta}}^{l+1}_i$ and $\tilde{\boldsymbol{\phi}}^{l+1}$ describe. Instead, we would like to find different $\hat{\boldsymbol{\theta}}^{l+1}_i$, that, combined with $\boldsymbol{\phi}^{l+1}$, yield to the same regression result as the parameter set $\{\tilde{\boldsymbol{\theta}}^{l+1}_i, \tilde{\boldsymbol{\phi}}^{l+1}\}$, i.e.:

$$\sum_{k=1}^{K} \pi_k(\hat{A}_k x + \hat{b}_k) = \sum_{k=1}^{K} \pi_k(\tilde{A}_k x + \tilde{b}_k), \tag{3-16}$$

with $A = \boldsymbol{\Sigma}^{\mathcal{OI}}\left(\boldsymbol{\Sigma}^{\mathcal{II}}\right)^{-1}$ and $b = \boldsymbol{\mu}^{\mathcal{O}} + \boldsymbol{\Sigma}^{\mathcal{OI}}\boldsymbol{\mu}^{\mathcal{I}}$. $\boldsymbol{\theta}$ consist of the elements of $\boldsymbol{\Sigma}^{\mathcal{OI}}$ and $\mu^{\mathcal{O}}$ and π is not considered as a variable during exploration. Hence, the elements of $\hat{\boldsymbol{\theta}}^{l+1}_i$ are obtained

by subsequently solving:

$$\hat{\boldsymbol{\Sigma}}_k^{\mathcal{OI}} = \tilde{\boldsymbol{\Sigma}}_k \left(\tilde{\boldsymbol{\Sigma}}_k^{\mathcal{II}}\right)^{-1} \hat{\boldsymbol{\Sigma}}_k^{\mathcal{II}}, \tag{3-17}$$

$$\hat{\boldsymbol{\mu}}_k^{\mathcal{O}} = \tilde{\boldsymbol{\mu}}_k^{\mathcal{O}} + \tilde{\boldsymbol{\Sigma}}_k^{\mathcal{OI}} \tilde{\boldsymbol{\mu}}_k^{\mathcal{I}} - \hat{\boldsymbol{\Sigma}}_k^{\mathcal{OI}} \boldsymbol{\mu}_k^{\mathcal{I}}. \tag{3-18}$$

Note that the elements of $\tilde{\boldsymbol{\Sigma}}_k^{\mathcal{II}}$, $\tilde{\boldsymbol{\mu}}_k^{\mathcal{I}}$ and $\boldsymbol{\Sigma}_k^{\mathcal{II}}$,$\boldsymbol{\mu}_k^{\mathcal{I}}$ are contained in $\tilde{\boldsymbol{\phi}}_k$ and $\boldsymbol{\phi}_k$, respectively.

Initializing the Exploration noise for layer $l+1$ based on the parameter samples of layer l will lead to a matrix that is not full-rank. Exploring in l means exploring in a sub-space of $l+1$. When $\tilde{\boldsymbol{\theta}}_i^l$ is projected to layer $l+1$, the samples will only lie in the sub-space. To enable exploration in all directions of $l+1$ and to create a non-singular noise matrix, the exploration noise must be increased for the unexplored directions.

The reuse of rewards r_i can only be done if the performance of the Tree-model is always evaluated at the most refined layer. Evaluation of the Tree-model at Layer l will result in a different reward as the evaluation of the same Tree-model at layer $l+1$ because the GMM at layer l and $l+1$ do not represent exactly the same input-output mapping. When the reward r_i would be calculated based on the GMM at layer l, the proposed method is not able to reconstruct the GMM at layer $l+1$ such that it will yield to the same reward. Hence, the reward of the 'reconstruction' has changed and the 'experience' is not retained. Hence, i this work we will always evaluate the reward at the most refined layer.

3-2-5 Overview of Exploration Parameters

Symbol	Meaning
S_{init}	Number of roll-outs before first update
I	Number of roll-outs considered for one update
M	The number of roll-outs considered during convergence detection
ϵ_{conv}	Convergence threshold
$\boldsymbol{\Sigma}^{Noise}$	Initial Exploration noise
ϵ_{Noise}	Fraction of initial exploration noise used as minimum exploration noise

Table 3-2: Summary of Tree-model exploration algorithm parameters

The exploration algorithm has a number of open parameters. Table 3-2 gives an overview of the open parameters.

Before calculating the first parameter update, a number of roll-outs must be generated to provide the algorithm with sufficient information to calculate the first parameter update. This number of roll-outs created before the first update, S_{init}, should be larger than the number of roll-outs I considered for an update. After the first update, one extra roll-out is created per update. This is done because creating roll-outs is relatively time-consuming in robotic applications. By updating after each roll-out, the policy used to generate new perturbations is always based on the latest, and possibly best, information.

M represents the number of roll-out rewards considered for convergence detection as described in Section 3-2-1. When the regressed slope is less than ϵ_{conv}, convergence is assumed.

The algorithm requires an initial exploration noise to determine the amount of exploration for the different parameters. This information is given to the algorithm in the form of the exploration noise Σ_{Noise}. The exploration noise is a $n \times n$ matrix where n is the length of the parameter vector $\boldsymbol{\theta}$. Throughout this work the initial exploration noise is given equal for each Gaussian in the GMM. Furthermore, only the variance for the different parameters is, i.e. the initial noise matrix is a diagonal matrix where the correlations coefficients are all zero. The minimum exploration noise is taken to be a fraction ϵ_{Noise} of the initial exploration noise.

3-3 Discussion

In this chapter we introduced a Tree-based policy representation and exploration algorithm. In this last section we discuss the design decisions made.

During the construction of the Tree-model, a system of local frames of reference was proposed. The use of local frames has as an advantage that changes of parent location and/or orientation automatically propagate to child Gaussians without the need for any further computations. For reproduction, however, the GMM in the local reference frames must be projected back to the global frames. Alternatively, we could have decided to define all layers in the global reference frame. This would require performing operations to maintain the parent-child relations at all layers during exploration. Either way, the behavior is the same. Updating the Tree relations will require computation either during, or after exploration. It was decided to use the representation using local frames because it was found to be more intuitive.

The Gaussian Split-algorithm described in Section 3-1-2 follows from earlier work [6]. Being based on SVD, the split algorithm provides a method that fits well with the method used for local frames of reference. The Child Gaussians are always split along the axis which represents the largest variability. The split algorithm depends on a large number of parameters that define the split. It was proposed to keep these parameters constant. It is believed this is a valid assumption, since after split an EM-algorithm is used to optimize the position of the child Gaussians.

For non-root layers, an adapted EM-algorithm was proposed. This adaptation involves including the posterior probability $\mathcal{P}(k|\boldsymbol{x}_n)$ of the parent when calculating the expectation of the child. This adaptation enables refining the data of the parent Gaussian in more detail without the need of defining crisp clusters. As a consequence, this 'soft' constraint may result in less optimal solutions. In other words, fitting a 4 Gaussian single layer model on a data set, might result in a different solution than fitting a 2 layer Tree-model with 4 Gaussians in the final layer. As a result, the Tree-model might not be able to fully exploit the experience contained in the demonstration data.

The Tree-model is constructed in a bottom-up approach, i.e. first the rough layer is created and subsequently more refinement is added by increasing the resolution using the Split algorithm and adapted EM-algorithm. Alternatively, we could have used a top-down approach, i.e. starting at the most refined level and merge Gaussians to obtain more coarse representations. The advantage of starting at the most refined level is that there is no need to use

the adapted EM-algorithm. After merging two Gaussians, the merged Gaussian will most likely not cluster the demonstration data better than the two Gaussians used to create it. However, using the bottom-up approach might provide a better way to selectively increase the resolution in certain parts of the policy. Although this selective resolution is not part of this initial proposal, future work, where the construction of the Tree is done autonomously, could focus on this type of policy construction.

The convergence detection used to determine when to switch from one layer to another is based on the method of least squares. This convergence detection introduces two tuning parameters to the algorithm. The convergence threshold ϵ_{conv} and the number of samples M that form the reward window. Compared to a 'flat' policy representation, these are additional tuning parameters. These extra parameters are an undesired side effect since they demand extra prior knowledge to initialize the exploration process.

A parameterization of GMM was proposed that reduce the number of parameters required for exploration by exploiting the properties of the GMR process. This method is advantageous for exploration since a low number of parameters reduces the search space dimensionality. The reduction of the number of parameters is achieved by exploring only part of the covariance parameter space. As a result, the algorithm does not learn the optimal joint pdf $\mathcal{P}(\boldsymbol{X})$ for a random variable $\boldsymbol{X}$. Instead, the algorithm learns the optimal conditional probability $\mathcal{P}(\boldsymbol{X}^{\mathcal{O}}|\boldsymbol{X}^{\mathcal{I}})$, where $\boldsymbol{X} = [\boldsymbol{X}^{\mathcal{I}}, \boldsymbol{X}^{\mathcal{O}}]^T$. Hence, we need to pre-define $\boldsymbol{X}^{\mathcal{I}}$ and $\boldsymbol{X}^{\mathcal{O}}$ in advance, and do not learn the joint pdf during exploration.

Chapter 4

Experiments

In Chapter 3 a Tree-based movement representation was proposed. It is hypothesized that the proposed method yields to faster convergence than existing 'flat' policy representations.

Three experiments are used to validate this hypothesis. In the first two experiments a simple trajectory learning task is used. The Tree-models used in these experiments are not based on demonstration data, but manually constructed. By removing the autonomous construction of the Tree-model, these two experiments focus on the core topic of this thesis: analyzing the benefits of using a Tree-model during exploration.

The first experiment is designed to deliver a proof of concept. This experiment uses only the means of the Gaussians during exploration. This experiment keeps the concept simple by omitting the usage of the proposed parameter reduction. The second experiment uses the parameterization proposed in Section 3-2-2. This second experiment explores the full 'shape space' given by the GMM-GMR policy representation.

The objective of the third experiment is improving a feed-forward control policy used to swing-up a double pendulum on a cart. This experiment uses the complete framework proposed in this thesis; First, a Tree-model is constructed based on demonstration data. Then Learning from Exploration (LfE) is used to optimize the performance of the policy. This experiment was designed to prove that the Tree-model can also function in a more realistic scenario; learning to control dynamics of an unknown dynamical system.

4-1 Learning Trajectories I

The learning performance of a Tree-model is analyzed using a Tree-structure in which exploration is only applied to the means of the Gaussians, i.e. $\boldsymbol{\theta} = [\boldsymbol{\mu}_1 \cdots \boldsymbol{\mu}_K]^T$. For this experiment we thus omit exploration of the covariance matrix. As a result we can test the concept of the Tree-based exploration without requiring a method to explore the covariance matrix. This approach will keep the algorithm simple since the exploration of the covariance matrix is not trivial due to the constraints that lie on a covariance matrix. As a consequence

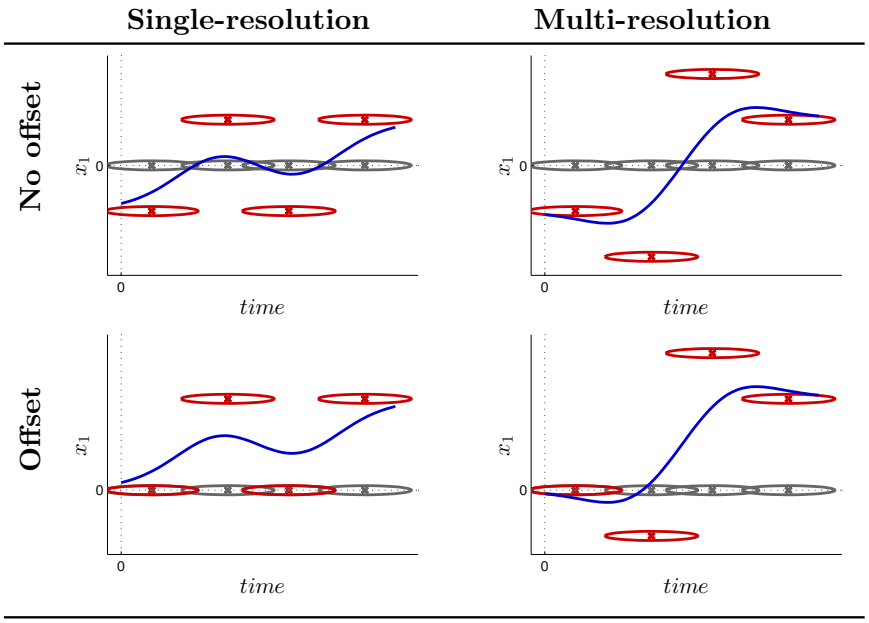


Figure 4-1: Four 2-dimensional trajectories generated by a GMM with 4 Gaussians (displayed in red). The Gaussians are placed in such a way that they do not contain a multi-resolution aspect (left column) or do contain a multi-resolution aspect (right column). In addition, an offset between the optimal trajectory and the initial condition of the GMM is introduced. The initial state of the GMM (displayed in gray) is centered with respect to the optimal GMM. In contrast, the initial state of the GMM in the bottom row has an offset with respect to the optimal solution.

we do not fully exploit the properties of the Gaussian Mixture Model (GMM) since we cannot influence the shape of the regressed trajectory through the covariance of the Gaussians.

In addition, the first experiment is used to discover if the Reuse of experience, discussed in Section 3-2-4, has positive effect on the learning performance.

The Tree-model is hypothesized to work best for learning tasks that inhibit a multi-resolution aspect. To this end four trajectories are designed. These trajectories are displayed in Figure 4-1. The trajectories are generated by a pre-defined GMM. The multi-resolution can be found in the placement of the Gaussians in the x_1 direction. The Figures in the right column contain a Multi-resolution aspect because the Gaussians can be grouped in two groups of two. The Figures in the left column do not have this multi-resolution aspect.

4-1-1 Experimental Set-up

The Tree-model used for these experiments has two layers. The first layer consist of 2 Gaussians and the second layer consist of 4 Gaussians. The Gaussians are placed at $x_1 = 0$ and are placed in such a way that the Gaussians of the second layer are equally spread over time. The initial state of the GMM is displayed in Figure 4-1. The Flat-model considered for comparison, is a GMM equal to the second layer of the Tree-Model.

The Task-performance is determined by analyzing the mean squared error between the goal

trajectory and the trajectory generated by the Tree-model:

$$r = \exp\left(-\alpha \frac{1}{N} \sum_{n=1}^{N} (x_n - x_n^g)^2\right) \tag{4-1}$$

here N is the number of samples used, x_n and x_n^g are the x position of the model and goal trajectory, respectively. The exponential function is used to force the reward to lie within the range $(0, 1]$. α is a tuning factor that is used to tune the 'climbing' behavior of the reward function. A low α will result a relative high reward, even for 'bad' trials. A high α will result only in high rewards for very good trials. Setting α too high will lead to very slow convergence towards the optimum since the reward only increases little until very close to the optimal solution.

Parameter Settings

Table 4-1 shows the Exploration settings used during this experiment. The number of roll-outs is set in such a way that the convergence to the final value is clearly visible. The number of Roll-outs before the first update, and the number of Roll-outs considered for the parameter update are set to 5.

The convergence threshold is set very strict. By setting the convergence threshold to zero, convergence is only detected when the reward increase is zero or negative for M successive roll-outs. This is only the case if a layer has truly converged. Choosing a lower convergence criterion resulted in not optimally exploiting the faster learning in the first layer.

The reward parameter α is tuned to make sure that the refinements of the motion represent a significant part of the reward. With α set too low, optimizing the first layer of the Tree-model results to a relative high reward, and the refinements made in the second layer will not clearly show in the reward.

Exploration of the parameters depends on two settings; the initial exploration noise matrix $\mathbf{\Sigma}_{init}^{noise}$ and the minimum exploration noise $\mathbf{\Sigma}_{min}^{Noise}$. The minimum exploration noise is required to prevent early convergence. Both matrices are initialized as a diagonal matrix. The diagonal elements describe the exploration along the different dimensions of the problem.

To determine the effect of the initial and minimum exploration noise on the learning performance of the Tree-model, different settings for these parameters are studied. A noise matrix $\mathbf{\Sigma}_{init}^{Noise}$ is scaled by factors γ and factors ϵ_{Noise} to obtain the initial and minimum noise, respectively. First, 6 experiments are performed using different minimum noise settings to determine the optimal value. During these experiments the initial noise is kept constant ($\gamma = 1$). Then 6 experiments are performed using different initial noise settings. During these latter experiments, the minimum noise is taken equal to the best performing minimum noise for the Flat-model. By selecting the minimum noise based on the flat-model performance, we ensure that the flat-model is in optimal performance.

The noise Matrix $\mathbf{\Sigma}_{init}^{Noise}$ that is used as a basis for all the experiments is obtained by manual tuning. The exploration in the x_1 direction is taken to be large enough to sample a sufficient part in the x_1 direction. The exploration noise in the time direction is set to a relatively low value to prevent the Gaussians from switching their relative position with respect to each-other.

All the trajectories displayed in Figure 4-1 are learned using both the Tree-model and the Flat-model. The experiment is repeated 30 times for each condition.

Variable	Value
roll-outs	400
S_{init}	5
I	5
M	20
ϵ_{conv}	0
α	20
$\mathbf{\Sigma}^{Noise}$	$\text{diag}([.05, 1])\,\gamma$ with $\gamma \in \{0.1,\ 0.2,\ 0.5,\ 1,\ 2, 5\}$
$\mathbf{\Sigma}^{Noise}_{min}$	$\text{diag}([.05, 1])\,\epsilon_{Noise}$ with $\epsilon_{Noise} \in \{1e{-}3,\ 2e{-}3,\ 5e{-}3,\ 1e{-}2,\ 2e{-}2,\ 5e{-}2\}$

Table 4-1: Summary of Tree-model exploration algorithm parameters used for experiments described in Section 4-1.

4-1-2 Results

In this section the results of Experiment I are presented. This section will only show the experimental results that are required to discuss the observations made. The complete overview of experimental results can be found in Appendix C-1.

Optimal minimum noise

The first series of experiments was performed to determine the optimal minimum noise setting. Figure 4-2 shows the learning curves when learning the multi-resolution motion without an offset for different noise settings per layer. The largest minimum noise setting, $\epsilon_{Noise} = 5e{-}2$, results in the lowest rewards for both the Tree-model and Flat-model. In case of the larger minimum noise settings, the algorithm fails to converge to the optimal solution. However, when $\epsilon_{Noise} = 5e{-}2$ is too small, the algorithm converges very slow as can be observed for both the Tree-model and the Flat-model. For the motions with multi-resolution, the Flat-model has the highest convergence value for $\epsilon_{Noise} = 1e{-}2$ and $\epsilon_{Noise} = 5e{-}3$. The increase in reward during the first 200 trials is steeper for $\epsilon^{Noise} = 1e{-}2$. Therefore, $\epsilon_{Noise} = 1e{-}2$ is used for the initial noise experiments in the next section.

The switching behavior of the Tree-model is displayed in Figure 4-3. The graph shows the number of Roll-outs made before switching from layer 1 to layer 2. It is observed that the algorithm tends to switch within less number of roll-outs as the minimum noise increases. This corresponds to the slower convergence of the first layer shown in Figure 4-2.

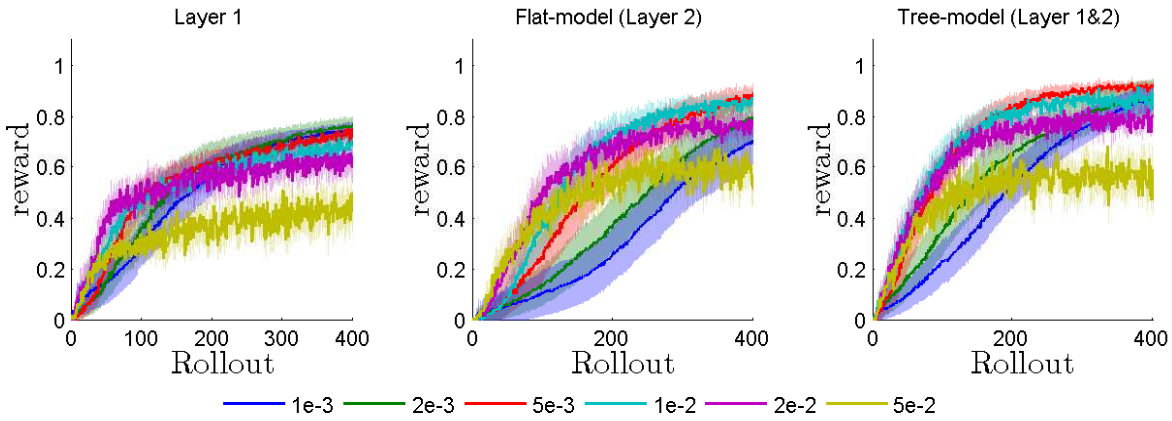


Figure 4-2: Learning curves for the multi-resolution movement without offset displayed in figure 4-1 for different minimum noise settings. The values in the legend indicate the ϵ_{Noise} values as indicated in Table 4-1.

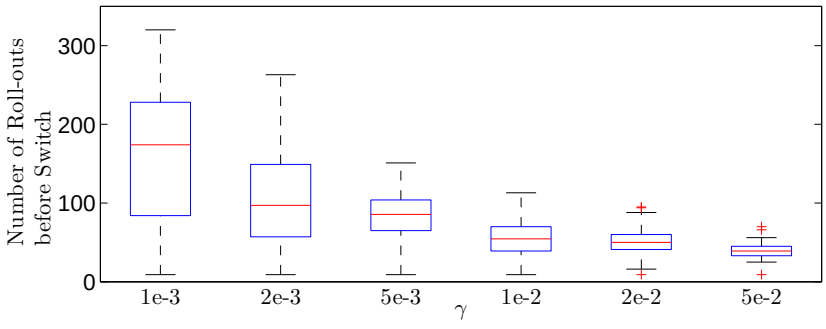


Figure 4-3: The number of Roll-outs before switching between Layers during learning the multi-resolution movement without offset displayed in figure 4-1 for different minimum noise settings. The values along the x-axis indicate the ϵ_{Noise} values as indicated in Table 4-1.

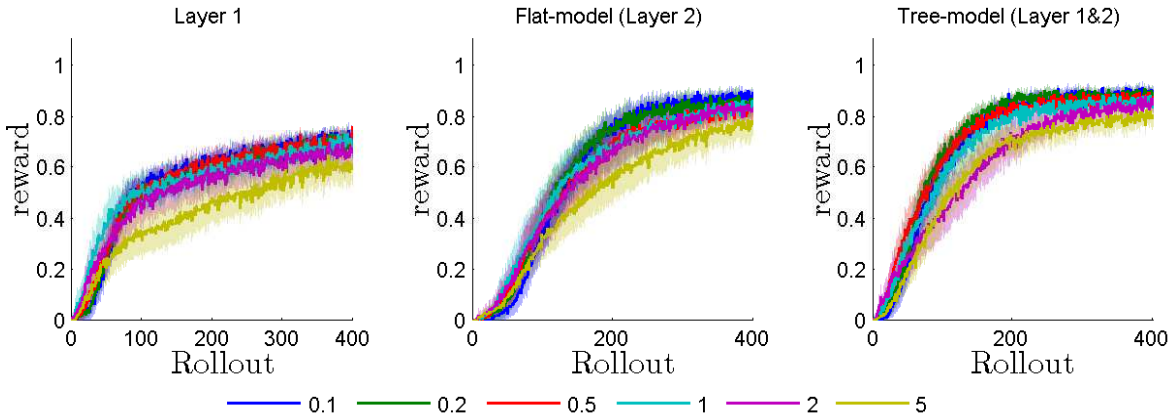


Figure 4-4: Learning curves for different initial noise settings while learning the Multi-resolution trajectory without offset as displayed in Figure 4-1. From left to right the plots represent; Learning using only the first layer, learning using the second layer (flat-model) and learning using both layers (Tree-model).

Optimal initial Noise

A second series of experiments is carried out to compare the learning performance of the Tree-model and the Flat-model for different initial noise settings. These 6 different initial noise settings are given in Table 4-1 and indicated by their γ value. γ indicates the fraction of the noise matrix that was obtained by manual tuning. Each of the 6 noise settings is used learn the 4 trajectories given in Figure 4-1, i.e. a total of $6 \times 4 = 24$ situations is tested. For all experiments the minimum noise found in previous section was used, i.e. $\mathbf{\Sigma}_{min}^{noise} = \text{diag}([0.1, 1]) \cdot 1e{-}2$.

Figure 4-4 shows learning curves for the Multi-resolution trajectory without offset. The effect of varying the initial noise is smaller compared to the variation of the minimum Noise. The largest noise setting, $\gamma = 5$, shows the lowest performance for the various movements. The performance of three lowest noise settings is about the same. The same holds for the other 3 trajectories of which the results are shown in Appendix C-1.

Comparing the Tree-model and the Flat-model

Figure 4-5 shows the learning curves of the 4 different movements that are considered in this experiment. The figure shows the results obtained with the minimum noise fraction $\epsilon_{Noise} = 1e{-}2$ and the noise fraction $\gamma = 0.1$. These are the settings for which the Flat-model performs best. The results of the other settings are given in Appendix C-1. Each sub-figure in Figure 4-5 shows three line representing learning using the first layer of the Tree-model, the second layer of the Tree-model (Flat-model), and both Layers of the Tree-model.

The first thing that stands out is the end performance of Layer 1. The final reward in the single-resolution trajectory is much lower than the final reward in the multi-resolution trajectory. In the single-resolution case, the first layer does not add a lot of value causing the small gain in performance. In the multi-resolution case the first layer does add value. But

in both the single-resolution and multi-resolution trajectories, the second layer is required to achieve optimal performance.

Both the Flat and the Tree model converge to about the same reward and thus perform equally well based on the maximum reward received. In the single-resolution case without offset (top left, Figure 4-5) the Tree-model is outperformed by the Flat-model. Although the learning curve appears equally steep, the Tree-model shows a drop in reward around 30 roll-outs. This point is around the switching point between layer 1 and 2, discussed later on.

In the multi-resolution case (right column, Figure 4-5), the Flat-model is outperformed by the Tree-model in learning speed. The results clearly show that optimization using Layer 1&2 initially overlap with the results of Layer 1. But as Layer 1 convergences, the reward of Layer 1&2 keep increasing.

To confirm our hypothesis that learning using the Tree-based movement representation is advantages over a flat policy representation a two-sample t-test is performed. The null-hypothesis H_0 represents the case that the Tree-model and the Flat-model perform equally well. The alternative hypothesis states that the Tree-model outperforms the Flat-model, i.e. results in higher reward. The results of this test are displayed in Figure 4-6. Based on this t-test it can be concluded, $p < 0.05$, that the Tree-model outperforms the Flat-model in the steepest part of the learning curve for the movements with a Multi-resolution aspect.

In the single resolution case, the Tree-model is not faster than the Flat-model. Based on the learning curve of the Single-resolution no offset case, we can conclude that the Tree-model switches too late. The learning curve of the Tree-model tends to 'stick' to Layer 1 between Roll-outs 10 to 30. This behavior does not show in the Multi-resolution case.

The results of the Tree-model presented in figures 4-5 and 4-6 are obtained using the 'Reuse of Experience' method. In this method, the exploration noise of the second layer is initialized based on the results of the first layer. To find out if this method functions well, a comparison is made between reusing the experience and not reusing the experience from the first layer. When not reusing the experience from the first layer in the second layer, the initial exploration noise of the second layer is taken equal to the initial exploration noise of the first layer. Using the same Tree-model as before, the 4 trajectories are learned without re-using experience when switching layers. Figure 4-7 shows the learning results for the Multi-resolution trajectory without an offset. The Reuse of rewards especially shows to be beneficial for large exploration noise. For smaller initial exploration noise,$\gamma = 0.1, 0.2, 0.3$, the performance is about equal.

4-2 Learning Trajectories II

The intention of the second experiment is to show the potential of the Tree-model by exploring the complete parameters space of the proposed approach, i.e. exploring the complete space of parameters that influence the Gaussian Mixture Regression (GMR) process.

4-2-1 Experimental Set-up

The experimental setup is similar to the experimental set-up described in Section 4-1-1. The objective is to learn the trajectories shown in Figure 4-8. The performance evaluation is based

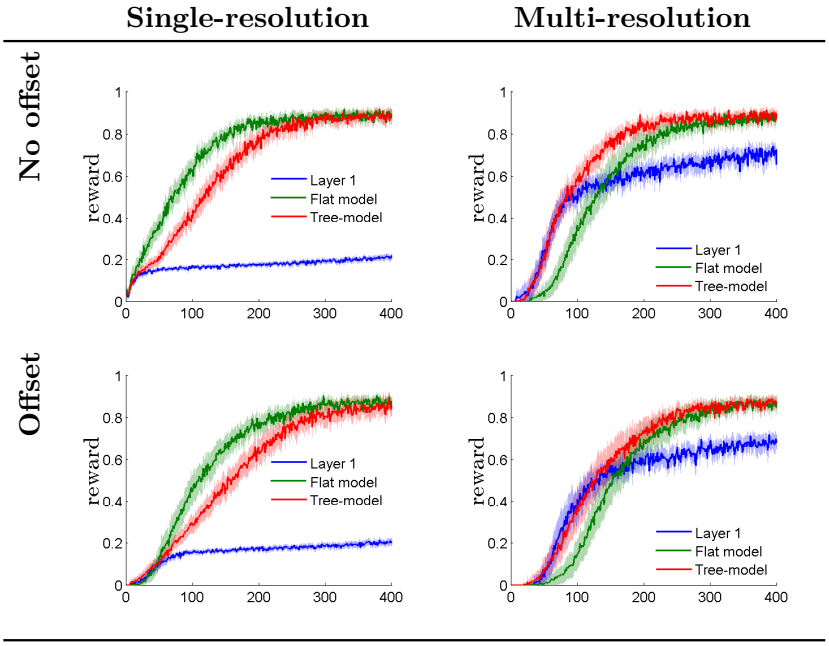


Figure 4-5: The reward plotted against the number of roll-outs. The different lines represent the different Layers: Layer 1 (blue), Layer 2 (green) and All layers (Red). The lines show the average reward over 30 repetitions. The lighter colored areas show the 95% confidence interval of the means with corresponding color.

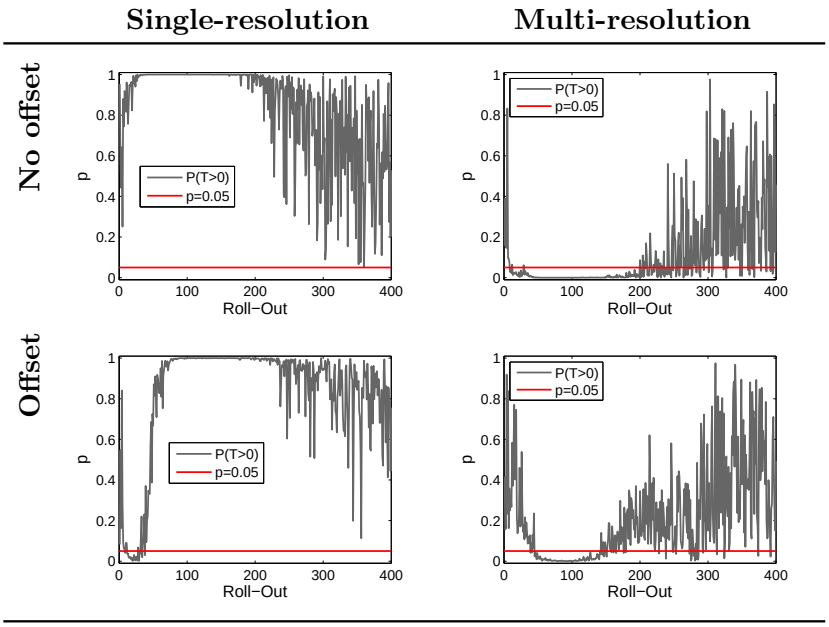


Figure 4-6: The Experimental results for learning the multi-resolution trajectory without an Offset. At each roll-out a two-sample t-test is performed based on the performance of 'Layer 2' and 'All layer'. The null hypothesis H_0 indicates that both situations perform equally well. The alternative hypothesis, H_1, indicates that 'All layer' outperforms 'Layer 2'. The p-value indicates the confidence in H_0 in favor of H_1 based on the right-tail probability.

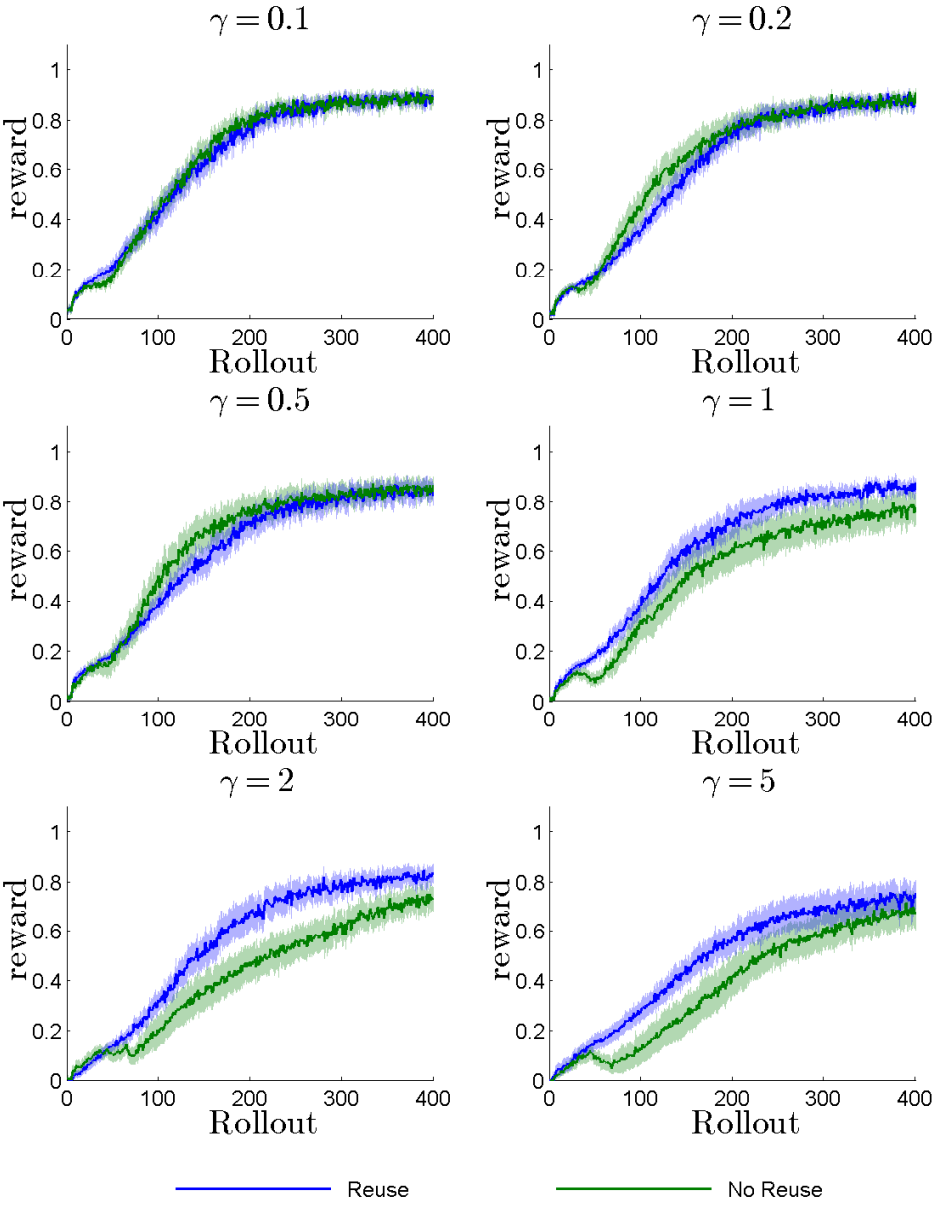
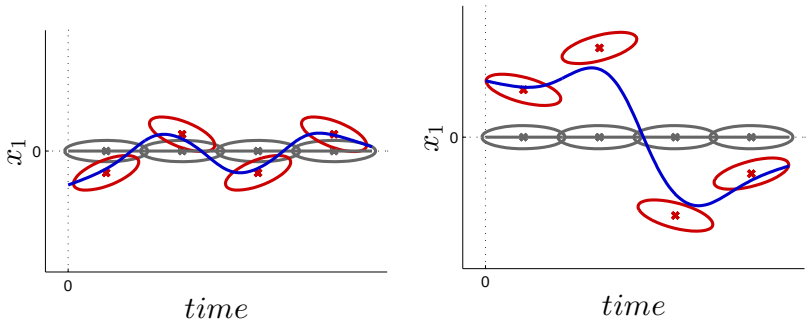


Figure 4-7: Learning curves for learning a Multi-resolution trajectory without offset with and without the reuse of reward. The different graphs represent different initial noise settings. γ corresponds to the value of the initial Noise matrix $\Sigma_{init}^{Noise} = \text{diag}([.1, 1])\,\gamma$. The lines show the average reward over 30 repetitions. The lighter colored areas show the 95% confidence interval of the means with corresponding color.



(a) Single-resolution **(b)** Multi-resolution

Figure 4-8: Two 2-dimensional trajectories generated by a GMM with 4 Gaussians (displayed in red). The Gaussians are placed such that they do not contain a multi-resolution aspect (left column) or do contain a multi-resolution aspect (right column). The initial state of the GMM (displayed in gray) is centered with respect to the optimal GMM. In contrast, the initial state of the GMM in the bottom row has an offset with respect to the optimal solution.

on the mean squared error between the goal trajectory and the trajectory generated by the model. The reward function is as given in Eq. (4-1).

A 2 layer Tree-model is used with 2 Gaussians in the first layer and 4 Gaussians in the second layer. During exploration the GMM parameterization proposed in section 3-2-2 is used. Each Gaussian is represented by 3 parameters:

$$\boldsymbol{\theta}_k = \begin{bmatrix} \mu^t \\ \mu^{x_t} \\ \theta^{\Sigma^{\mathcal{IO}}} \end{bmatrix} \tag{4-2}$$

μ^t and μ^{x_t} represent the location of the mean, and $\theta^{\Sigma^{\mathcal{IO}}}$ controls the correlation between t and x_1.

All the trajectories displayed in Figure 4-8 are learned using both the Tree-model and the Flat-model. The parameters for the learning algorithm are listed in Table 4-2. These parameters were manually tuned. Note that the initial noise Σ^{Noise} per Gaussian now has three entries, two entries for the mean and one for the Covariance $\boldsymbol{\Sigma}^{\mathcal{OI}}$.

The experiment is repeated 30 times for each condition.

4-2-2 Results

In this section the results of Experiment II are presented. This section will only show the experimental results that are required to discuss the observations made. The complete overview of experimental results can be found in Appendix C-2.

Optimal Minimum Noise

First, 6 experiments were performed to determine the optimal minimum noise. This optimal minimum noise setting is used in later experiments to compare the Tree-model and the Flat-

Variable	Value
roll-outs	400
S_{init}	5
I	5
M	20
ϵ_{conv}	0
$\mathbf{\Sigma}^{Noise}$	$\text{diag}([0.1, 1, 0.1])\,\gamma$ with $\gamma \in \{0.1,\ 0.2,\ 0.5,\ 1,\ 2, 5\}$
$\mathbf{\Sigma}^{Noise}_{min}$	$\text{diag}([0.1, 1, 0.1])\,\epsilon_{Noise}$ with $\epsilon_{Noise} \in \{1e{-}3,\ 2e{-}3,\ 5e{-}3,\ 1e{-}2,\ 2e{-}2,\ 5e{-}2\}$
α	20

Table 4-2: Summary of Tree-model exploration algorithm parameters used for experiments described in Section 4-2.

model for various initial noise settings. The results of the minimum noise experiment are shown in Figure 4-9.

The results are similar to the results of Experiment I. Increasing ϵ_{Noise} towards $5e{-}2$ results in a relatively low convergence value. Decreasing ϵ_{Noise} towards $1e{-}3$ results in very slow convergence but with a higher convergence value. In contrast to the results of experiment I, the increase in reward of layer 1 is not as large as the increase of the Flat-model or the Tree-model. When looking to the solution space, shown in 4-10, we observe that Layer 1 converges to roughly two different solutions of which one is less optimal than the other. The average reward of the first layer is thus lower due to the less optimal solutions that were found.

Based on the results of the Flat-model a noise fraction is selected for later experiments. Both $\epsilon_{Noise} = 1e{-}2$ and $\epsilon_{Noise} = 5e{-}3$ seem to be reasonable options. $\epsilon_{Noise} = 1e{-}2$ has a slightly higher learning speed within the first 200 roll-outs in favor of $\epsilon_{Noise} = 5e{-}3$. $\epsilon_{Noise} = 5e{-}3$ shows a slightly higher convergence value in favor of $\epsilon_{Noise} = 1e{-}2$. For the initial noise experiments discussed in the following section, a minimum noise fraction of $\epsilon_{Noise} = 1e{-}2$ is used.

Optimal Initial Noise

A second series of experiments is performed to compare the learning performance of the Tree-model and the Flat-model for different initial noise settings. These 6 different initial noise settings are given in Table 4-2 and indicated by their γ value. γ indicates the fraction of the noise matrix that was obtained by manual tuning. Each of the 6 noise settings used during the learning of the 2 trajectories given in Figure 4-8, i.e. a total of $6 \times 2 = 12$ situations were tested. For all experiments the minimum noise found in previous section was used, i.e. $\mathbf{\Sigma}^{noise}_{min} = \text{diag}([0.1, 1, 0.1]) \cdot 1e{-}2$.

The results of the experiments are summarized in Figure 4-11. Both the Tree-model and the Flat-model perform best for low initial noise, i.e. $\gamma = 0.1$ and $\gamma = 0.2$. The highest initial noise settings, $\gamma = 5$ clearly converge much slower for both Tree-model and Flat-model.

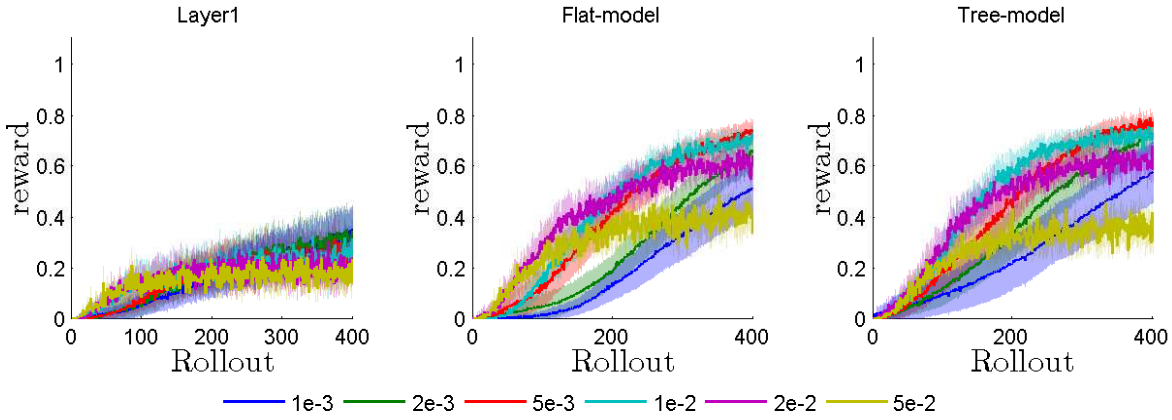
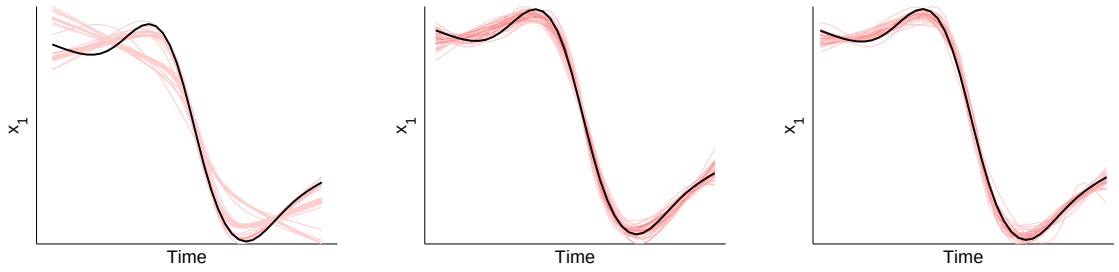


Figure 4-9: Learning curves when learning the Multi-resolution trajectory for different minimum noise settings. The results are split per layer. The numbers in the legend represent the ϵ_{Noise} to determine the noise matrix $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])\,\epsilon_{Noise}$. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.



(a) Layer 1 **(b)** Flat-model **(c)** Tree-model

Figure 4-10: Figures show the solutions obtained after learning the Multi-resolution motion for 30 repetitions using either the first layer of the Tree-model (a), the second layer of the Tree-model (b, Flat-model) or using both layers of the Tree-model (c). The solutions are given in red and the desired trajectory in black. The results were obtained using $\epsilon_{Noise} = 1e{-}2$, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,1e{-}2$.

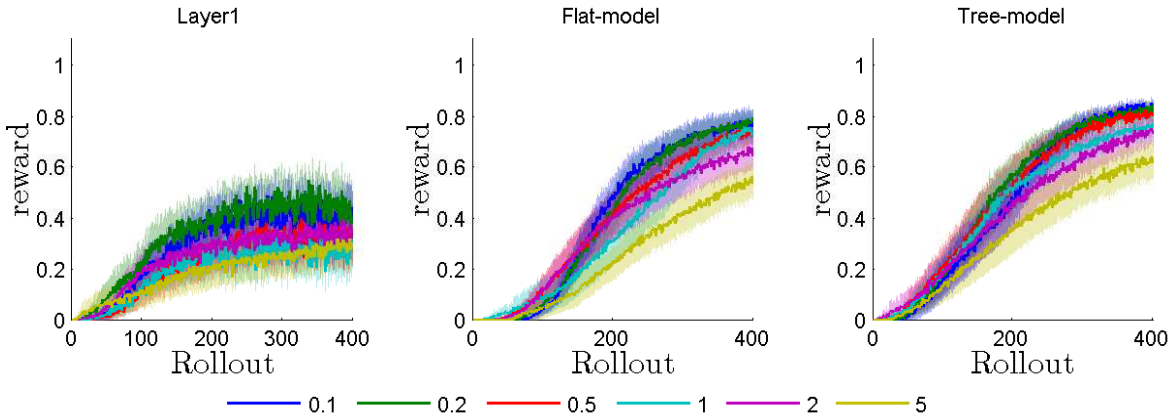


Figure 4-11: Learning curves resulting the learning of a Multi-resolution trajectory for different initial noise settings. The results are splitted per layer. The numbers in the legend represent the γ to determine the noise matrix $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])\,\gamma$. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

Figure 4-12 summarizes the switching behavior of the algorithm when using Tree-based exploration. The median number of Roll-outs made before switching is about equal for all initial noise settings. However, in case of the low Initial noise settings, the interquartile range (IQR) stretches all the way to the minimum number of roll-outs required before a switch can be made: Sometimes the Algorithm switches at the first possibility.

Comparing Tree-model and Flat-model

To compare the Tree-model with the Flat-model, the rewards of the Tree-model and the Flat-model are plotted together. Figure 4-13 shows the results for $\gamma = 0.1$, i.e. the best performing initial noise setting.

The resulting trajectories of all 30 repetitions for Layer 1, Flat-model and the Tree-model are given in Figure 4-15. Both the Tree-model and the Flat-model converge towards the optimal trajectory. The results of Layer 1 are less optimal. This is mainly caused by the fact that the first layer does not have enough flexibility to learn the optimal motion. This is visible in the first part of the trajectory. Secondly, similar to the minimum noise experiments, layer 1 sometimes seems to converges to a bad solution. This is visible in the latter part of the trajectory.

In case of the Multi-resolution trajectory the Tree-model seems to outperform the Flat-model in the steepest learning part. To confirm this observation, a two-sample t-test is performed similar to Experiment I. The results of the two-sample t-test are given in Figure 4-14. The t-test shows that Tree-model has indeed a higher average reward between approximately roll-out 50 and 120 ($p < 0.05$). However, after approximately 120 roll-outs, the performance of the two models is about the same. For the other initial noise settings the same results, or results in favor of the Tree-model are obtained as shown in Appendix C-2.

Based on Figure 4-13a one might conclude that the Tree-model outperforms the flat-model for the single-resolution trajectory. However, the difference between the Tree-model and the

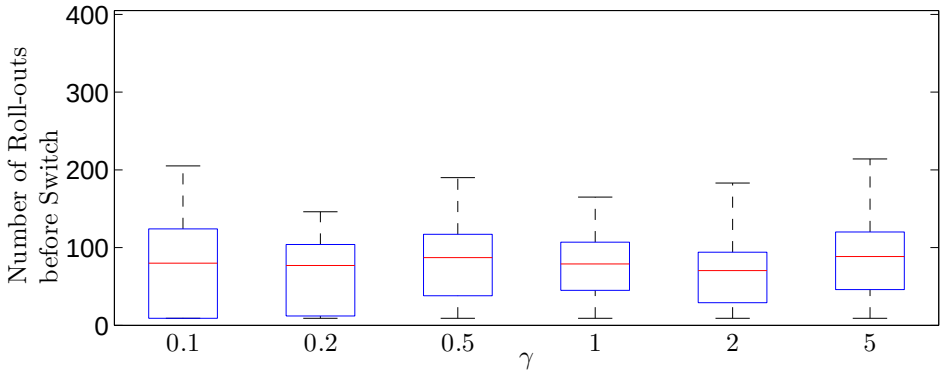
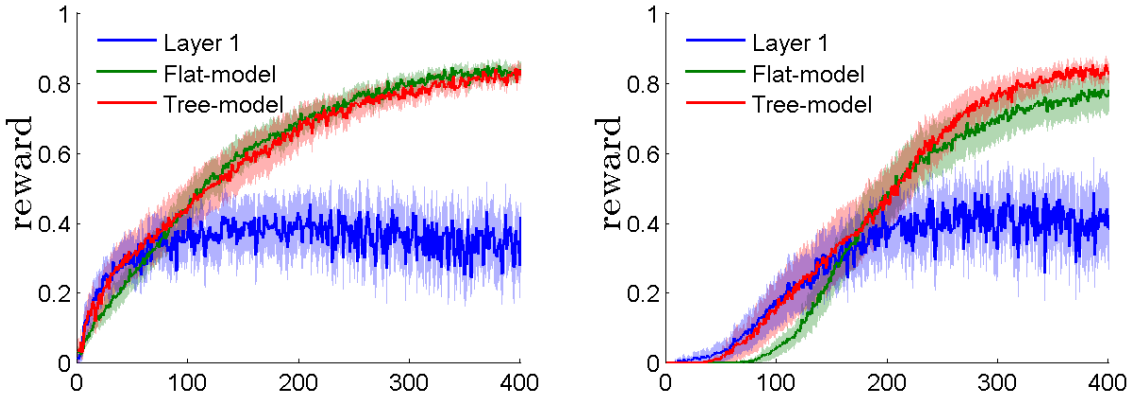


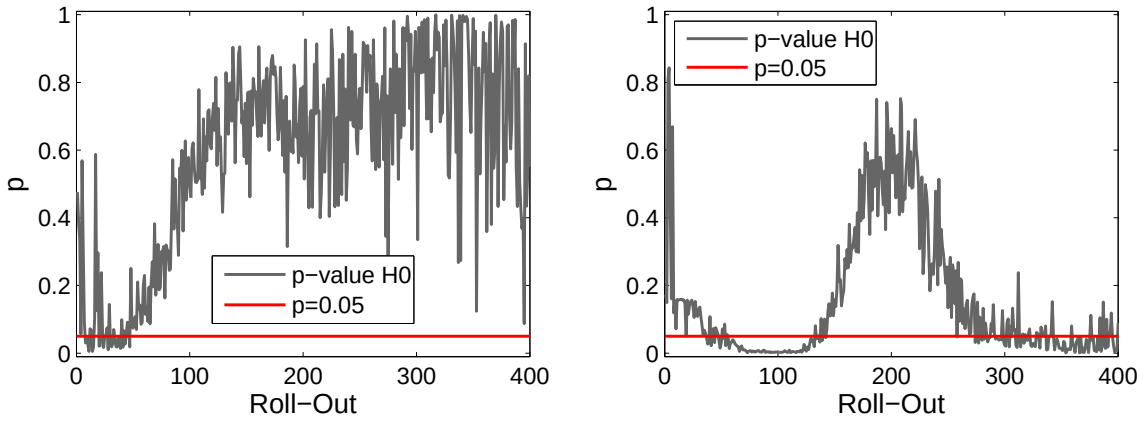
Figure 4-12: Figure gives a summary of the switching behavior of the Tree-model while learning the Multi-resolution trajectory (shown in Figure 4-8) for different initial noise settings. The boxplots indicate the number of roll-outs used before switching from the first layer of the Tree-model to the second layer of the Tree-model. Each boxplot is based on 30 observations. The ϵ_{Noise} in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\epsilon_{Noise}$.

Flat-model is not significant as shown in Figure 4-14a. For other initial noise settings, it is shown clearer that the Tree-model does not outperform the Flat-model in the single-resolution case.



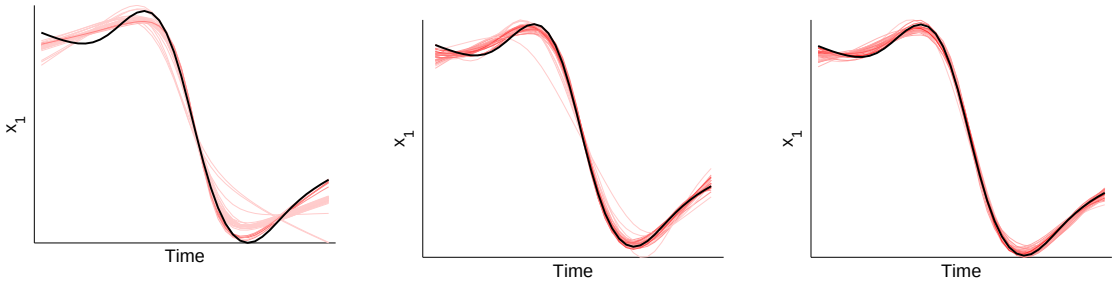
(a) Single-resolution **(b)** Multi-resolution

Figure 4-13: The Figure shows the learning performance of a 2 Layer Tree-model while learning a single-resolution (a) and a multi-resolution trajectory (shown in Figure 4-8). The results are based on the Initial noise with $\gamma = 0.1$ i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1, .1])\, 0.1$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.



(a) Single-resolution **(b)** Multi-resolution

Figure 4-14: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning a Single-resolution (a) and Multi-resolution (b) trajectory (shown in Figure Figure 4-8). The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flat-model. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The initial noise fraction was set to $\gamma = 0.1$, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])\, \gamma$.



(a) Layer 1 **(b)** Flat-model **(c)** Tree-model

Figure 4-15: Figures show the solutions obtained after learning the Multi-resolution motion for 30 repetitions using either the first layer of the Tree-model (a), the second layer of the Tree-model (b) or using both layers of the Tree-model (c). The solutions are given in red and the desired trajectory in black. The results were obtained using $\gamma = 0.1$, i.e. $\Sigma^{Noise} = \text{diag}\left([0.1, 1]\right) 0.1$.

4-3 Double pendulum on cart: Swing-up

The experiments discussed in the previous sections have a reward function that is based on the mean-squared error. The reward is thus directly related to the shape of the movement. In practical applications, the reward is often not directly related to trajectory of the movement. Instead, the reward depends on the outcome of the task. For example, in the ball-in-cup task [15] the reward depends on how close the ball is to the cup. In this example the control-policy actuates an unknown dynamical system, and the reward is based on the observable state of the unknown dynamical system.

To show that the Tree-model can learn to control an unknown dynamical system, we investigate the swing-up phase of a double pendulum attached to a cart. The objective is to improve the performance of a feed-forward controller that is able to swing-up the double pendulum to a state in which both pendulums are in their unstable equilibrium. This problem is interesting for the Tree-model because this type of controller can be represented by a sum of cosine [7]. The feed-forward swing-up controller thus has a potential multi-resolution aspect.

The approach proposed by Graichen et al. [7] relies on a feed-forward controller and a linear feed-back controller with time-varying gains. Both controllers are active during the swing-up phase and after swing-up the feed-forward controller is switched off. The swing-up phase relies on both the feedback and the feed-forward controller. Without the feedback control, the feed-forward controller is not able to perform a reasonable swing-up. The objective of the experiment used in this thesis is to improve the feed-forward controller, in such a way that it is able to swing the double pendulum towards the unstable equilibrium.

4-3-1 Experimental Set-up

Dynamical System

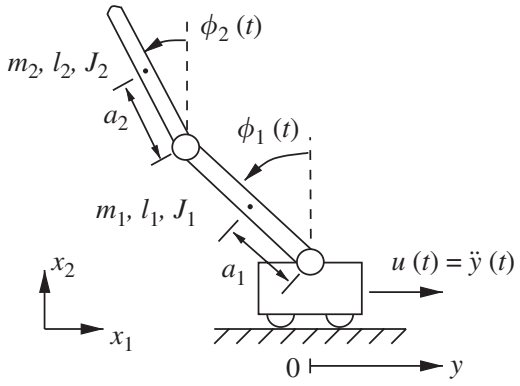


Figure 4-16: Schematic representation of the double pendulum on a cart. Image adopted from [7]

The objective is to learn a mapping between time t and cart acceleration $\ddot{y}$:

$$\pi : t \mapsto \ddot{y} \tag{4-3}$$

The double pendulum on a cart is simulated using Matlab. The double pendulum on a cart, shown in Figure 4-16, has three degrees of freedom (ϕ_1, ϕ_2, y). Like,[7], $\ddot{y}$ is an input value and the dynamics of the cart are thus omitted. The dynamics of the remaining system are described by two differential equations that are derived in Appendix B. Each trial the Equations of Motion (EOM) are integrated using the Matlab `ODE45` function.

Policy Representation and Initialization

Learning is initialized by a demonstration obtained from a non-functioning feed-forward controller. The initial trajectory is based on the desired trajectory of the cart:

$$y(t, \boldsymbol{p}) = a_0 + a_1 \cos\left(\frac{\pi t}{T}\right) + \sum_{i=2}^{5} p_{i-1} \cos\left(\frac{i\pi t}{T}\right). \tag{4-4}$$

Here $\boldsymbol{p}$ is a vector of free parameters, $a_0 = -p_1 - p_3$ and $a_1 = -p_2 - p_4$. T refers to the swing-up time. For this experiment we adopt the parameters found in [7] for $T = 2.2$, i.e. $p_1 = -0.106, p_2 = -0.185, p_3 = 0.092, p_4 = 0.134$.

The demonstration data used to initialize the learning process is obtained by differentiating Eq. (4-4) twice with respect to time:

$$\ddot{y}(t, \boldsymbol{p}) = -\frac{\pi^2 a_1}{T^2} \cos\left(\frac{\pi t}{T}\right) - \sum_{i=2}^{5} \frac{(i\pi)^2 p_{i-1}}{T^2} \cos\left(\frac{i\pi t}{T}\right) \tag{4-5}$$

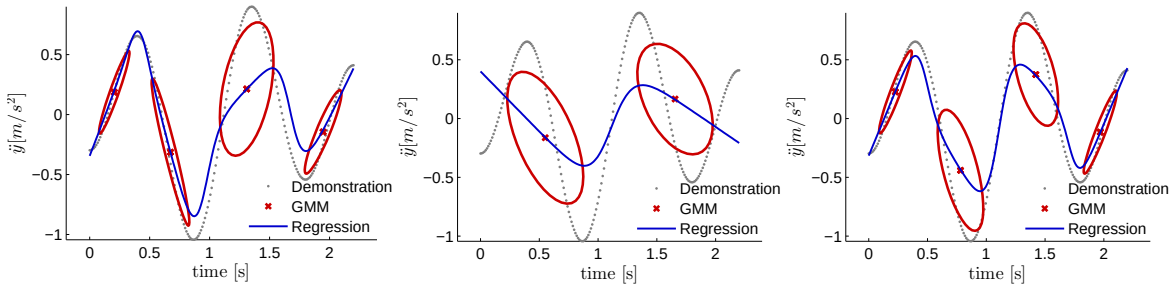
Based on the demonstration data, a 2 layer Tree-model is created. The first layer consist of 2 Gaussians and the final layer consists of 4 Gaussians. The Tree-model is initialized using the procedure described in Section 3-1. For comparison a Flat-model with 4 Gaussians is created using standard Expectation Maximization (EM). Both the Flat-model and the first Layer of the Tree model were initialized before applying the EM algorithm by placing the Gaussians at $\ddot{y} = 0$ equally spaced in time. A regularization term was added to the Covariance of each Gaussian during EM to prevent singularities. This regularization term prevents the Gaussians from 'collapsing' onto the single demonstrated trajectory.

The Tree-model and the Flat-model are displayed in Figure 4-16. Notice how the Flat-model (Figure 4-17a) and the second layer of the Tree-model (Figure 4-17c) yield to a different Solution. The Flat-model and Layer 2 of the Tree-model result in a Log-likelihood of -0.6764 and -0.7585, respectively. The Flat-model thus produces a better fit of the demonstration data. However, none of the models shows a tight fit to the demonstrated trajectory.

Reward function

The objective of the feed-forward controller is to get the system in the state $\phi_1 = \phi_2 = 0, \dot{\phi}_1 = \dot{\phi}_2 = 0, y = 0$. This optimal state is translated in the following cost-function per time step:

$$c_t = \frac{5}{\pi^2}\left(\phi_1^2 + \phi_2^2\right) + \left(\frac{7y}{10}\right)^4 + \frac{1}{50}\left(\dot{\phi}_1^2 + \dot{\phi}_2^2 + \dot{y}^2\right) \tag{4-6}$$



(a) Flat model **(b)** Tree model: Layer 1 **(c)** Tree model: Layer 2

Figure 4-17: Based on the demonstration data describing the accelleration of the cart during the swing-up phase, a Flat model with 4 Gaussians, and a 2 layer Tree-model is created. A regularization term was used at the end of the Maximization step of the EM-algorithm to prevent singularities in the solution.

This cost-function is a modified version of the reward function used for a single pendulum swing-up used by Kober et al. [15]. The reward given to each roll-out is the 'best' state of the episode, i.e. the state with the lowest cost. The reward is calculated as:

$$r = \exp\left(-\alpha \min_{t \in T}(c_t)\right) \tag{4-7}$$

Experimental Settings

The exploration settings are summarized in Table 4-3. The settings were manually tuned to achieve optimal performance. Each episode consist of simulation of the dynamical system for $2.5[s]$, starting from the initial condition $\phi_1 = \phi_2 = \pi, \dot{\phi}_1 = \dot{\phi}_2 = 0, y = 0$, i.e. stable equilibrium.

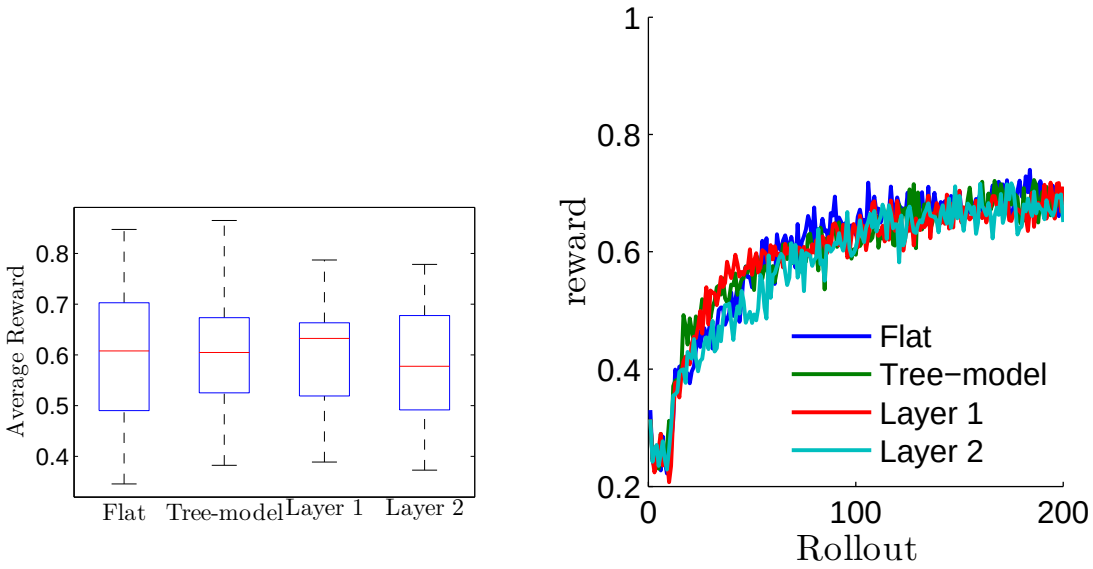
Learning is done in four different settings: (i) learning the Flat model, (ii) learning using only the first layer of the Tree-model, (iii) learning using only the second layer of the Tree-model, and (iv) learning using both the first and the second layer of the Tree-model. Each setting is repeated 30 times to obtain sufficient statistical proof.

4-3-2 Results

The results of learning the swing-up task are summarized in Figure 4-18.

The results do not show striking difference between the different experimental settings. The increase in reward is equal for all conditions and the convergence value is also equal for all cases. Figure 4-18a shows the average reward obtained over all roll-outs. This plot confirms that there is almost no difference between Flat-model, Tree-model or the individual layers of the Tree-model. The variance of the results is very large. This indicates that none of the methods is guaranteed to provide a working feed-forward controller after 200 roll-outs.

Notice that layer 1 is able to converge to solutions with the same reward as layer 2 this indicates that the added value of the second layer of the Tree-model is doubtful.



(a) Average reward **(b)** Learning Rewards

Figure 4-18: The learning results of the swing-up double pendulum on a cart task for four different conditions. The plots are based on 30 repetitions of each condition. Figure 4-18a shows the Average reward over all roll-outs. Figure 4-18b shows the learning curve of during the first 200 roll-outs. The entries of the legend; Flat: A four Gaussian single layer policy as displayed in 4-17a; Layer 1&2: Using a 2 layer Tree-model with 2 Gaussians in Layer 1 and four Gaussians in layer 2; Layer 1: Using the Tree-model, but only optimize at the first layer; Layer 2: Using the Tree-model but only optimize the second layer.

Variable	Value
roll-outs	200
S_{init}	10
I	2
M	20
ϵ_{conv}	0
$\mathbf{\Sigma}^{Noise}$	$\text{diag}([.01, 0.3, .02])$
ϵ_{Noise}	$5e-3$
α	.25

Table 4-3: Summary of Tree-model exploration algorithm parameters used for the double pendulum swing-up

4-4 Discussion

In this Chapter three experiments and their results were presented. In this final Section the results are discussed.

It was hypothesized that the Tree-model policy representation will result in faster convergence. The Experiments discussed in Sections 4-1 and 4-2 have shown that the Tree-model has faster convergence for tasks which inhibit a multi-resolution aspect. This comes at the expense of slightly decreasing the performance over flat-model for none multi-resolution cases. However, it is promising that the performance loss is only small.

The performance of the convergence criterion varies within the results. For the multi-resolution trajectories, the switch between layer 1 & 2 seems optimal. The learning curve of the Tree-model follows the learning curve of Layer 1 until layer 1 start to converge. From that point forward, the Tree-model continues the steep learning curve, i.e. it switches to the second layer. However, for the single-resolution cases the Tree-model seems to switch too late. The learning curve of the Tree-model 'sticks' to the learning curve of the first layer. This behavior results in lower performance because precious roll-outs are used to explore Layer 1, whereas those roll-outs could have better been spent on layer 2.

In section 4-1 the proposed method to reuse rewards to initialize the covariance matrix was validated. The results showed that for large initial covariance noise, the reuse of rewards prevented a drop in reward. However, this behavior is not visible for smaller Initial covariance noises. When not reusing rewards, the initial exploration noise at the start of layer 2 is taken equal to the noise used at the start of layer 1. For large initial exploration noise a drop in reward is caused at the start of Layer 2 due to too rough exploration. The reuse of rewards prevents this behavior because the noise at layer 2 is initialized based on the results obtained in the previous layer. In case of smaller initial exploration noise, the drop in reward in layer 2 is less severe because the magnitude of the noise is better suited for the second layer. The reuse of rewards can be advantageous in cases where the exploration noise is badly estimated. In those cases, the badly estimated exploration noise will only affect the learning process once. In contrast, when not reusing rewards, poorly estimated exploration noise will be used at each layer.

Although the results of Experiments I and II are promising, they are based on artificially created problems especially designed to deliver a proof-of-concept of the Tree-model. The objective motions that had to be learned were designed in such a way that they contained a multi-resolution aspect. These trajectories were generated by a GMM with the same number of Gaussians as the most refined layer of the Tree-model.

A third experiment was designed to deliver proof-of-concept in a more realistic scenario: The double pendulum on a cart swing-up task. This scenario is more realistic since it does not involve learning a pre-defined trajectory. Instead, the objective is to learn to control a unknown dynamical system. The Tree-model and Flat-model have shown to perform equally well. Apparently the multi-resolution aspect of this task was large enough for the Tree-model to be beneficial.

When one would consider Optimization of Layer 2 as a 'flat' model, like we did for experiment 1 and 2, Tree-model exploration, i.e. using Layer 1 & 2, does not outperform the Flat-model (Layer 2). This can be explained by looking at the optimization results of Layer 1 of the Tree-model. Layer 1 does not outperform layer 2 in any stage of the learning process. Hence, first optimizing layer 1 and then optimizing layer 2 does not add any benefit, i.e. using Tree-based exploration is not useful for this particular problem. But again, the loss of performance yielding from the use of the Tree-model is not large.

The results obtained from the Swing-up task is actually counter intuitive; Layer 1 has half the number of parameter compared to Layer 2 and both layers converge to about the same average value. One would expect that Layer 1 would converge faster to the optimal value since the search space is less complex. This is however not the case; both Layer 1 and layer 2 converge equally fast.

The third experiment used the adapted EM algorithm presented in Section 3-1-3. This algorithm optimizes the different layers using soft-clustering. When one compares the fit of the Flat-model to the final layer of the Tree-model, one observes that they are not the same. In fact, the demonstration data set has a lower likelihood to belong to the Tree-model than to belong to the Flat-model. Assuming that the demonstration data provides good information about the skill to be learned, the Flat-model seems to better able to use this prior knowledge. The usage of the Adapted EM-algorithm thus has a negative effect on the fitting of the demonstration data.

Chapter 5

Conclusion

In this thesis a novel multi-resolution policy representation was presented. It was hypothesized that a multi-resolution policy representation increased learning performance in terms of convergence speed and quality of the found solution. The presented approach has a Tree-based structure. The different layers of the Policy are linked together like a Tree. At the root the policy has a very coarse representation. The root branches out, forming the subsequent layers with a more refined policy representation. The Tree-model is constructed in two phases; (i) Construction of the Tree based on Demonstration data, (ii) improvement of the Tree using Exploration.

The parameters of the first layer of the Tree-model are learned using a Standard EM-algorithm. To increase the resolution of the Gaussian Mixture Model (GMM) used to describe the first layer, a second layer is formed by splitting each Gaussian in two. These child Gaussians are fitted to the demonstration data by an adapted EM-algorithm. The adaptation involves taking into account the likelihood of the data to belong to the parent Gaussian. The adapted EM-algorithm is able to describe the data represented by the parent Gaussian using a higher 'resolution'. However, due to this extra 'constraint' the solution of the adapted EM-algorithm might not be globally optimal.

The Tree-model is based on GMM. The properties of GMM are used to construct the Tree-structure. A disadvantage of using GMM, is the relatively large number of parameters required to describe the covariance matrices. We proposed a method to reduce the number of parameters significantly by exploiting the properties of the Gaussian Mixture Regression (GMR) process. This method reduces the number of parameters for exploration from $K\left(1+\frac{d(d+3)}{2}\right)$ to $K\left(d^{\mathcal{O}}(2d^{\mathcal{I}}+1)\right)$. The parameter reduction makes the parameterization no longer quadratic in the dimension of the state-action space. As a price to pay, it is no longer possible to learn all correlations between along the different dimensions. Only the correlation between the input and output parameters is preserved.

The Tree-model was tested in trajectory learning tasks. These experiments showed that the learning using a Tree-based movement representation is advantageous when the objective trajectory inhibits a multi-resolution aspect. However, the trajectories used were created

in such a way that they deliberately contained a multi-resolution aspect. Therefore, these experiments only show that there might be an advantage in the use of a Tree-based movement representation.

The exploration algorithm designed for the Tree-model contains an algorithm that is able to 'project' experience obtained in previous layers to the current layer under exploration. The simulation results showed that the algorithm used to reuse the experience increases learning performance. Especially in cases where the initial exploration noise is badly estimated, the reuse of experience resulted in faster learning.

The Tree-model was also used to learn a double-pendulum-on-a-cart swing-up task. This experiment was selected because it was known from literature that this experiment contained a small multi-resolution aspect. The objective of this task was to improve a feed-forward controller to swing-up the double pendulum to the 'up-up' position while holding the cart on a fixed position. The simulation results have shown that this task can be learned using a Tree-based policy representation. However, the Tree-based movement representation does not outperform a single resolution policy.

The simulation results obtained in the presented experiments give a weak proof that the presented method improves the learning performance. The main reason for calling this proof 'weak', is the fact that we are unable to delivering proof that the Tree-model can achieve faster learning when learning dynamical systems such as the double pendulum swing-up.

Future work

We presented a framework to build a Tree-model based on GMM and focused on the development of an exploration algorithm. The autonomous construction of the Tree was left unexplored. In future work one could investigate the possibilities of autonomous selection of the number of layers and number of states per layer. In addition, one could focus on a Tree-structure that is less homogeneous. In this work we assumed that every parent Gaussian was split in two child Gaussians. But it might be more convenient to split a Gaussian in more than 2 children, or to create only a single child Gaussian. By creating a more flexible tree-structure, higher resolution is only obtained in areas where this is required. This could reduce the number of parameters and result in faster learning. The more flexible tree-structure might also make the Tree-model more generic and better applicable to 'real' problems. In addition, one could investigate the possibility to construct the Tree-model top to bottom, i.e. first defining the most refined layer and then merging Gaussians to construct more coarse layers.

We were unable to deliver sufficient proof to show that the Tree-model yields to faster learning. This is mainly caused by the fact that no application area was found that contained sufficient multi-resolution aspect. Future research could focus on finding such applications.

In the current framework it was assumed that all layers are explored until convergence. One could imagine a scenario where a robot has only a limited number of Roll-outs to improve it's performance. Using these Roll-outs on a parameter space with high complexity might be useless because it requires a large number of roll-outs to shape the covariance noise. In such cases, exploration of a layer of lower resolution might be useful because a faster increase in reward could be obtained in a parameter space of lower complexity.

Appendix A

GMM parameterization

The method proposed in this thesis relies on Gaussian Mixture Model (GMM) and Gaussian Mixture Regression (GMR). To use GMM in an exploration process, a suitable parameterization must be found. In this appendix the parameterization used throughout this thesis is discussed.

First an introduction is given about GMM and GMR in section A-1 and A-2, respectively. Section A-3 shows which parameters of the GMM are required for exploration. Exploration of the GMM requires exploration of the covariance matrix. Exploring the covariance matrix is not straightforward since the covariance matrix is constrained to be positive definite. Section A-4 will discuss and compare different methods that can be used to explore the space of covariance matrices.

A-1 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a linear combination of Gaussian distributions. It can be used to model non-linear behavior by local linear approximations.

To model a linear relation between d random variables X one can use a multivariate Gaussian distribution. The Probability Density Function (pdf) of the Gaussian distribution is given by:

$$\mathcal{P}(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} e^{\frac{1}{2}(\boldsymbol{x}-\mu)\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})^T}, \tag{A-1}$$

with mean μ and covariance Σ. Throughout this thesis the Gaussian distribution of a random variable $\boldsymbol{X}$ will be denoted by $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

The Gaussian distribution can be used to model linear behavior between random variables. To model non-linear behavior between random variables X_i one can use a mixture of Gaussians:

$$\mathcal{P}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{A-2}$$

The K Gaussians are defined by their means $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. The Gaussians are weighted by the mixing coefficient π_k which satisfy the condition to be probabilities:

$$0 \leq \pi_k \leq 1 \ , \quad \sum_{k=1}^{K} \pi_k = 1 \tag{A-3}$$

Comparing (A-2) with the law of total probability:

$$\mathcal{P}(\boldsymbol{x}) = \sum_{k}^{K} \mathcal{P}(k)\, \mathcal{P}(\boldsymbol{x}|k), \tag{A-4}$$

one can conclude that π_k is the prior probability of x belonging to Gaussian k.

For a more detailed description of GMM and its properties the reader is referred to [25].

A-2 Gaussian Mixture Regression

GMM allows to model the joint probability between different random variables X_i. Policy representations require a mapping between input and output variables. Gaussian Mixture Regression (GMR) [44] is a commonly used method to create a mapping between input and output variables given a GMM. This section will introduce the GMR process.

The variables modeled in the GMM are split into input and output variables $\boldsymbol{\xi}^{\mathcal{I}}$ and $\boldsymbol{\xi}^{\mathcal{O}}$, respectively:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\xi}^{\mathcal{I}} \\ \boldsymbol{\xi}^{\mathcal{O}} \end{bmatrix}, \tag{A-5}$$

the GMM is then defined defined by:

$$\boldsymbol{\mu}_k = \begin{bmatrix} \boldsymbol{\mu}_k^{\mathcal{I}} \\ \boldsymbol{\mu}_k^{\mathcal{O}} \end{bmatrix}, \quad \boldsymbol{\Sigma}_k = \begin{bmatrix} \boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}} & \boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}} \\ \boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{I}} & \boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{O}} \end{bmatrix}. \tag{A-6}$$

$\boldsymbol{\mu}^{\mathcal{I}}$ and $\boldsymbol{\mu}^{\mathcal{O}}$ define the mean of the input and output variables. $\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}}$ and $\boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{O}}$ represent the covariance matrices of the input and output variables, respectively. $\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}} = (\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}})^T$ represents a matrix with correlation coefficients between the input and output variables.

A single Gaussian describes the joint probability $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{I}}, \boldsymbol{\xi}^{\mathcal{O}})$ between input and output variables. However, we are interested in the probability of the output variables, given the input variables. Therefore, we calculate the conditional probability $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{O}}|\boldsymbol{\xi}^{\mathcal{I}})$. The conditional probability is again Gaussian and defined by [25]:

$$\boldsymbol{\mu}_k^{\mathcal{O}|\mathcal{I}} = \boldsymbol{\mu}_k^{\mathcal{O}} + \boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{I}} \left(\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}}\right)^{-1} (\boldsymbol{\xi}^{\mathcal{I}} - \boldsymbol{\mu}_k^{\mathcal{I}}), \tag{A-7}$$

$$\boldsymbol{\Sigma}_k^{\mathcal{O}|\mathcal{I}} = \boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{O}} - \boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{I}} \left(\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}}\right)^{-1} \boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}}. \tag{A-8}$$

In GMR, all K Gaussians in the GMM are conditioned to the input variable $\boldsymbol{\xi}^{\mathcal{I}}$, and used to estimate a single Gaussian distribution $\mathcal{N}(\hat{\boldsymbol{\mu}}^{\mathcal{O}}, \hat{\boldsymbol{\Sigma}}^{\mathcal{O}})$. The estimated distribution is defined

by:

$$\hat{\boldsymbol{\mu}}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) = \sum_{k=1}^{K} h_k(\boldsymbol{\xi}^{\mathcal{I}})\boldsymbol{\mu}_k^{\mathcal{O}|\mathcal{I}} \tag{A-9}$$

$$\hat{\boldsymbol{\Sigma}}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) = \sum_{k=1}^{K} h_k^2(\boldsymbol{\xi}^{\mathcal{I}})\boldsymbol{\Sigma}_k^{\mathcal{O}|\mathcal{I}}, \tag{A-10}$$

with weighting coefficient h_k defined as:

$$h_k(\boldsymbol{\xi}^{\mathcal{I}}) = \frac{\pi_k \mathcal{N}(\boldsymbol{\xi}^{\mathcal{I}}|\boldsymbol{\mu}_k^{\mathcal{I}}, \boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}})}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\boldsymbol{\xi}^{\mathcal{I}}|\boldsymbol{\mu}_i^{\mathcal{I}}, \boldsymbol{\Sigma}_i^{\mathcal{I}\mathcal{I}})} \tag{A-11}$$

This weighting coefficient is the probability of Gaussian k being responsible for input $\boldsymbol{\xi}^{\mathcal{I}}$ [9]. This process results in a continuous mapping between input and output variables.

A-3 Exploiting GMR to reduce search space complexity

A GMM is defined by a set of parameters $\{\pi_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^{K}$. The objective is to find a suitable parameterization of the GMM that can be combined with existing exploration algorithms. State of the art exploration algorithms, such as PoWER [32] and PI^2 [35], calculate their parameter update by making linear weighted combinations:

$$\boldsymbol{\theta}^{new} = \sum_{i}^{n} w_i \tilde{\boldsymbol{\theta}}_i, \tag{A-12}$$

The complexity of the search space depends on the number of elements in $\boldsymbol{\theta}$. During exploration one would like to minimize the size of $\boldsymbol{\theta}$ because this keeps the complexity of the search space low. The number of parameters to uniquely define a single Gaussian with dimensionality d is:

Mean	d
Covariance	$\frac{d(d+1)}{d}$
Total	$\frac{d(d+3)}{2}$

For GMM, one additional parameter per Gaussian, the prior π, is required. The total number of parameters required to describe a GMM is thus:

$$K\left(1 + \frac{d(d+3)}{2}\right). \tag{A-13}$$

For a 3 dimensional problem with 6 states it would require 60 parameters to define a GMM. To reduce this number of parameters, knowledge about the GMR process is exploited. GMR is based on calculation of the conditional probability $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{I}}|\boldsymbol{\xi}^{\mathcal{O}})$, which is again Gaussian. When only the regressed trajectory with the highest probability is required, only $\mu^{\mathcal{O}|\mathcal{I}}$ has to be calculated. Rewriting Eq. (A-7) in the more familiar form $y(x) = Ax + b$ yields:

$$\boldsymbol{\mu}_k^{\mathcal{O}|\mathcal{I}}(\boldsymbol{\xi}^{\mathcal{I}}) = \underbrace{\boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{I}}\left(\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{I}}\right)^{-1}}_{A}\underbrace{\boldsymbol{\xi}^{\mathcal{I}}}_{x} + \underbrace{\boldsymbol{\mu}_k^{\mathcal{O}} + \boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{I}}\mu_k^{\mathcal{I}}}_{b}. \tag{A-14}$$

Observe that the behavior of $\boldsymbol{\mu}_k^{\mathcal{O}|\mathcal{I}}$ can be completely controlled by $\boldsymbol{\mu}_k^{\mathcal{O}}$ and $\boldsymbol{\Sigma}^{\mathcal{O}\mathcal{I}}$. The value of A can be fully influenced by $\boldsymbol{\Sigma}_k^{\mathcal{O}\mathcal{I}}$ and thus requires $d^{\mathcal{I}} \times d^{\mathcal{O}}$ parameters. With $d^{\mathcal{I}}$ and $d^{\mathcal{O}}$ being the dimension of the input and the output variables, respectively. The value of b can be fully determined by $\boldsymbol{\mu}_k^{\mathcal{O}}$ which is of dimension $d^{\mathcal{O}}$. Therefore, the maximum number of parameters which is required to fully control the shape of the regressed trajectory is:

$$K(1 + d^{\mathcal{O}}(d^{\mathcal{I}} + 1)). \tag{A-15}$$

In the GMR process, Eq. (A-9) and Eq. (A-10), the prior π_i is used to calculate the weight $h_k(\boldsymbol{\xi}^{\mathcal{I}})$. When the priors have about the same value, one could omit π_k without major consequences to the regression process. This is generally the case when the Gaussians are equally spread over the input space. This would reduce the number of parameters with K to:

$$K(d^{\mathcal{O}}(d^{\mathcal{I}} + 1)). \tag{A-16}$$

Consider again the example given in the introduction of this section. A GMM consisting of 6 states with $d = 3$ can be described with a maximum of 24 parameters (given that $d^{\mathcal{I}} = 1$ and $d^{\mathcal{O}} = 2$). Compared to the 60 parameters which were required to fully describe a GMM this is a immense reduction.

Based on the parameter reduction we define $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \vdots \\ \boldsymbol{\theta}_K \end{bmatrix}, \quad \text{with } \boldsymbol{\theta}_k = \begin{bmatrix} \boldsymbol{\theta}_{\mu_k^{\mathcal{O}}} \\ \boldsymbol{\theta}_{\Sigma_k^{\mathcal{I}\mathcal{O}}} \end{bmatrix} \quad k = 1, \cdots, K. \tag{A-17}$$

$\boldsymbol{\theta}_{\mu_k^{\mathcal{O}}}$ and $\boldsymbol{\theta}_{\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}}}$ are vectors which define the elements $\boldsymbol{\mu}_k^{\mathcal{O}}$ and $\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}}$ of Gaussian k, respectively. For $\boldsymbol{\mu}_k^{\mathcal{O}}$ this parameterization is straightforward:

$$\boldsymbol{\theta}_{\mu_k^{\mathcal{O}}} = \boldsymbol{\mu}_k^{\mathcal{O}} \tag{A-18}$$

The parameterization of $\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}}$ is not straightforward since the elements of $\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}}$ must be in such a way that $\boldsymbol{\Sigma}_k$ remains positive definite. Section A-4 will discuss the parameterization of $\boldsymbol{\Sigma}_k^{\mathcal{I}\mathcal{O}}$.

A-4 Parameterization of the covariance matrix

Covariance matrices of Gaussian distributions are positive definite. A covariance matrix is defined by $d(d-1)/2$ parameters. Section A-3 showed that when one is only interested in the regressed $\hat{\boldsymbol{\mu}}^{\mathcal{O}}$, only the $\boldsymbol{\Sigma}^{\mathcal{I}\mathcal{O}}$ elements of the covariance matrix are required during exploration. Therefore, the number of parameters of the covariance matrix considered during exploration is $d_{\mathcal{I}}d_{\mathcal{O}}$.

Let $\beta = \{\beta_1, \ldots, \beta_{d(d-1)/2}\}$ the set of parameters required to fully define a covariance matrix. And $\boldsymbol{\theta}^{\Sigma^{\mathcal{I}\mathcal{O}}} \subset \beta$ with dimension $d_{\mathcal{I}}d_{\mathcal{O}}$ is defined as the set of parameters required to change $\boldsymbol{\Sigma}^{\mathcal{I}\mathcal{O}}$.

In this chapter we discuss multiple parameterizations of the Covariance matrix which allow to fully control the elements of $\mathbf{\Sigma}^{\mathcal{IO}}$ with $d_{\mathcal{I}}d_{\mathcal{O}}$ parameters. For the parameterizations discussed in this chapter it is important that the input and output variables of the covariance matrix are ordered such that the covariance matrix has the following form:

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}^{\mathcal{II}} & \mathbf{\Sigma}^{\mathcal{IO}} \\ \mathbf{\Sigma}^{\mathcal{OI}} & \mathbf{\Sigma}^{\mathcal{OO}} \end{bmatrix}.$$

A-4-1 Constrained and un-constrained parameterization

The covariance matrix is a symmetric Positive Definite (PD) matrix. This means that all eigenvalues of the covariance matrix are strictly positive, i.e. greater than zero. There are two types of parameterizations of the covariance matrix, constrained and unconstrained parameterization. In constrained parameterization the values of the parameters are forced to lie within certain bounds. These bounds ensure that the covariance matrix remains positive definite. The unconstrained parameterization has no bounds on the parameters and therefore guarantees Positive Definiteness intrinsically.

Combining constrained parameterization with exploration algorithms is likely to result in violation of constraints: Samples created by the exploration algorithm could lie outside the bounds set by the exploration algorithm.

This can be solved in two ways: (i) Draw new samples until all requires samples lie within the bounds (ii) Use a heuristic to force the out of bound samples to lie within the bound. Both methods will influence the nature of the sampling distribution. Hence we prefer an unconstrained parameterization.

A-4-2 Related work

In the fields of modeling longitudinal-data and multivariate analysis estimating covariance matrices is an important aspect [45]. Various methods have been investigated to estimate the covariance matrices from large sets of (incomplete) data. Among these methods are parameterizations of covariance matrices. These methods are mainly based on spectral decomposition and Cholesky decomposition of the covariance matrix.

Spectral decomposition

The spectral decomposition is defined as follows:

$$\Sigma = V\Lambda V^T, \tag{A-19}$$

with V as a diagonal matrix containing the Eigenvectors of Σ and Λ a diagonal matrix with the Eigenvalues λ corresponding to the columns of V. The spectral decomposition does not provide a way to select parameters that only control $\Sigma^{\mathcal{IO}}$.

Cholesky decomposition

In [42], Pinheiro and Bates discuss two parameterizations of the covariance matrix based on the Cholesky decomposition. The Cholesky decomposition decomposes a positive definite matrix in a triangular matrix. Since Σ is always positive definite for a Gaussian distribution its decomposition always exists:

$$\Sigma = LL^T \tag{A-20}$$

were L is a lower triangular matrix. This decomposition provides a relatively simple relation between the parameters and the Covariance matrix.

Pinheiro and Bates argue that, although this decomposition is computationally stable and simple, it does not provide a good statistical interpretation of the individual parameters. The main concern in this work is the ability to control fully control the value of Σ^{IO} with only $d^{\mathcal{I}}d^{\mathcal{O}}$ parameters. The simple relation between L and Σ provides the possibility to do this. Statistical interpretation is less important in this case.

The parameters of the Cholesky parameterization [42] are all non-zero elements of the L matrix. Given:

$$L = \begin{bmatrix} \zeta_1 & 0 & 0 & \dots & 0 \\ \psi_1 & \zeta_2 & 0 & \dots & 0 \\ \psi_2 & \psi_d & \ddots & & \vdots \\ \vdots & \vdots & & \zeta_{d-1} & 0 \\ \psi_d & \psi_{2d-3} & \dots & \psi_{d(d-1)/2} & \zeta_d \end{bmatrix}. \tag{A-21}$$

the complete set of parameters β to parameterize the covariance matrix is:

$$\beta = \left\{\zeta_1, \cdots \zeta_d, \psi_1, \cdots, \psi_{d(d-1)/2}\right\} \tag{A-22}$$

The parameters responsible for $\Sigma^{\mathcal{IO}}$ are the elements at the entries L_{ij} with $i \in \mathcal{I}$ and $j \in \mathcal{O}$. These are only the off-diagonal elements, i.e. $\phi \subset \{\psi_1, \cdots, \psi_{d(d-1)/2)}\}$. The values for L_{ij} are calculated by:

$$L_{ij} = \frac{\Sigma_{ij} + \sum_{k=1}^{d} \mathrm{f}(i,j,k)}{L_{jj}}, \quad \text{with } \mathrm{f}(i,j,k) = \begin{cases} L_{i,k}L_{j,k} & k \neq j \\ 0 & k = j \end{cases} \tag{A-23}$$

The Cholesky decomposition is not unique. If L is the Cholesky decomposition so is a matrix which is obtained by multiplying any subset of the rows of L with -1. Uniqueness can be achieved by forcing the diagonal elements of L to be positive. This can be achieved by replacing ζ_i by $\ln(\zeta_i)$ in β [42]. Since we are only interested in varying a subset of ψ_i this operation is not required in our algorithm.

Pourahmadi decomposition

Pourahmadi [46, 47] proposed a parameterization which is related to the Cholesky decomposition of the *inverse* covariance matrix. For a positive definite covariance matrix Σ there exist

a diagonal matrix D with positive entries and a lower triangular matrix T with unit diagonal such that

$$T\Sigma T^T = D \text{ or } \Sigma^{-1} = T^T D^{-1} T \tag{A-24}$$

with D as a diagonal matrix and T a lower unitriangular matrix [1]. Throughout this work this decomposition is referred to as the Pourahmadi decomposition. The Pourahmadi parameterization of a d dimensional matrix Σ is given by:

$$\beta = \{\psi_1, \cdots, \psi_{d(d-1)/2}, \ln(\zeta_1), \ldots, \ln(\zeta_d)\} \tag{A-25}$$

with ψ_i representing the lower off-diagonal elements of T and ζ_i the diagonal elements of D. ln() forces ζ to remain positive. The use of the ln() function has consequences for the sampling behaviour as shown in Figure A-1. During exploration samples are drawn from a normal distribution. To convert the sampled θ to σ we need to take the exponential of the samples. This yields to completely different distribution of the samples.

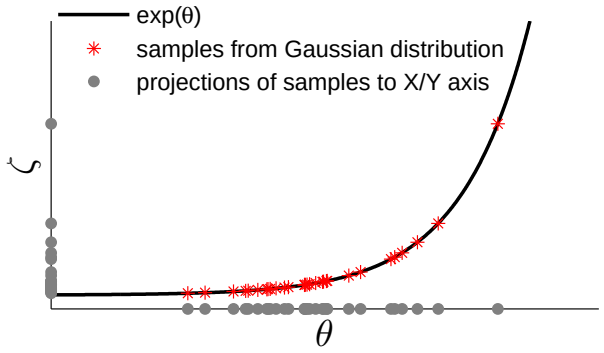


Figure A-1: The Pourahmadi decomposition parameterizes variance of the variables ζ by $\theta = \ln(\zeta)$. Sampling θ with a normal distribution $\mathcal{N}(0, \sigma^{noise})$ yields samples of ζ with $\exp(\mathcal{N}(0, \sigma^{noise}))$. ζ is thus sampled from a skew distribution a skew distribution.

The Pourahmadi decomposition is based on incremental linear least-squares (ILLS) prediction. In the ILLS one tries to predict a value y_t given its predecessors $y_{t-1}, \cdots, y_1$. This is done using the following relation:

$$\hat{y}_t = \mu_t + \sum_j \phi_{t,j}(y_j - \mu_j). \tag{A-26}$$

The values contained in D and T have statistical meaning related to this process. The diagonal elements of D are prediction error variances and the strictly lower triangular elements of T are the negatives of the auto-regression coefficients $\phi_{t,j}$. This statistical interpretation of the parameters explains why ζ_i must remain positive; the error variance cannot be negative.

Due to the fact that the Pourahmadi parameterization is defined as the inverse of the covariance matrix, it is not straightforward to derive a general expression for the covariance matrix in terms of β. To give an illustration of relationships between the different parameters a 3×3 example is given. Consider D and T:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ -\psi_1 & 1 & 0 \\ -\psi_2 & -\psi_3 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} \zeta_1^2 & 0 & 0 \\ 0 & \zeta_2^2 & 0 \\ 0 & 0 & \zeta_3^2 \end{bmatrix} \tag{A-27}$$

[1] A unitriangular matrix is a matrix with ones on the diagonal.

Using Eq. (A-24) Σ^{-1} is calculated, inverting the result yields:

$$\Sigma = \begin{bmatrix} \zeta_1^2 & \zeta_1^2\psi_1 & \zeta_1^2(\psi_2+\psi_1\psi_3) \\ \zeta_1^2\psi_1 & \zeta_x^2\psi_1^2+\zeta_2^2 & \zeta_1^2(\psi_3\psi_1^2+\psi_2\psi_1)+\zeta_2^2\psi_3 \\ \zeta_1^2(\psi_2+\psi_1\psi_3) & \zeta_1^2(\psi_3\psi_1^2+\psi_2\psi_1)+\zeta_2^2\psi_3 & \zeta_1^2\left(\psi_1^2\psi_3^2+2\psi_1\psi_2\phi_3+\psi_2^2\right)+\zeta_2^2\psi_3^2+\zeta_3^2 \end{bmatrix} \tag{A-28}$$

Notice that the dependence between the variables increases as we move through the different entries of the matrix from the top left to the bottom right. When the structure of the covariance matrices is ordered in such a way that the input and output components are as given in Eq. (A-25), it is possible to select those autoregressive coefficients that only change $\Sigma^{\mathcal{IO}}$. When we consider for example the dimension 1 as input and dimension 2 and 3 as output, we can select regression coefficients ϕ_1 and $\phi 2$ or ϕ_3.

Note that it is not required to use ζ in the parameterization; this omits the concern about changed sampling distribution due to the ln() function.

A-4-3 Comparing basic Cholesky and Pourahmadi parameterization

Both the Cholesky decomposition and the Pourahmadi decomposition offer a way to parameterize the covariance matrix allowing a change of $\Sigma^{\mathcal{IO}}$ using $d^{\mathcal{I}}d^{\mathcal{O}}$. In this section we compare the performance of both methods.

Analytic comparison

First a comparison between the two methods is made by analyzing the Covariance matrix of both methods for a 3 dimensional example. To fully describe the covariance matrix one needs $3(3+1)/2 = 6$ parameters. The set of parameters β is defined as:

$$\beta = \{\psi_1, \psi_2, \psi_3, \zeta_1, \zeta_2, \zeta_3\} \tag{A-29}$$

The 3×3 Covariance matrix obtained by the inverse Cholesky parameterization is given by:

$$\Sigma = \begin{bmatrix} \zeta_1^2 & \psi_1\zeta_1 & \psi_2\zeta_1 \\ \psi_1\zeta_1 & \psi_1^2+\zeta_2^2 & \psi_1\psi_2+\psi_3\zeta_2 \\ \psi_2\zeta_1 & \psi_1\psi_2+\psi_3\zeta_2 & \psi_2^2+\psi_3^2+\zeta_3^2 \end{bmatrix} \tag{A-30}$$

The the covariance matrix expressed by the Pourahmadi parameterization is given in Eq. (A-28). Note that the values ζ_i and ψ_i in Eq. (A-28) and Eq. (A-30) do, except for trivial cases, not have the same numerical value.

An element by element comparison shows that the Pourahmadi parameterization has a more complex relation between the ψ_i and the off-diagonal elements of the covariance matrix than the Cholesky decomposition. Consider for example element $\Sigma^{\mathcal{IO}} = [\Sigma_{13}, \Sigma_{23}]^T$. In the Pourahmadi decomposition the elements are defined as:

$$\Sigma_{13} = \psi_2\zeta_1^2 + \psi_3\psi_1\zeta_1^2 \tag{A-31}$$

$$\begin{aligned} \Sigma_{23} &= \zeta_1^2\left(\psi_3\psi_1^2+\psi_2\psi_1\right)+\zeta_2^2\psi_3 \\ &= \psi_2\left(\psi_1\zeta_1^2\right)+\psi_3\left(\zeta_2^2+\psi_1^2\zeta_1^2\right) \end{aligned} \tag{A-32}$$

In the Cholesky decomposition the elements are define as:

$$\Sigma_{13} = \psi_2\zeta_1 \tag{A-33}$$
$$\Sigma_{23} = \psi_2\psi_1 + \psi_3\zeta_2 \tag{A-34}$$

For both the Pourahmadi and the Cholesky it is possible to obtain a linear relation between the parameters $\boldsymbol{\theta}_{\Sigma^{\mathcal{IO}}}$. When selecting $\boldsymbol{\theta}_{\Sigma^{\mathcal{IO}}} = \{\psi_2, \psi_3\}$ a linear relation between ϕ_i is obtained for both the Cholesky and Pourahmadi decomposition. Although the Pourahmadi parameterization has more complex relations between the complete set of parameters β both Σ_{13} and Σ_{23} are linear expressions in terms of ψ_2 and ψ_3.

Having a linear relation between the parameters defining $\boldsymbol{\theta}_{\Sigma^{\mathcal{IO}}}$ is advantageous in the exploration process. Exploration is based on a Gaussian distribution $\mathcal{N}(0, \Sigma^{noise})$. The exploration noise Σ^{Noise} determines the 'direction' of exploration. Σ^{noise} encodes the dependence between θ_i in a linear fashion. When the actual relation between θ_i is indeed a linear relation, the estimation of the exploration direction will be more accurate.

The only difference between the Cholesky decomposition and Pourahmadi decomposition is the magnitude of ψ_i. The Pourahmadi decomposition has ζ_i^2 in the off-diagonal elements where the Cholesky has ζ_i in the off-diagonal elements. The magnitude of ψ_i and the sensitivity of $\boldsymbol{\Sigma}$ to changes in ψ_i is expected to be different for the Pourahmadi decomposition and Cholesky decomposition. This will require a different initialization of the exploration noise Σ^{noise}.

Experimental evaluation

To find out the difference between the Pourahmadi and Cholesky parameterization an experiment is done in which both parameterizations are used to in a Learning from Exploration (LfE) setting.

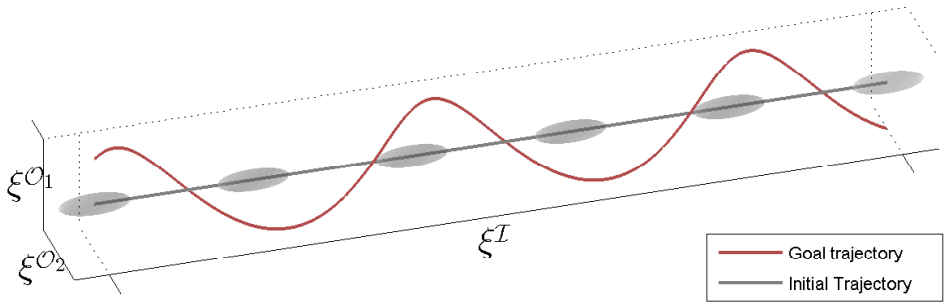


Figure A-2: An illustration of learning problem considered to compare the learning performance of the Pourahmadi and Cholesky decomposition. In red, goal trajectory which has the shape of a spiral. The initial state of the GMM is shown by the gray ellipsoids, each representing one Gaussian. The gray line represents the regressed trajectory.

Experimental set-up The experiment consist of a simulation. The objective of the experiment is to learn a spiral trajectory as shown in Figure A-2. The trajectory is given by:

$$\begin{bmatrix} \xi_g^{\mathcal{O}_1} \\ \xi_g^{\mathcal{O}_2} \end{bmatrix} = \begin{bmatrix} \sin(\boldsymbol{\xi}^{\mathcal{I}}) \\ \cos(\boldsymbol{\xi}^{\mathcal{I}}) \end{bmatrix} \tag{A-35}$$

A $K = 6$ states GMM was constructed to represent the motion. The initial condition of the states are given by:

$$\mu_k = \begin{bmatrix} (k-1)\pi \\ 0 \\ 0 \end{bmatrix} \ , \ \Sigma_k = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \ , \ \pi_k = \frac{1}{K} \tag{A-36}$$

The initial state of the GMM and the corresponding regressed trajectory is displayed in Figure A-2.

The exploration algorithm used is an adjusted version of the PoWER as used in [48]. The performance of each episode is evaluated based on the mean squared error:

$$r = e^{-\alpha\epsilon} \ , \quad \epsilon = \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{\xi}_g^{\mathcal{O}} - \boldsymbol{\xi}^{\mathcal{O}} \right)^T \left(\boldsymbol{\xi}_g^{\mathcal{O}} - \boldsymbol{\xi}^{\mathcal{O}} \right) \tag{A-37}$$

The exponential function forces the reward to lie within the the range $(0, 1]$. The tuning factor α was manually tuned to 5.

The settings for the experiment are given in table A-1. The initial exploration noise for both parameterizations is set to the same value. The minimum exploration noise, required to prevent early convergence, is set as a fraction $(1e{-}3)$ of the Initital exploration noise.

Table A-1.

Variable	Value
Number of episodes	1000
Number of episodes before first update	20
Number of points considered for regression	10
Initial Exploration noise Cholesky $(\mu^{\mathcal{O}_1}, \mu^{\mathcal{O}_1}, \phi_1, \phi_2)$	diag $([0.15, 0.5, 0.5, 0.1, 0.1])$
Exploration noise Pourahmadi $(\mu^{\mathcal{O}_1}, \mu^{\mathcal{O}_1}, \phi_1, \phi_2)$	diag $([0.15, 0.5, 0.5, 0.1, 0.1])$

Table A-1: A summary of the exploration setting used during the experiment that compared Pourahmadi parameterization with Cholesky parameterization in Appendix A-4

Results The results of the simulations are summarized in Figure A-3. Both the Pourahmadi and Cholesky parameterization have about the same learning behavior. Both methods learn about equally fast and converge to the same value. Based on this experiment we cannot conclude there is a difference in performance.

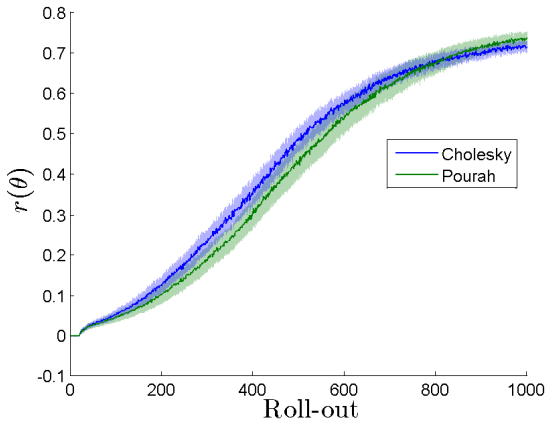


Figure A-3: The learning performance of the Pourahmadi and Cholesky parameterization when learning the objective motion displayed in figure A-2.

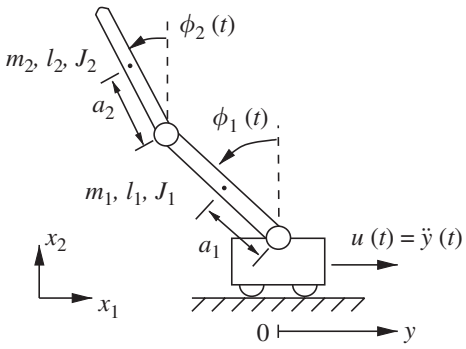
Conclusion

Both the Pourahmadi and Cholesky decomposition are suitable to for unconstrained covariance parameterization. No clear advantage has been found for one method to be in favor of the other. In this thesis we used the method based on Cholesky decomposition.

Appendix B

Double pendulum on a Cart: Equations of Motion

Figure B-1 show the schematic represenation of the double pendulum on a cart. In this appendix the Equations of Motion (EOM) are derived using Schwab's TMT method [49].



Property	symbol	$i=1$	$i=2$
Mass	$m_i[kg]$	0.853	0.510
Inertia	$J_i[kg \cdot m^2]$	0.0126	0.0185
Length	$L_i[m]$	0.323	0.480
Dist. to c.o.m. [a]	$a_i[m]$	0.2145	0.223
Viscous friction	$d_i[N \cdot m/s]$	0.005	0.005

[a] Center of Mass

Figure B-1: Left: Schematic representation of the double pendulum on a cart. Right: Physical properties of the double pendulum on a cart. (Image adopted from [7])

Given the state vector:

$$\boldsymbol{q} = \begin{bmatrix} \phi_1 & \phi_2 & y \end{bmatrix}^T, \tag{B-1}$$

we define Transformation function $T(\boldsymbol{q})$ that decribes the location of the center's of mass in

terms of the state vector $\boldsymbol{q}$:

$$\boldsymbol{T}(\boldsymbol{q}) = \begin{bmatrix} \phi_1 \\ x_{1,1} \\ x_{2,1} \\ \phi_2 \\ x_{1,2} \\ x_{2,2} \\ y \end{bmatrix} = \begin{bmatrix} \phi_1 \\ a_1 \sin(\phi_1) \\ a_1 \cos(\phi_1) \\ \phi_2 \\ l_1 \sin(\phi_1) a_2 \sin(\phi_2) \\ l_1 \cos(\phi_1) a_2 \cos(\phi_2) \\ y \end{bmatrix}, \tag{B-2}$$

Define the mass-matrix as:

$$\boldsymbol{M} = \text{diag}\left([J_1, m_1, m_1, J_2, m_2, m_2, m_{cart}]\right), \tag{B-3}$$

the friction:

$$\boldsymbol{f} = \begin{bmatrix} d_1 & d_2 & 0 \end{bmatrix}^T \dot{\boldsymbol{q}} \tag{B-4}$$

and finally the gravity vector:

$$\boldsymbol{g} = \begin{bmatrix} 0 & 0 & -9.81 & 0 & 0 & -9.81 & 0 \end{bmatrix}^T \tag{B-5}$$

Define J as the Jacobian of $\boldsymbol{T}$, i.e. $\boldsymbol{J} = \text{Jac}(\boldsymbol{T})$; And $\boldsymbol{c} = \text{Jac}(\boldsymbol{J}\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ as the convective forces. The equations of motions are given by:

$$\underbrace{\boldsymbol{J}^T M \boldsymbol{J}\ddot{\boldsymbol{q}}}_{\bar{M}} = -\boldsymbol{J}^T \boldsymbol{M}\left(\boldsymbol{c} + \boldsymbol{g}\right) + \boldsymbol{f} \tag{B-6}$$

For the experiment in Section 4-3, it is assumed that the input of the system is $\ddot{y}$. Hence, we can omit the dynamics of the cart since the movement profile of the cart is directly derived from $\ddot{y}$. The terms relating to the pendula $\bar{\boldsymbol{M}}_{31}\ddot{y}$ and $\bar{\boldsymbol{M}}_{32}\ddot{y}$ are added to the corresponding EOM as inputed force. This yields to the two remaining EOM:

$$\begin{bmatrix} J_1 + a_1^2 m_1 + l_1^2 m_2 & a_2 l_1 m_2 \cos(\phi_1 - \phi_2) \\ a_2 l_1 m_2 \cos(\phi_1 - \phi_2) & J_2 + a_2^2 m_2 \end{bmatrix} \begin{bmatrix} \ddot{\phi}_1 \\ \ddot{\phi}_2 \end{bmatrix} = \tag{B-7}$$

$$\begin{bmatrix} (a_1 m_1 + l_1 m_2) g \sin(\phi_1) - a_2 l_1 m_2 \sin(\phi_1 - \phi_2)\dot{\phi}_2^2 - d_1\dot{\phi}_1 - d_2(\dot{\phi}_1 - \dot{\phi}_2) + (a_1 m_1 + l_1 m_2)\cos(\phi_1)\ddot{y} \\ a_2 l_1 m_2 \sin(\phi_1) - \phi_2)\dot{\phi}_1^2 + d_2(\dot{\phi}_1 - \dot{\phi}_2) + a_2 m_2 \cos(\phi_2)\ddot{y} \end{bmatrix}$$

Appendix C

Full Experimental Results

This appendix shows all the experimental results obtained for the experiments described in Chapter 4. Each experiment described in Chapter 4 is treated in a separate section. For a description of the experimental set-up the reader is referred to the corresponding experiments in Chapter 4.

The results are reported using 3 different plot-types:

- **Learning performance (per Noise setting)**; The reward per noise setting plot shows the reward increase over the number of Roll-outs including an 95% confidence interval. Each plot shows one noise setting, but all layers with the same setting in the same plot.
- **Learning performance (per Layer)**; The reward per layer setting plot shows the reward increase over the number of Roll-outs including an 95% confidence interval. Each layer (Layer 1, Layer 2, All layer) has a different plot. All noise settings are plotted in the same plot.
- **t-test**; Show the result of the statistical test comparing situations 'Flat-model' and 'Tree-model'. The null hypothesis H_0 indicates that both situations perform equally well. The alternative hypothesis, H_1, indicates that 'All layer' outperforms 'Layer 2'. The p-value indicates the confidence in H_0 in favor of H_1.
- **Switching behavior**; Displays the Roll-out number at which the Tree-model switches from layer 1 to layer 2 when using Tree-based exploration. The results of all observations are displayed in a boxplot.

C-1 Experiment I

In this Section all results are given of the parameter study described in Section 4-1-1. The experiment consisted of learning 4 different motions. Two series of experiments were carried

out. The first series of experiments were used to determine the optimal minimum noise Settings. The Results of these experiments are given in C-1-1.

The second set of experiments are used to compare the performance of the Flat-model with the Tree-model for different initial noise settings. In the second set of experiments the optimal minimum noise Setting of the first Series of experiments is used. The results of the second series of experiments is given in C-1-2.

All settings were tested by learning 4 different trajectories. These trajectories are displayed in Figure C-1.

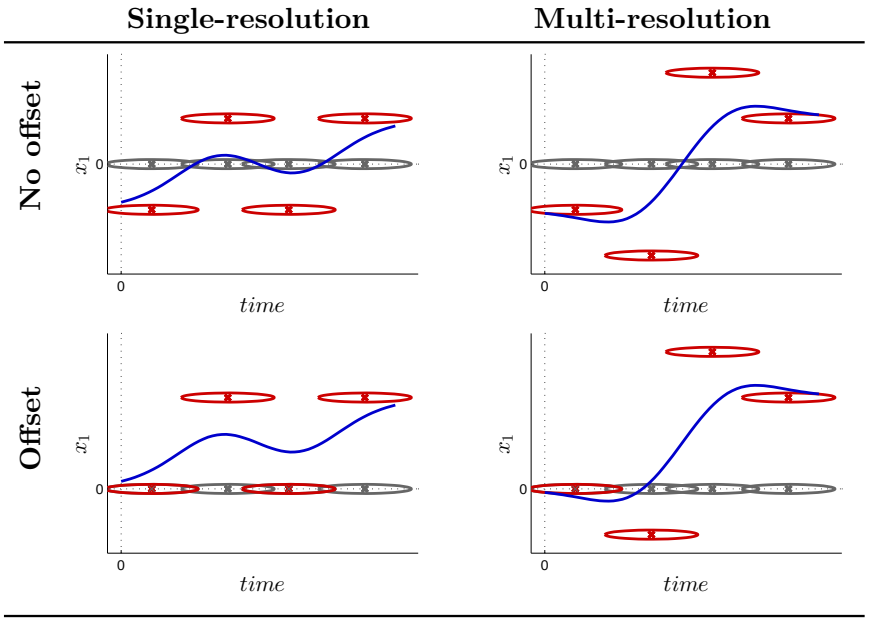
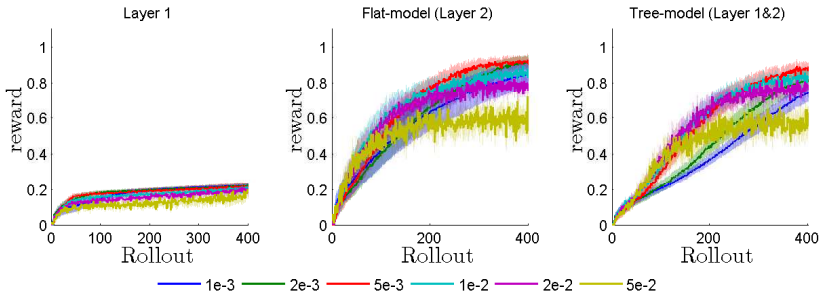


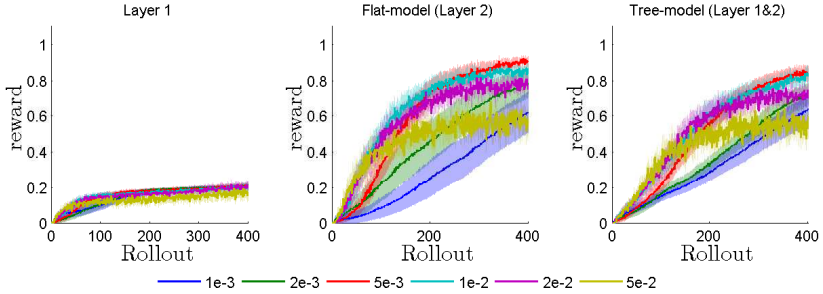
Figure C-1: Four 2-dimensional trajectories generated by a 4 state GMM (displayed in red). The Gaussian are placed such that they do not contain a multi-resolution aspect (left column) or do contain a multi-resolution aspect (right column). In addition an offset between the optimal trajectory and the the initial condition of the GMM is introduced. The initial state of the GMM is centered with respect to the optimal GMM. In contrast, the initial state of the GMM in the bottom row has as bias with respect to the optimal solution.

C-1-1 Results: Minimum Noise Experiments

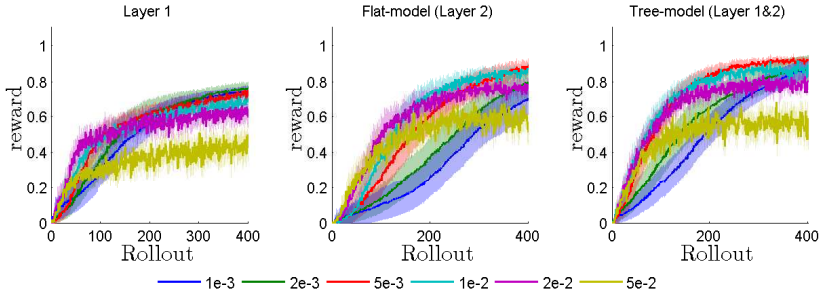
The minimum noise experiments are done to determine the optimal minimum noise setting. Figure C-2 shows the results of this experiment splitted per layer.



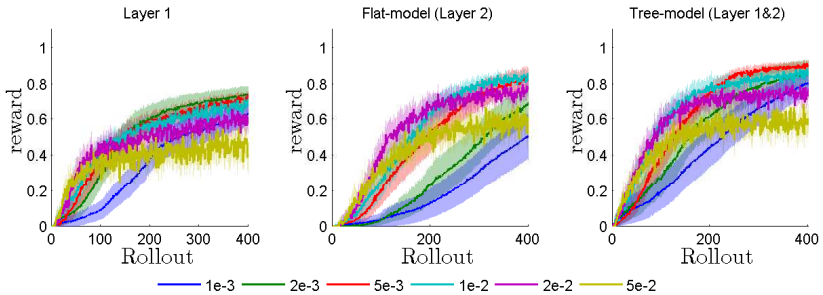
(a) Single resolution no Offset



(b) Single resolution Offset



(c) Multi resolution no Offset

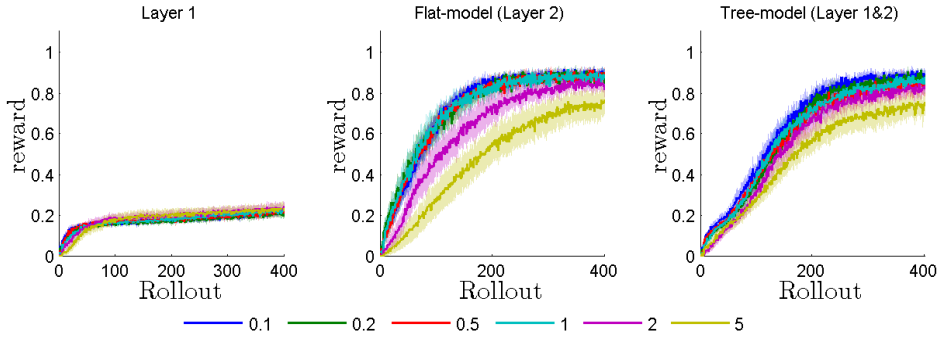


(d) Multi resolution Offset

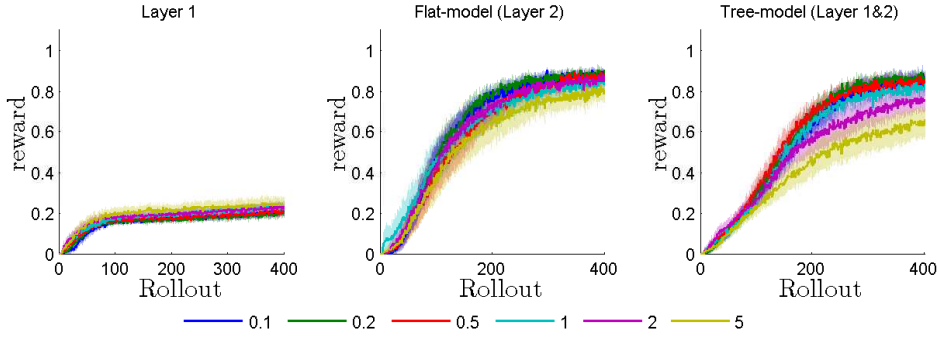
Figure C-2: Figure shows learning performance of 4 different movements (a-d) using a 2 layer Tree-model. Each movement (a-d) is learned by exploring only Layer 1, exploring only Layer 2 (Flat-model) or exploring both Layer 1 and 2 (Tree-model). The different lines in the graphs indicate different minimum noise settings. The numbers in the legend represent the minimum noise fraction ϵ_{Noise} of the Initial noise matrix $\Sigma_{Noise} = \text{diag}([0.1, 1])$.

C-1-2 Results: Initial Noise Experiments

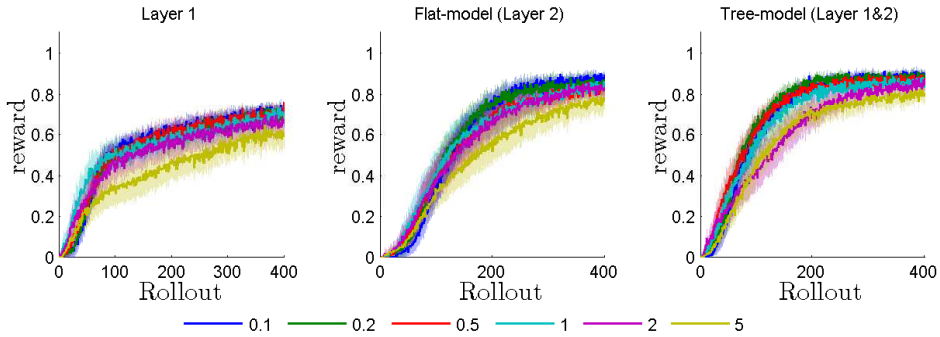
Results per layer



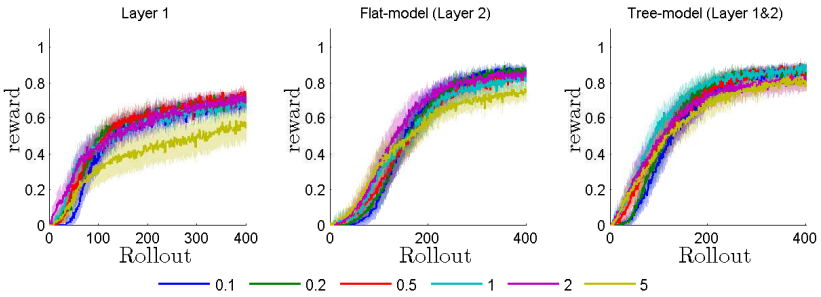
(a) Single resolution no Offset



(b) Single resolution Offset



(c) Multi resolution no Offset



(d) Multi resolution Offset

Figure C-2: Figure shows learning performance of 4 different movements (a-d) using a 2 layer Tree-model. Each movement (a-d) is learned by exploring only Layer 1, exploring only Layer 2 (Flat-model) or exploring both Layer 1 and 2 (Tree-model). The different lines in the graphs indicate different initial noise settings. The numbers in the legend represent the initial noise fraction ϵ_{Noise} of the Initial noise matrix $\Sigma_{Noise} = \text{diag}([0.1, 1])$. The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

Comparison: Single Resolution no Offset

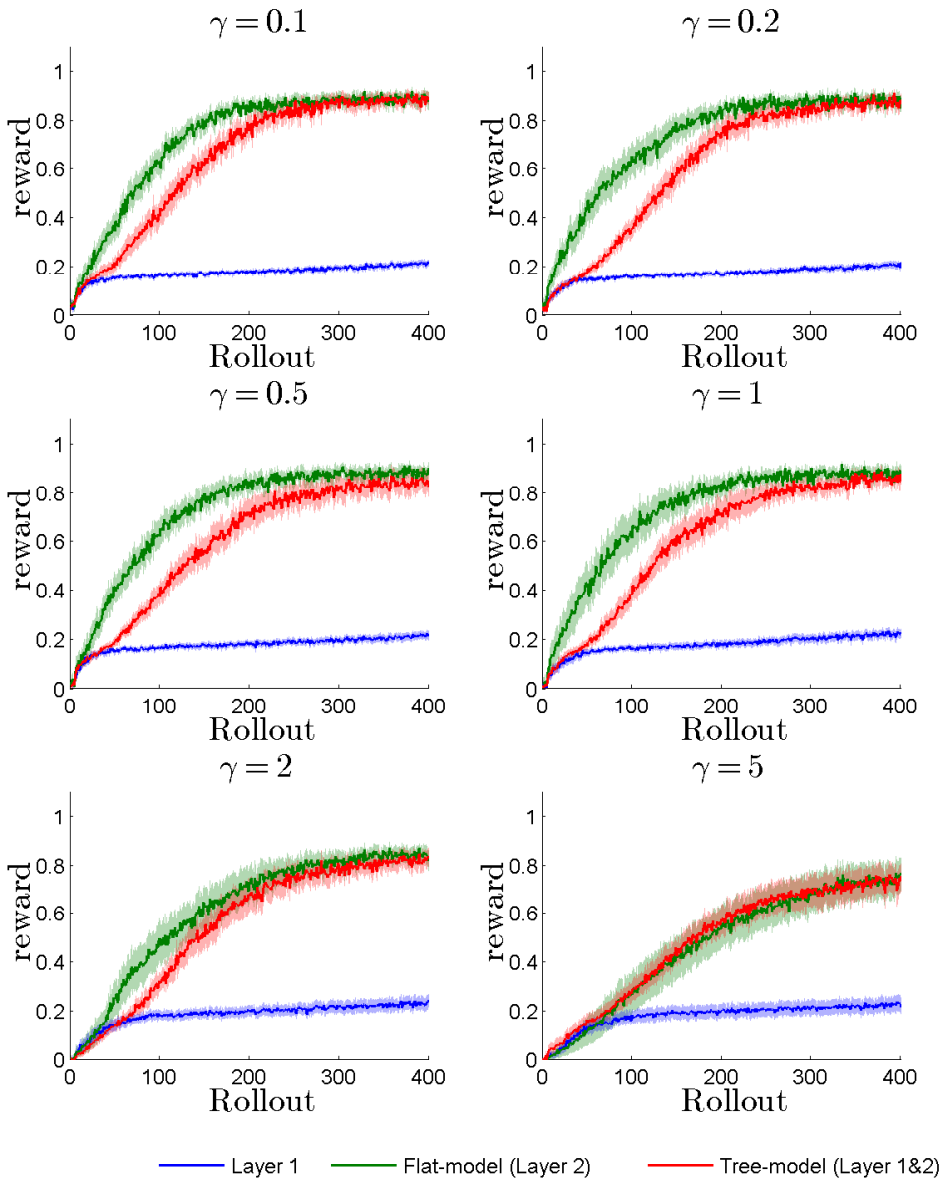


Figure C-3: The Figure shows the learning performance of a 2 Layer Tree-model while learning the Single resolution movement without an offset (shown in Figure C-1) for different initial noise settings. Each graphs gives the results for a different initial noise settings. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

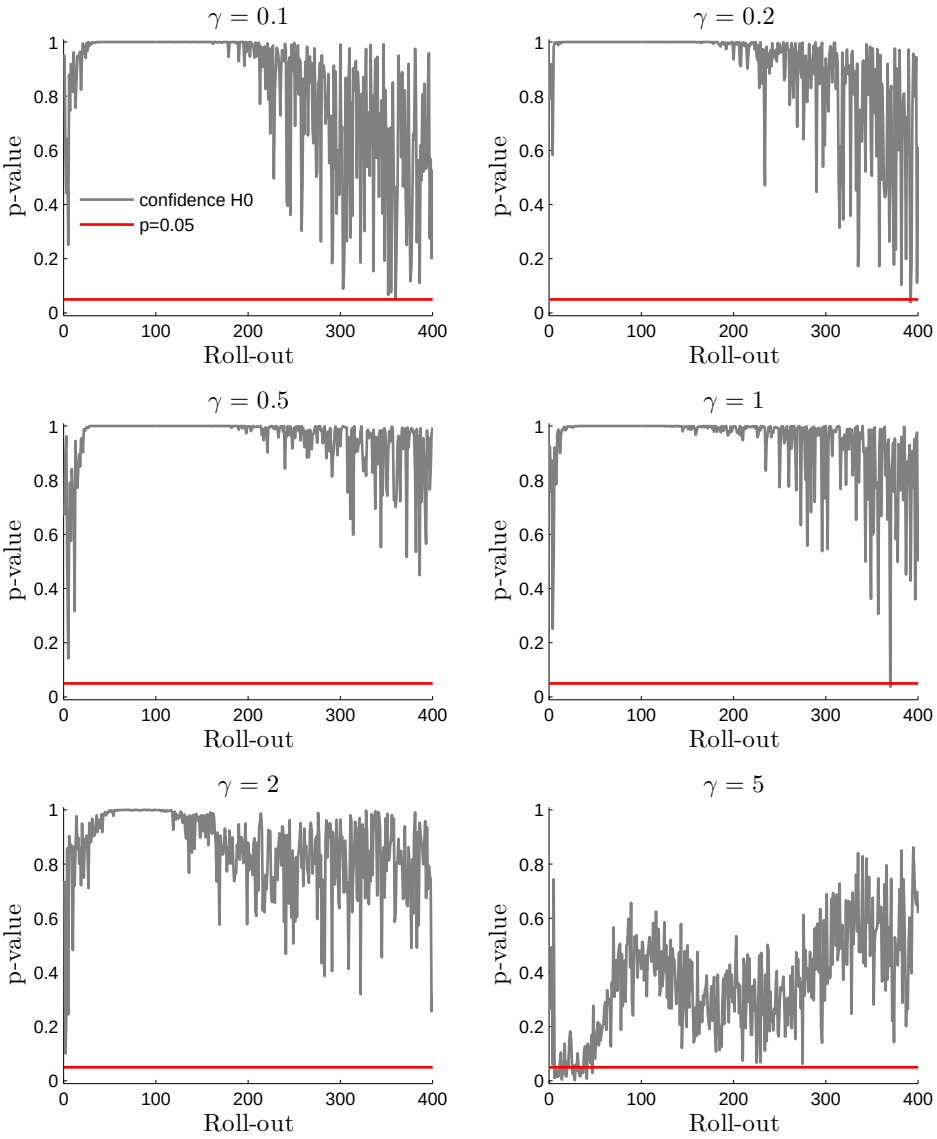


Figure C-4: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning the single-resolution trajectory without an Offset (shown in Figure C-1) for different noise settings. The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flatmodel. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}\left([0.1, 1]\right)\gamma$.

Comparison: Single Resolution Offset

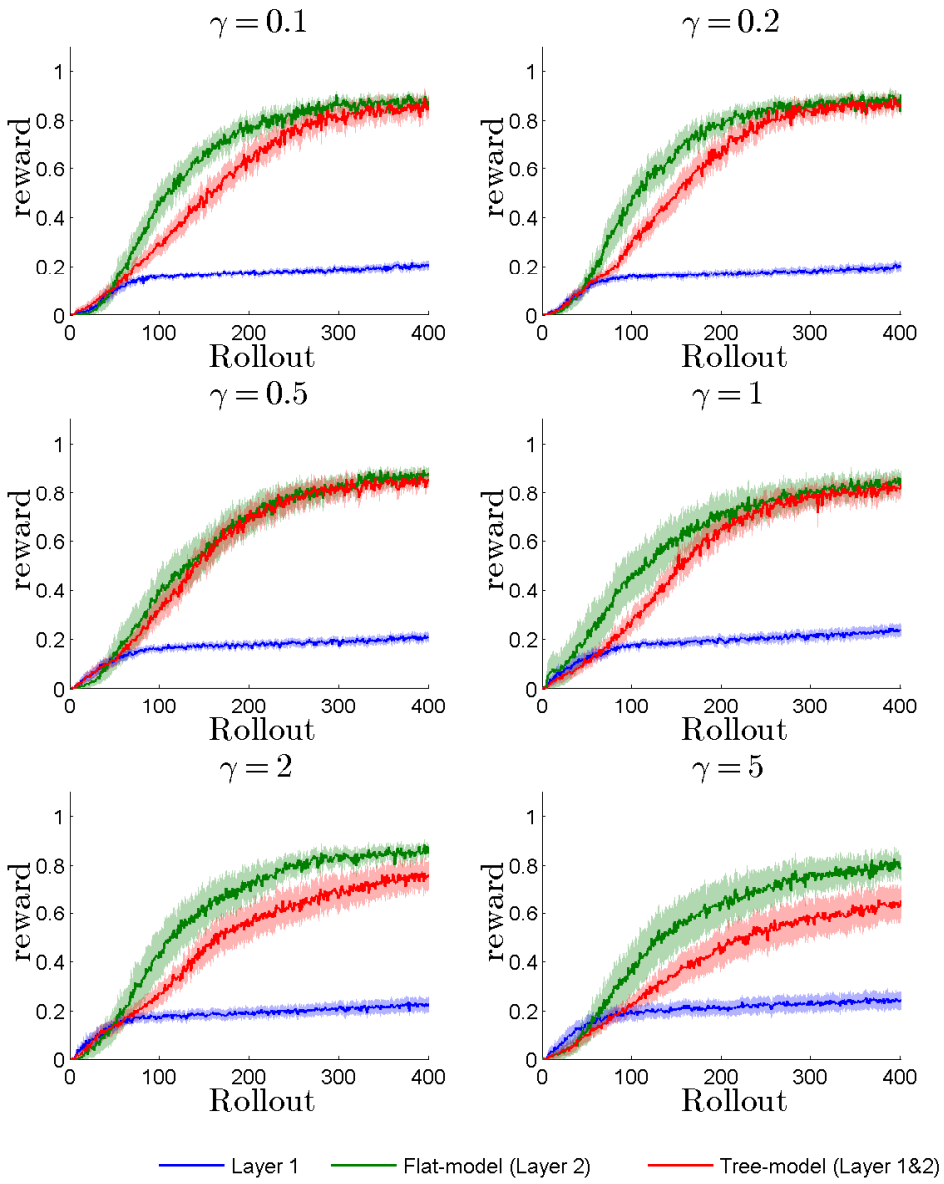


Figure C-5: The Figure shows the learning performance of a 2 Layer Tree-model while learning the Single resolution movement with an offset (shown in Figure C-1) for different initial noise settings. Each graphs gives the results for a different initial noise settings. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

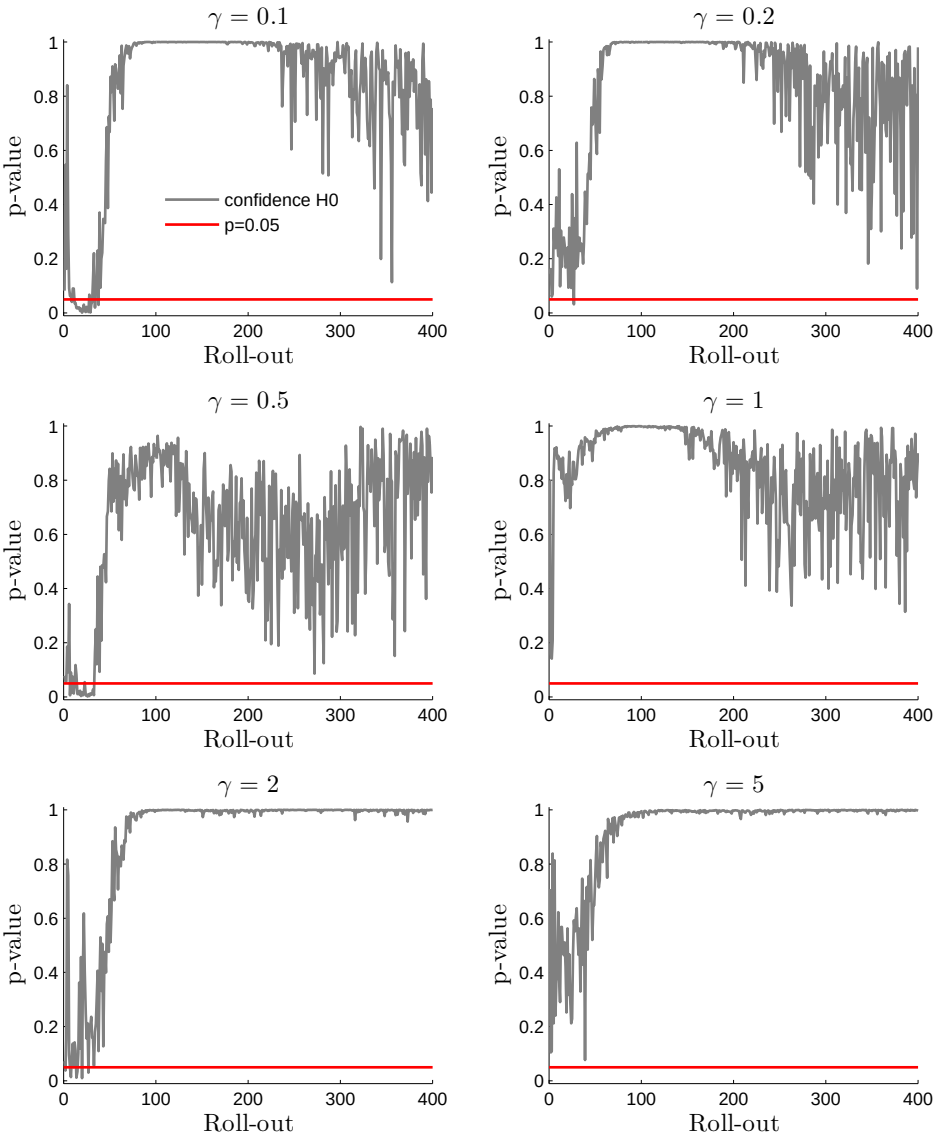


Figure C-6: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning the single-resolution trajectory with an Offset (shown in Figure C-1) for different noise settings. The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flatmodel. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}\left([0.1, 1]\right)\gamma$.

Comparison: Multi Resolution no Offset

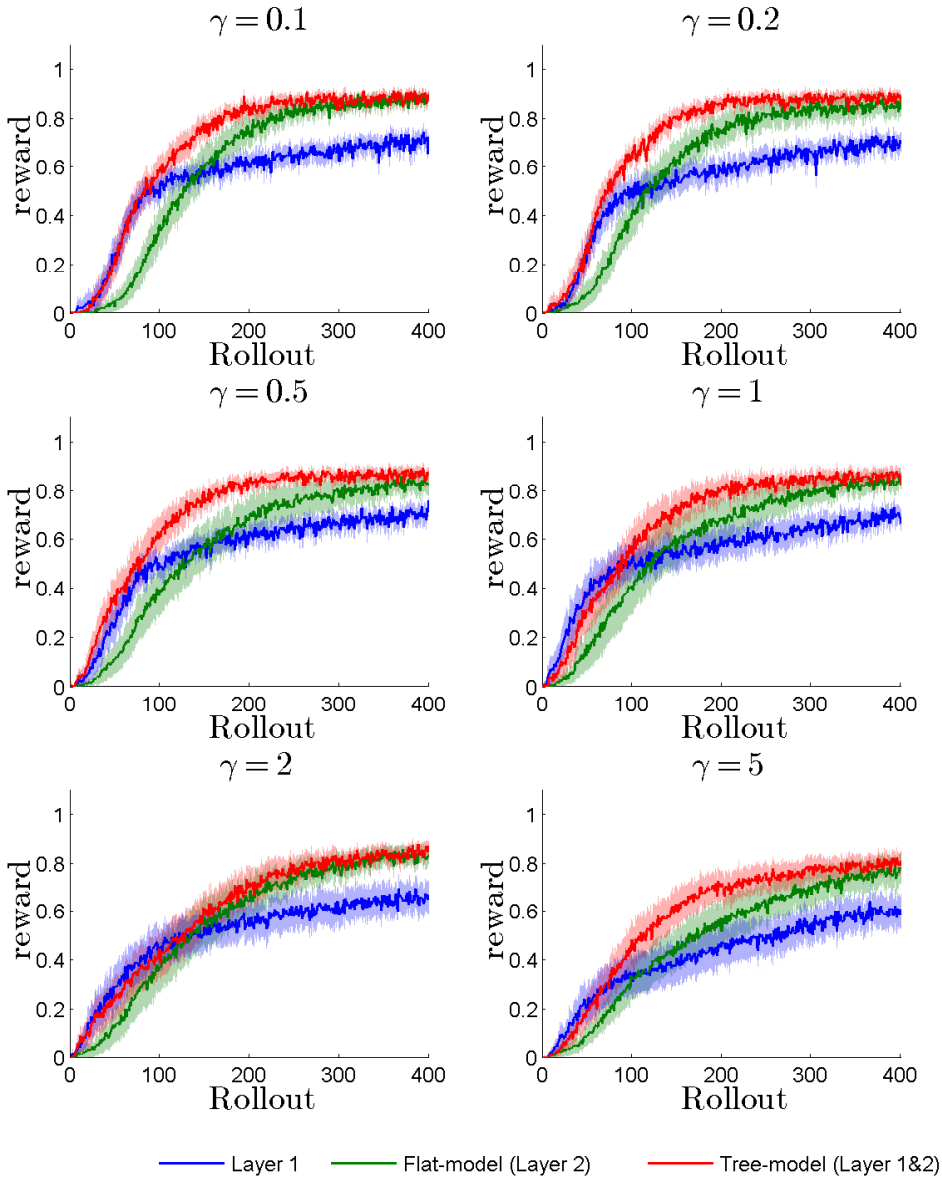


Figure C-7: The Figure shows the learning performance of a 2 Layer Tree-model while learning the multi-resolution movement without an offset (shown in Figure C-1) for different initial noise settings. Each graphs gives the results for a different initial noise settings. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

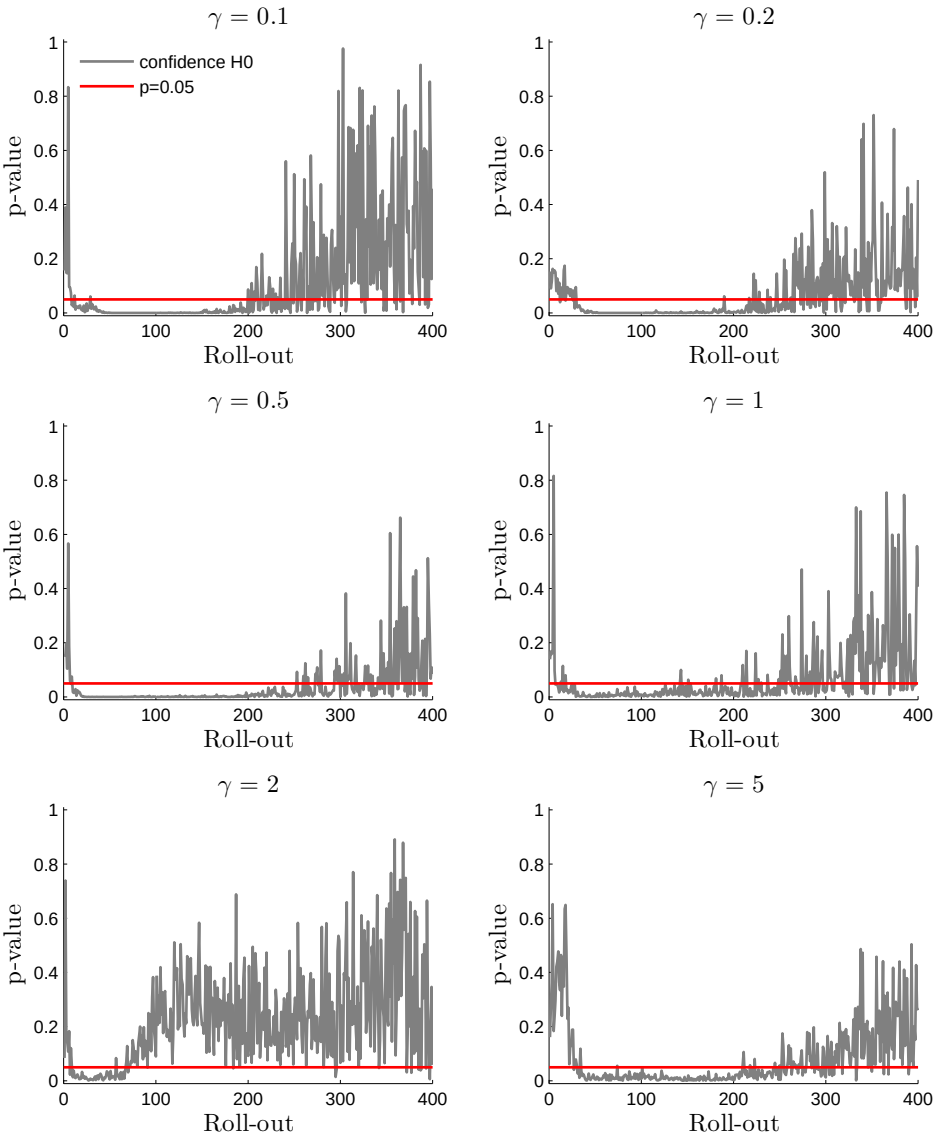


Figure C-8: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning the multi-resolution trajectory without an Offset (shown in Figure C-1) for different noise settings. The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flatmodel. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$.

Comparison: Multi Resolution Offset

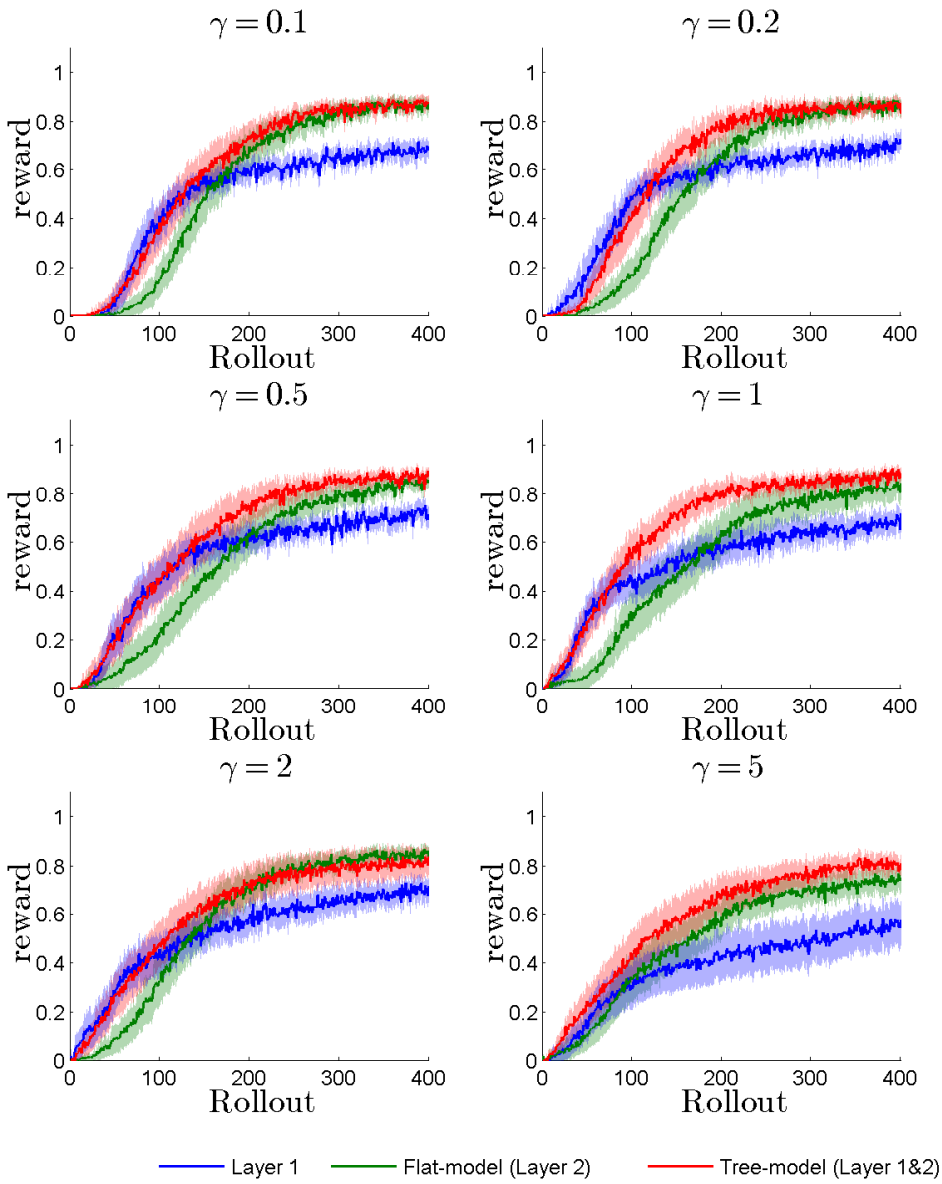


Figure C-9: The Figure shows the learning performance of a 2 Layer Tree-model while learning the multi-resolution movement with an offset (shown in Figure C-1) for different initial noise settings. Each graphs gives the results for a different initial noise settings. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

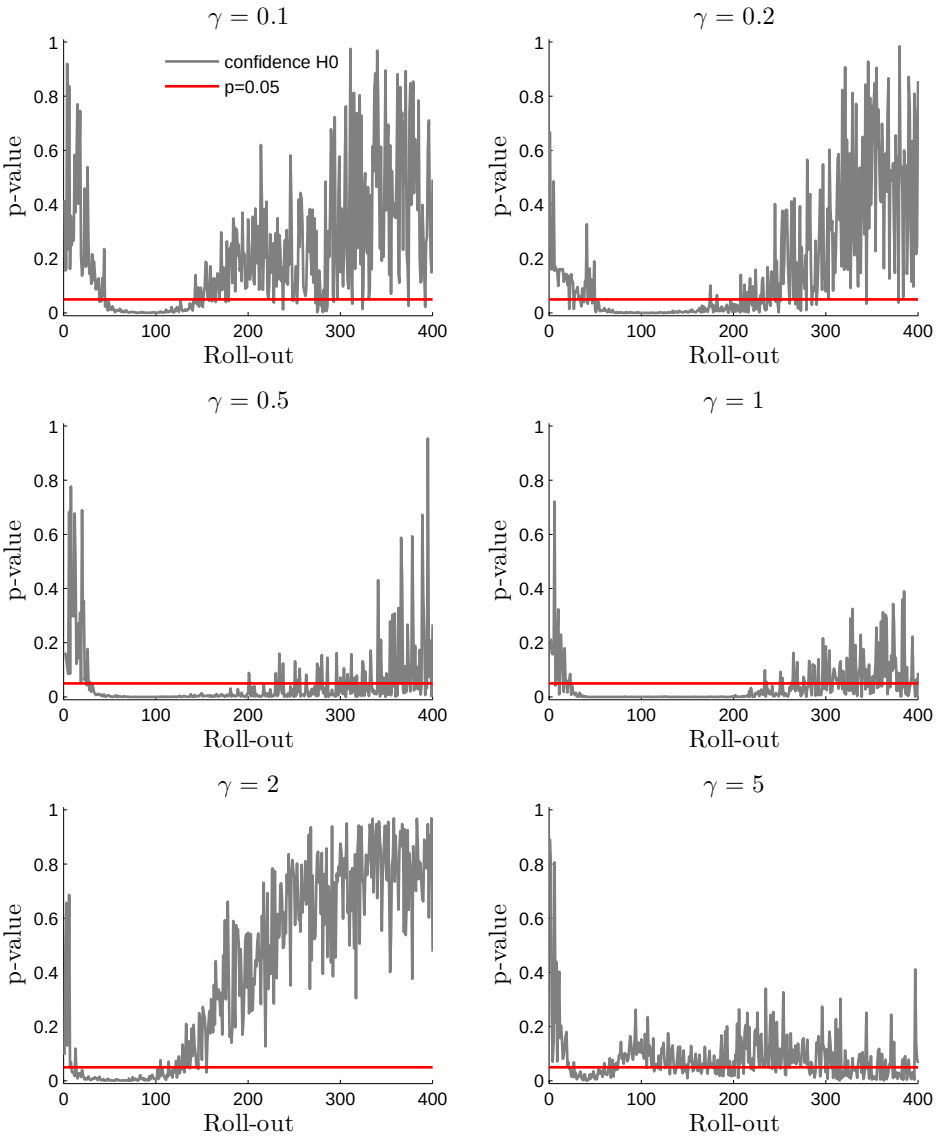
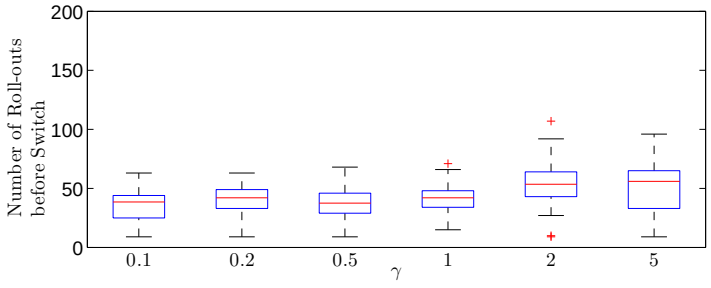
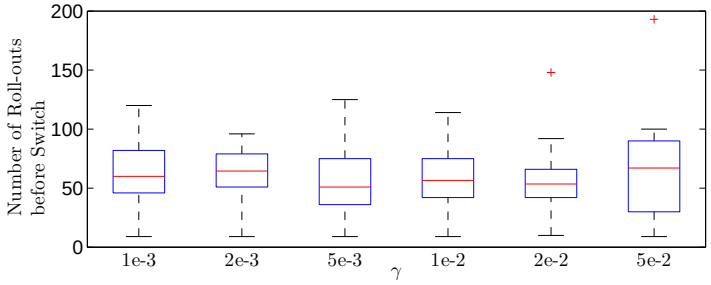


Figure C-10: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning the multi-resolution trajectory with an Offset (shown in Figure C-1) for different noise settings. The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flatmodel. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$.

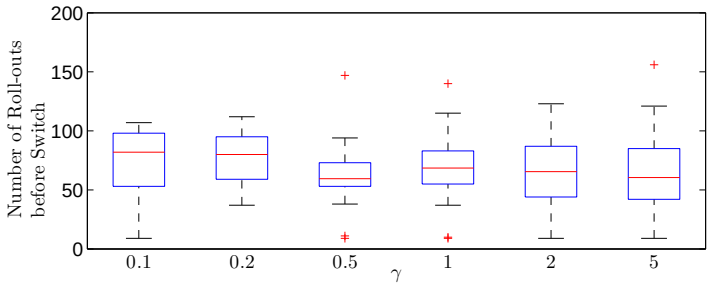
Switching Behavior



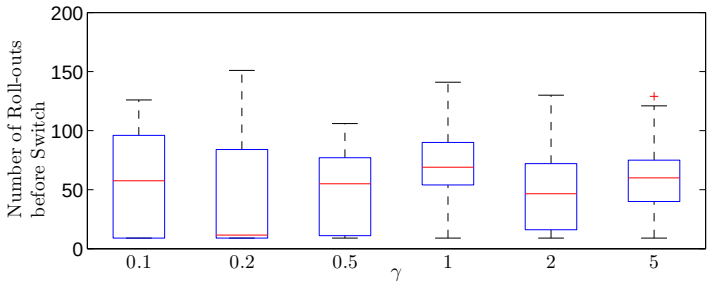
(a) Single resolution no Offset



(b) Single resolution Offset



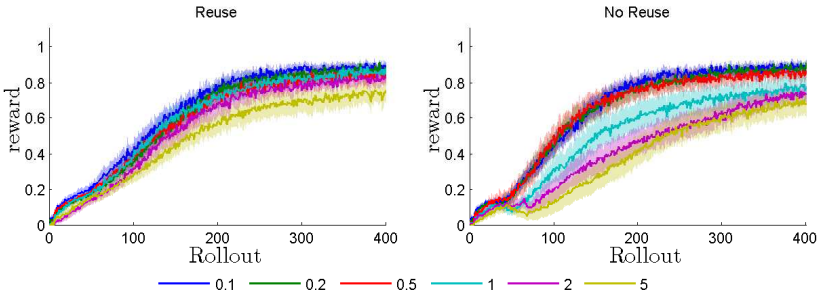
(c) Multi resolution no Offset



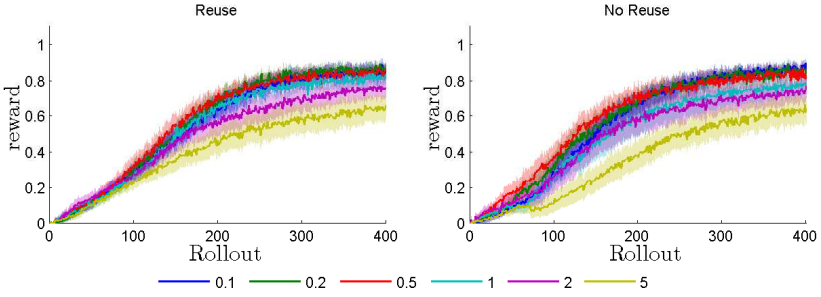
(d) Multi resolution Offset

Figure C-11: Figure gives a summary of the switching behavior of the Tree-model while learning the 4 different movements (a-d) (shown in Figure C-1) for different initial noise settings. The boxplots indicate the number of roll-outs used before switching from the first layer of the Tree-model to the second layer of the Tree-model. Each boxplot is based on 30 observations. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}\left([0.1, 1]\right)\gamma$.

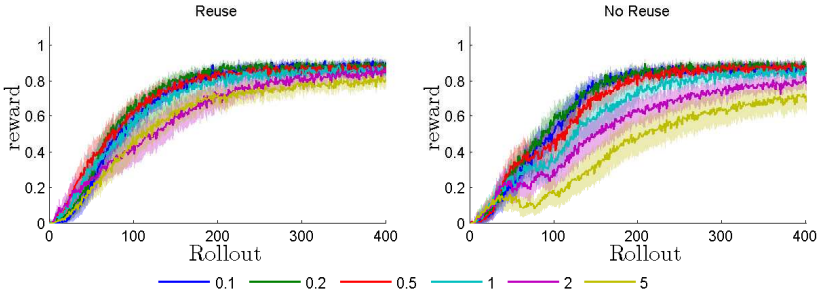
C-1-3 Reuse of rewards



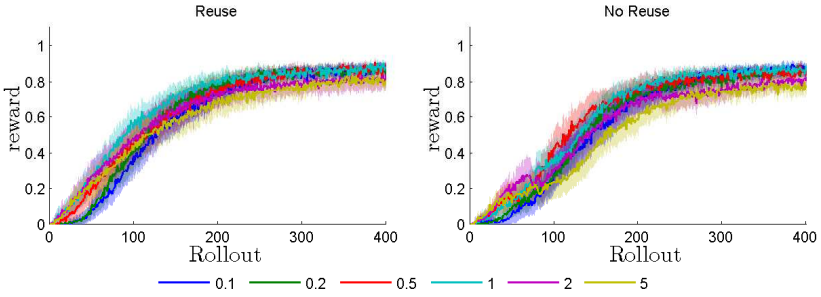
(a) Single resolution no Offset



(b) Single resolution Offset



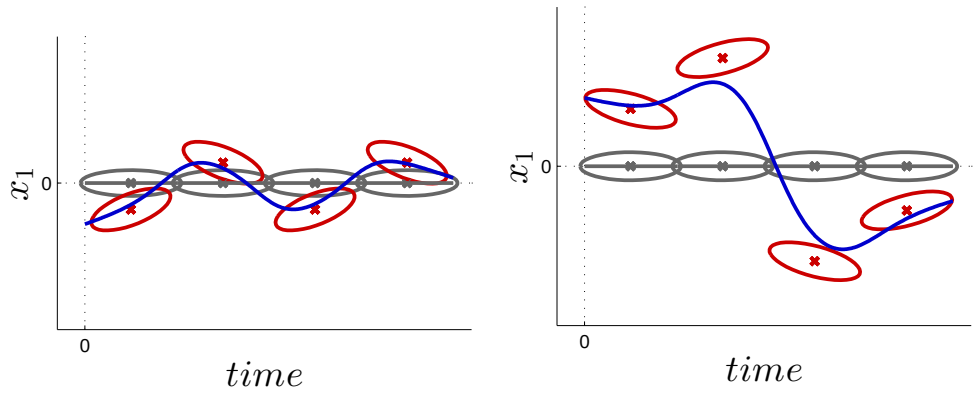
(c) Multi resolution no Offset



(d) Multi resolution Offset

Figure C-12: The figure shows the learning performance of a 2 layer Tree-model learning 4 different movements (a-d), shown in Figure C-1, when reusing experience and not reusing experience. When experience is reused, the exploration noise of the second layer is initialized based on the experience obtained during layer 1. When not reusing experience, the exploration noise of layer 2 is initialized similar to the initial exploration noise of layer 1. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

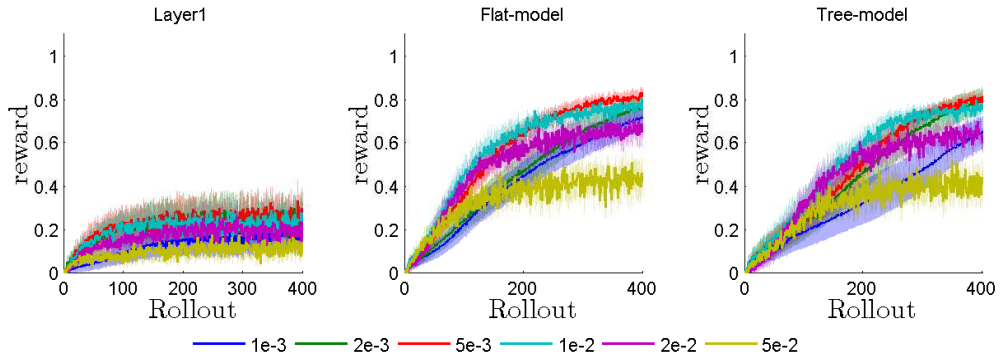
C-2 Experiment II



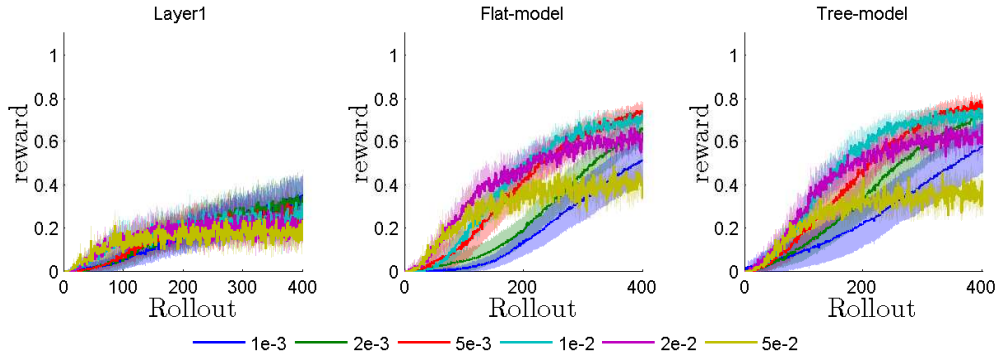
(a) Single-resolution **(b)** Multi-resolution

Figure C-13: The objective trajectories used in Experiment II. The goal trajectory (in blue) was generated by a pre-defined GMM. Learning starts from an initial state displayed in gray.

C-2-1 Results: Minimum Noise Experiments

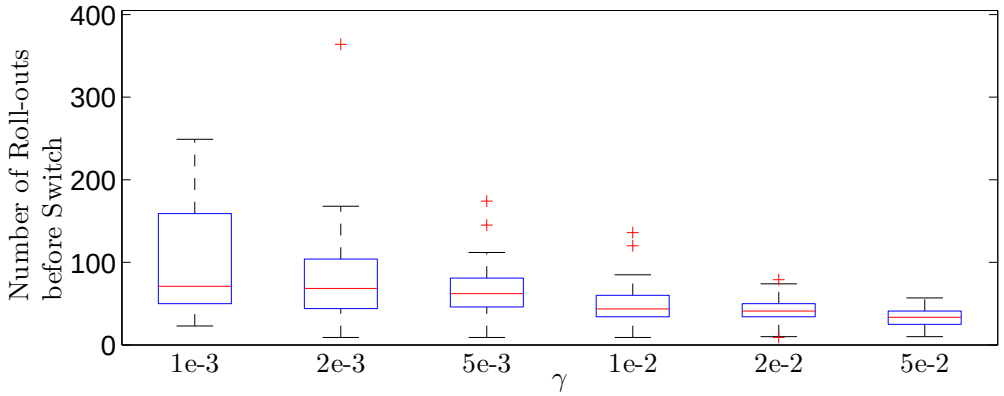


(a) Single Resolution

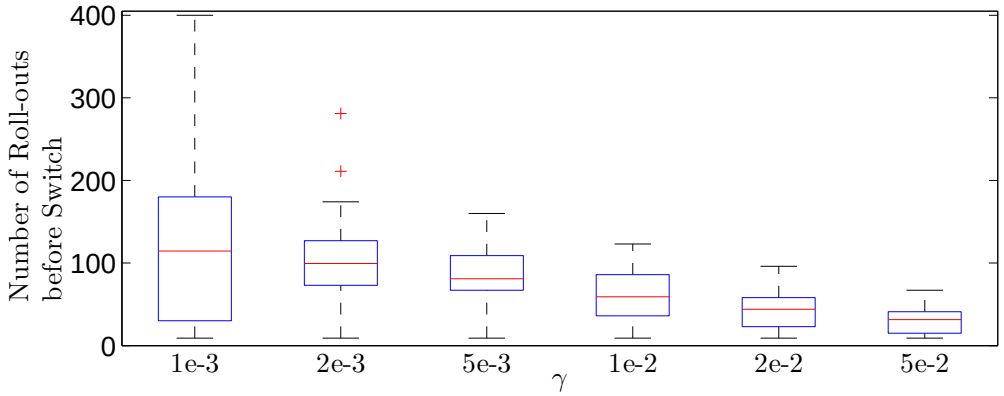


(b) Multi Resolution

Figure C-14: Figure shows learning performance of 2 different movements (a-b) using a 2 layer Tree-model. Each movement (a-b) is learned by exploring only Layer 1, exploring only Layer 2 (Flat-model) or exploring both Layer 1 and 2 (Tree-model). The different lines in the graphs indicate different initial noise settings. The numbers in the legend represent the initial noise fraction ϵ_{Noise} of the Initial noise matrix $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])$. The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.



(a) Single Resolution

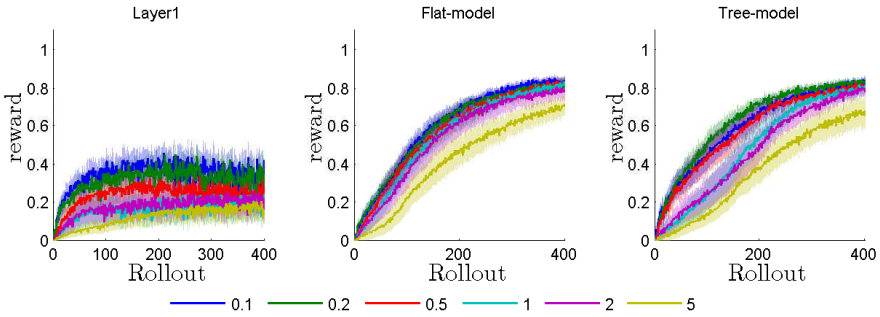


(b) Multi Resolution

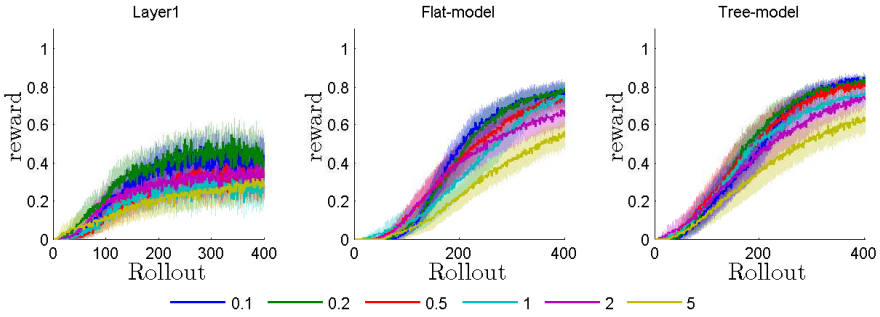
Figure C-15: Figure gives a summary of the switching behavior of the Tree-model while learning the 2 different movements (a-d) (shown in Figure C-13) for different initial noise settings. The boxplots indicate the number of roll-outs used before switching from the first layer of the Tree-model to the second layer of the Tree-model. Each boxplot is based on 30 observations. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])\,\gamma$.

C-2-2 Results: Initial Noise Experiments

Results per layer



(a) Single-resolution



(b) Multi-resolution

Figure C-16: Learning curves resulting the learning of 2 different trajectories (a-b) for different initial noise settings. The results are splitted per layer. The numbers in the legend represent the γ to determine the noise matrix $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])\,\gamma$. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

Comparison: Multi-resolution

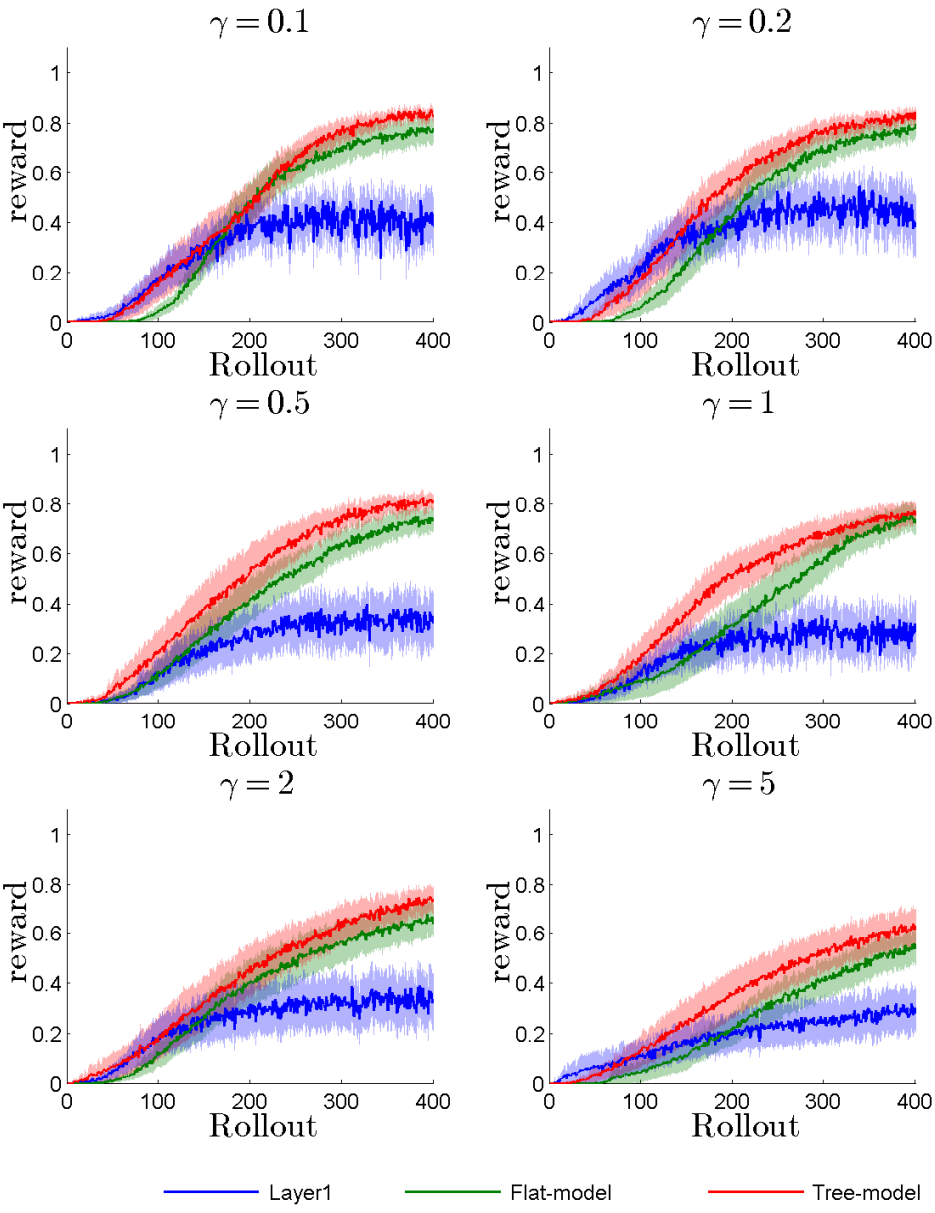


Figure C-17: The Figure shows the learning performance of a 2 Layer Tree-model while learning the multi-resolution movement without an offset (shown in Figure C-13) for different initial noise settings. Each graphs gives the results for a different initial noise settings. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1, .1])\,\gamma$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

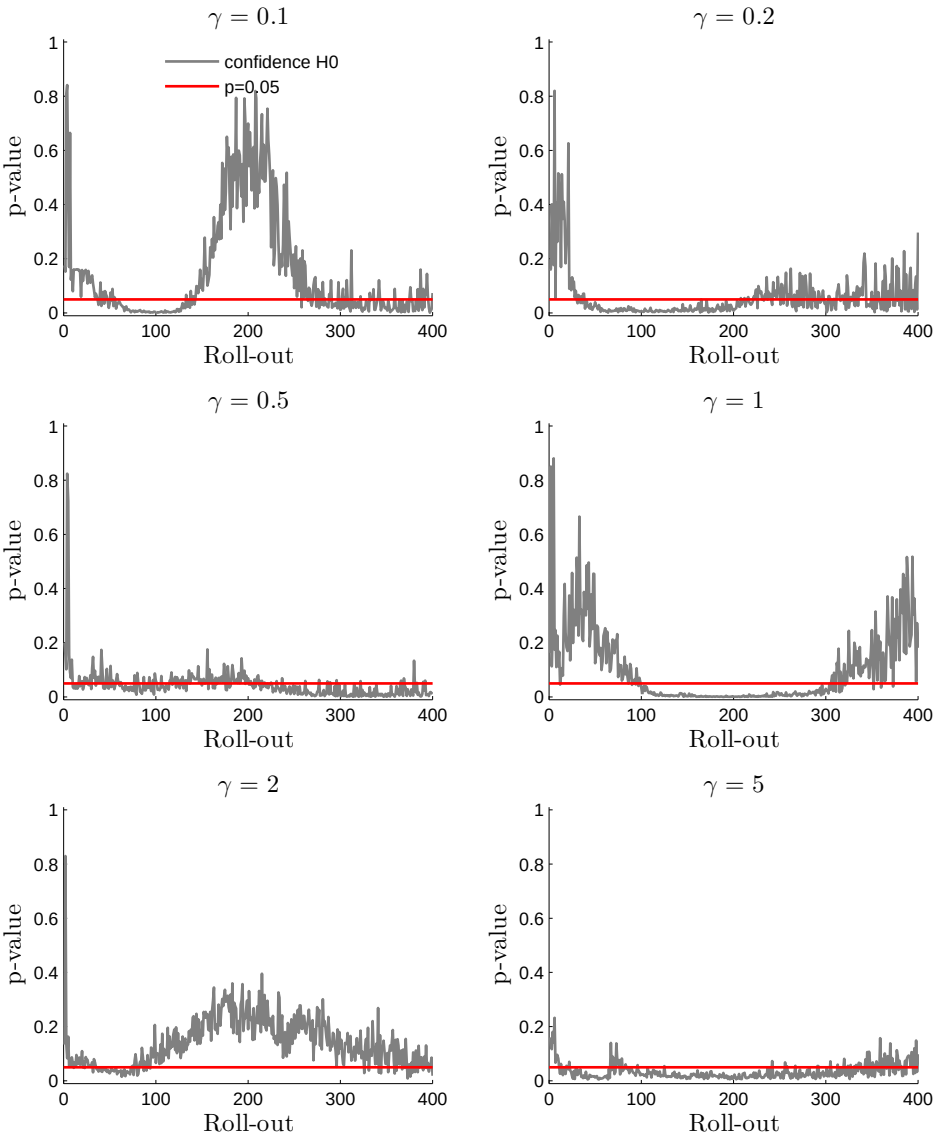


Figure C-18: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning the multi-resolution trajectory (shown in Figure C-1) for different noise settings. The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flatmodel. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}\left([0.1, 1, 0.1]\right)\gamma$.

Comparison: Single-resolution

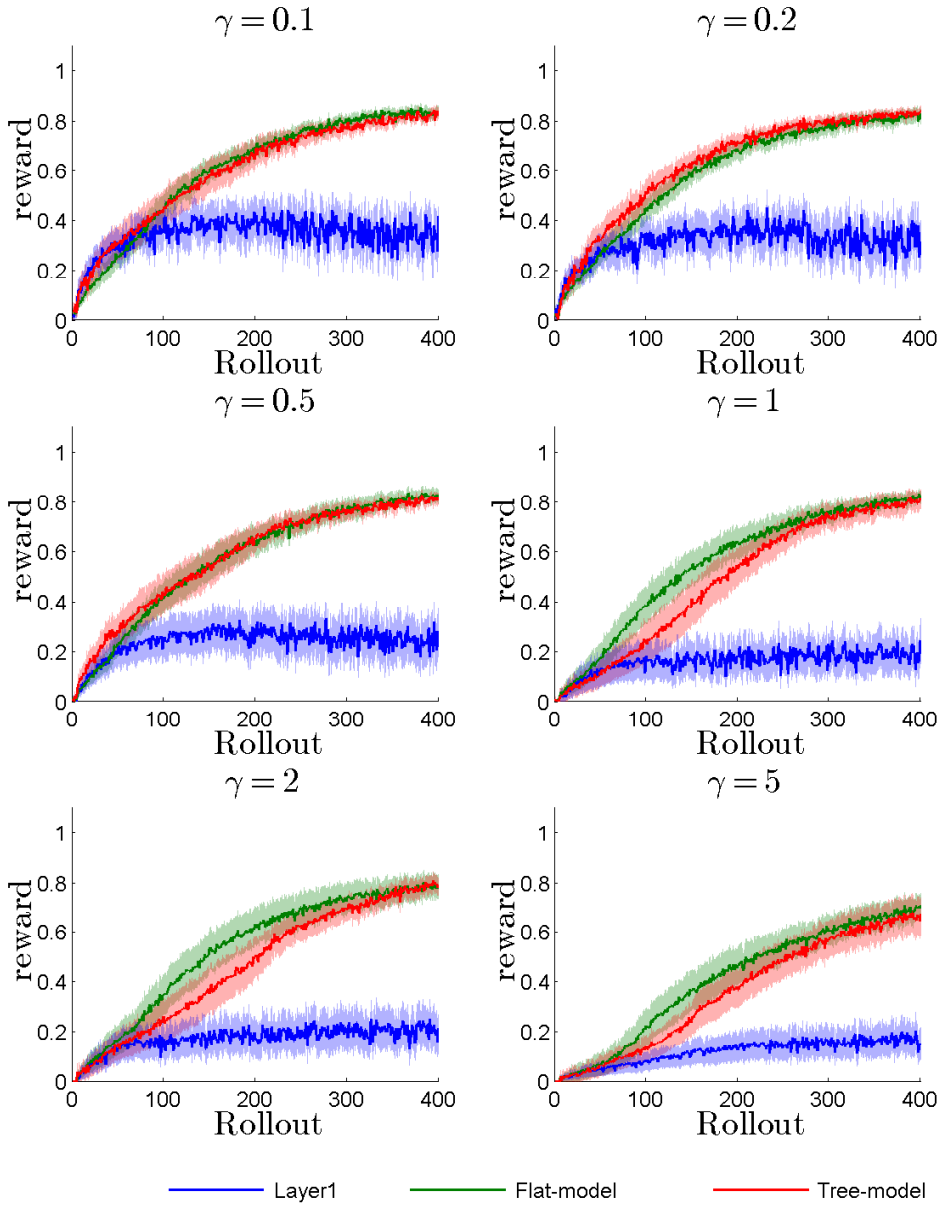


Figure C-19: The Figure shows the learning performance of a 2 Layer Tree-model while learning the single-resolution trajectory (shown in Figure C-13) for different initial noise settings. Each graphs gives the results for a different initial noise settings. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1, .1])\,\gamma$. Each graph shows the learning performance for only exploring Layer 1 (blue), exploring only Layer 2 (green, Flat-model) or exploring both Layer 1 and 2 (red, Tree-model). The results are based on 30 repetitions of each situation. The colored areas indicate the 95% confidence interval corresponding to the lines with similar color.

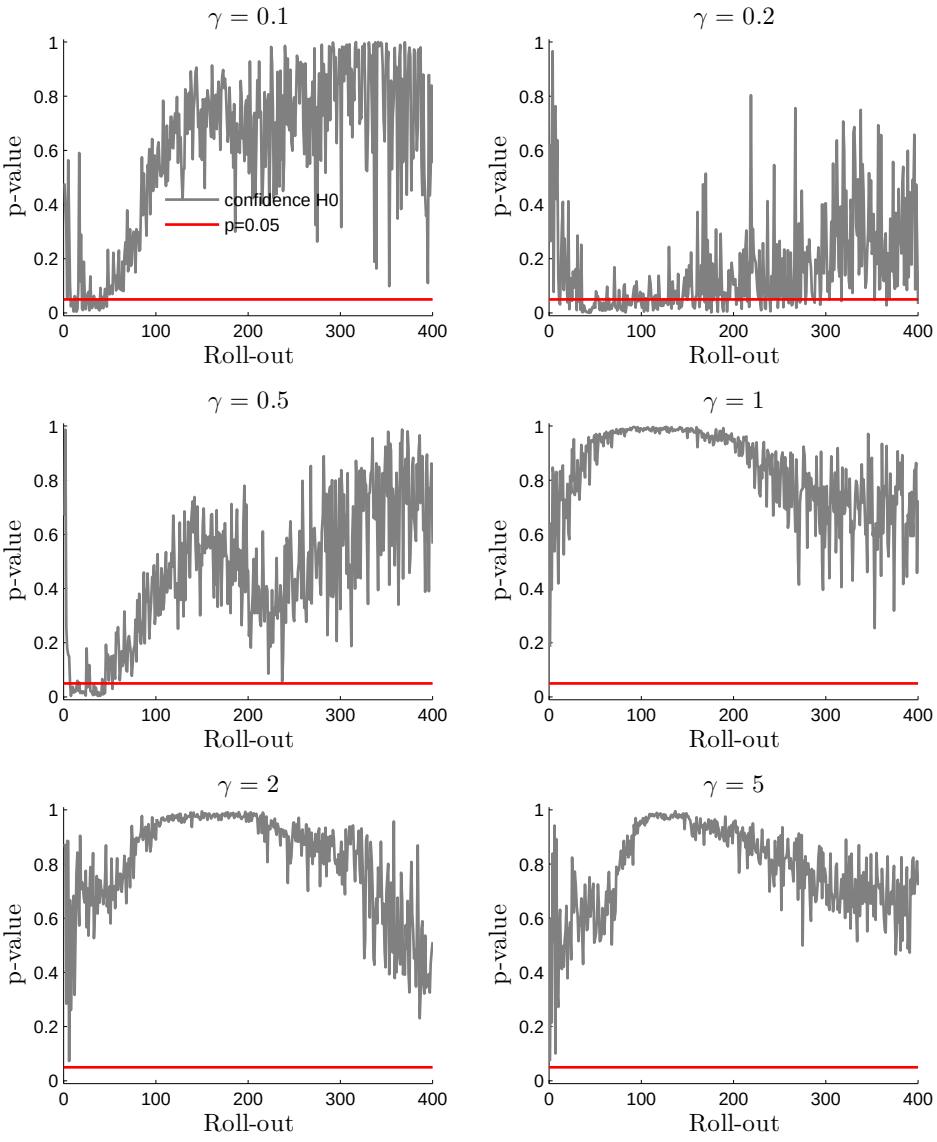
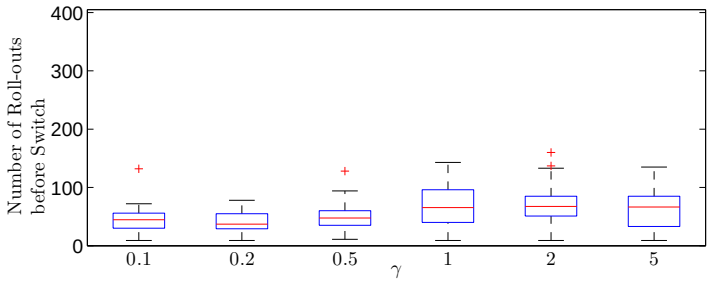
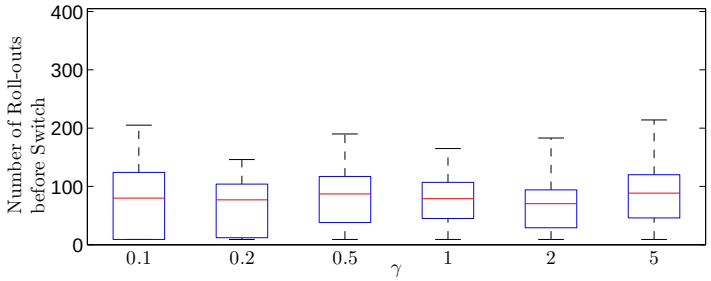


Figure C-20: Figure shows the p-value of a two-sample t-test that compares the learning performance of the Flat-model and the Tree-model while learning the single-resolution trajectory (shown in Figure C-1) for different noise settings. The null hypothesis H_0 indicates that the Flat-model and the Tree-model perform equally well. The alternative hypothesis, H_1, indicates that the Tree-model outperforms the Flatmodel. The p-value indicates the confidence in H_0 in favor of H_1. The results are based on 30 repetitions of each situation. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1, 0.1])\,\gamma$.

Switching Behavior



(a) Single-resolution



(b) Multi-resolution

Figure C-21: Figure gives a summary of the switching behavior of the Tree-model while learning the 2 different movements (a-b) (shown in Figure C-13) for different initial noise settings. The boxplots indicate the number of roll-outs used before switching from the first layer of the Tree-model to the second layer of the Tree-model. Each boxplot is based on 30 observations. The γ in the title of each graph indicates the magnitude of the noise, i.e. $\Sigma_{Noise} = \text{diag}([0.1, 1])\,\gamma$.

Bibliography

[1] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *RSJ Advanced Robotics, Special Issue on Imitative Robots*, vol. 21, no. 13, pp. 1521–1544, 2007.

[2] P. Kormushev, S. Calinon, and D. G. Caldwell, "Approaches for learning human-like motor skills which require variable stiffness during execution," in *IEEE Intl Conf. on Humanoid Robots (Humanoids), Workshop on Humanoid Robots Learning from Human Interaction*, 2010.

[3] J. Kober and J. Peters, "Imitation and reinforcement learning: Practical algorithms for motor primitives in robotics," *IEEE Robotics and Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[4] J. Peters, K. MÃijlling, K. Kober, D. Nguyen-Tuong, and O. KrÃűmer, "Towards motor skill learning for robotics," in *Robotics Research* (C. Pradalier, R. Siegwart, and G. Hirzinger, eds.), Springer, 2011.

[5] F. Stulp and O. Sigaud, "Policy improvement methods: Between black-box optimization and episodic reinforcement learning," 2012. (under review).

[6] S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning," *Robotics and Autonomous Systems*, 2012.

[7] K. Graichen, M. Treuer, and M. Zeitz, "Swing-up of the double pendulum on a cart by feedforward and feedback control with experimental validation.," *Automatica*, vol. 43, no. 1, pp. 63–71, 2007.

[8] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469 – 483, 2009.

[9] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration." Handbook of Robotics, . chapter 59, 2008, 2008.

[10] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Reinforcement Learning* (M. Wiering and M. Otterlo, eds.), vol. 12 of *Adaptation, Learning, and Optimization*, pp. 579–610, Springer Berlin Heidelberg, 2012.

[11] P. Kormushev, S. Calinon, D. G. Caldwell, and B. Ugurlu, "Challenges for the policy representation when applying reinforcement learning in robotics," in *IJCNN*, pp. 1–8, 2012.

[12] R. Sutton, H. Maei, D. Precup, S. Bhatnagar, D. Silver, C. SzepesvÃąri, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *In Proceedings of the 26th International Conference on Machine Learning*, 2009.

[13] S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation," *Phil. Trans. R. Soc. Lond. B*, vol. 358, pp. 537–547, 2003.

[14] F. Stulp, J. Buchli, E. Theodorou, and S. Schaal, "Reinforcement learning of full-body humanoid motor skills," in *2010 IEEE-RAS International Conference on Humanoid Robots Nashville, TN, USA, December 6-8, 2010*, IEEE, 2010.

[15] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, no. 1-2, pp. 171–203, 2011.

[16] P. Kormushev, B. Ugurlu, S. Calinon, N. G. Tsagarakis, and D. G. Caldwell, "Bipedal walking energy minimization by reinforcement learning with evolving policy parameterization," in *IROS*, pp. 318–324, 2011.

[17] A. Ijspeert, J. Nakanishi, and S. Schaal, "Trajectory formation for imitation with nonlinear dynamical systems," 2001.

[18] A. Ude, C. Atkeson, and M. Riley, "Programming full-body movements for humanoid robots by observation," *Robotics and Autonomous Systems*, vol. 47, no. 23, pp. 93 – 108, 2004.

[19] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[20] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 37, no. 2, pp. 286–298, 2007.

[21] S. Calinon, I. Sardellitti, and D. G. Caldwell, "Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 249–254, 2010.

[22] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation: An approach based on hidden Markov model and Gaussian mixture regression," *IEEE Robotics and Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010.

[23] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 1-2, pp. 41–77, 2003.

[24] A. Ijspeert, N. J., and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *in Advances in Neural Information Processing Systems*, pp. 1523–1530, MIT Press, 2003.

[25] C. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[26] S. H. Lee, I. H. Suh, S. Calinon, and R. Johansson, "Learning basis skills by autonomous segmentation of humanoid motion trajectories," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, (Osaka, Japan), pp. 112–119, 2012.

[27] S. Calinon, Z. Li, T. Alizadeh, N. G. Tsagarakis, and D. G. Caldwell, "Statistical dynamical systems for skills acquisition in humanoids," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, (Osaka, Japan), 2012.

[28] P. Kormushev, S. Calinon, and D. G. Caldwell, "Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input," *Advanced Robotics*, vol. 25, no. 5, pp. 581–603, 2011.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[30] I. Grondman, L. Buşoniu, and R. Lopes, G.A.D. Babuška, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on systems, man, and cybernetics*, 2012.

[31] R. Rückstieß, M. Felder, and J. Schmidhuber, "State-Dependent Exploration for policy gradient methods," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2008, Part II, LNAI 5212*, pp. 234–249, 2008.

[32] J. Kober, "Reinforcement learning for motor primitives," diploma, Universty of Stuttgart, 8 2008.

[33] N. Hansen, "The cma evolution strategy: A comparing review," in *Towards a New Evolutionary Computation* (J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, eds.), vol. 192 of *Studies in Fuzziness and Soft Computing*, pp. 75–102, Springer Berlin / Heidelberg, 2006.

[34] L. Busoniu, D. Ernst, J. De Schutter, and R. Babuska, "Cross-entropy optimization of control policies with adaptive basis functions," *IEEE Transactions on Systems Man and Cybernetics*, vol. 41, pp. 196–209, 2011.

[35] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, pp. 3137–3181, Dec. 2010.

[36] J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber, "Ontogenetic and phylogenetic reinforcement learning," *Künstliche Intelligenz*, pp. 30–33, March 2009.

[37] K. Yamane, Y. Yamaguchi, and Y. Nakamura, "Human motion database with a binary tree and node transition graphs," in *Proceedings of Robotics: Science and Systems*, (Seattle, USA), June 2009.

[38] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. pp. 1–38, 1977.

[39] C. C. S. J. C. K. Zhang, C., "Em algorithms for gaussian mixtures with split-and-merge operation," *Pattern Recognition*, vol. 9, pp. 1973–1983, 2003.

[40] H. Trautmann, T. Wagner, B. Naujoks, M. Preuss, and J. Mehnen, "Statistical methods for convergence detection of multi-objective evolutionary algorithms," *Evol. Comput.*, vol. 17, pp. 493–509, Dec. 2009.

[41] F. Provost, D. Jensen, and T. Oates, "Efficient progressive sampling," in *In Proc. of the fifth ACM SIGKDD intl. conf. on Knowledge discovery and data mining*, KDD '99, (New York, NY, USA), pp. 23–32, ACM, 1999.

[42] J. C. Pinheiro and D. M. Bates, "Unconstrained parameterizations for variance-covariance matrices," *Statistics and Computing*, vol. 6, pp. 289–296, 1996.

[43] S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning," *Robotics and Autonomous Systems*, vol. 61, pp. 369âĂŞ–379, April 2013.

[44] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an em approach," in *Advances in Neural Information Processing Systems 6*, pp. 120–127, Morgan Kaufmann, 1994.

[45] F. Wong, "Efficient estimation of covariance selection models," *Biometrika*, vol. 90, no. 4, pp. 809–830, 2003.

[46] M. Pourahmadi, "Joint mean-covariance models with applications to longitudinal data: Unconstrained parameterisation," *Biometrika*, vol. 86, pp. 677–690, 1999.

[47] M. Pourahmadi, "Maximum likelihood estiamtion of generalised linear models for multivariate normal covariance matrix," *Biometrika*, vol. 87, pp. 425–435, 2000.

[48] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with EM-based reinforcement learning," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[49] R. van der Linde and A. Schwab, "Multibody dynamics b," 1998.

Glossary

List of Acronyms

TU Delft	Delft University of Technology
IIT	Instituto Italiano di Tecnologia
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
EM	Expectation Maximization
pdf	Probability Density Function
LfE	Learning from Exploration
PbD	Programming by Demonstration
RL	Reinforcement Learning
BBO	Black Box Optimization
SVD	Singular Value Decomposition
DMP	Dynamic Movement Primitives
DS-GMR	Dynamical System Gaussian Mixture Regression
CEM	Cross Entropy Method
CMA-ES	Covariance Matrix Adaptation, Evolutionary Strategy
PoWER	Policy improvement by Weighting Expected Return
EOM	Equations of Motion

List of Symbols

β	List of parameters describing a lower triangular matrix obtained from Cholesky decomposition of a covariance matrix.
ϵ_{conv}	Convergence threshold
ϵ_{Noise}	Minimum exploration noise
λ_i	Eigen value
$\boldsymbol{\Lambda}$	A diagonal matrix of which the diagonal entries are eigenvalues.
$\boldsymbol{\phi}$	Parameters required to parameterize the Covariance matrix, but not required during exploration (see section 3-2-2)
$\boldsymbol{\Sigma}^{Noise}$	The covariance of the Exploration noise
$\boldsymbol{\mu}_k$	Mean of the k-th Gaussian probability density function.
$\boldsymbol{\phi}$	An vector of parameters that are not explored during Learning from Exploration (LfE), but are required to reconstruct the GMM during deparameterization.
$\pi(\boldsymbol{s})$	The policy of a robot, it defines a mapping between states and actions.
π_k	The prior probability of Gaussian k
Ψ^g	Global frame of reference
$\Psi_i^{l,p}$	Frame of reference of a Tree model; i indicates the number of the frame of reference in layer l. p indicates the parent frame of reference which - by convention - is always is defined in layer $l-1$.
Ψ^l	Local frame of reference
$\boldsymbol{\Sigma}_k$	Covariance of the k-th Gaussian probability density function.
$\boldsymbol{\tau}_k$	Information resulting from a roll-out of the policy, e.g. State transitions, rewards etc.)
$\boldsymbol{\theta}$	A vector of policy parameters
$\boldsymbol{\theta}$	A vector of policy parameters
$\boldsymbol{\theta}$	Vector of policy parameters
$\boldsymbol{\theta}_{\Sigma^{\mathcal{IO}}}$	Parameters considered for exploration related to the covariance matrix
$\boldsymbol{\xi}^{\mathcal{I}} = \boldsymbol{s}$	Vector of output variables
$\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{a}$	Vector of output variables
β	Set of parameter required to fully define a covariance matrix.
β_i	Parameter required to describe part of the covariance matrix.
$\boldsymbol{a}_t$	An action performed by the robot at time t
$\boldsymbol{H}_j^i$	Affine transformation matrix from frame of reference j to frame of reference i
$\boldsymbol{L}$	Lower triangular matrix
$\boldsymbol{o}_j^i$	Represents the origin of frame of reference Ψ^j expressed in frame of reference Ψ^i
$\boldsymbol{R}_j^i$	Rotation matrix represents the rotation of frame of reference Ψ^j with respect to frame of reference Ψ^i
$\boldsymbol{s}_t$	The state of a robot and/or the environment at time t.
$\boldsymbol{V}$	Matrix of which the columns are eigenvectors
$\boldsymbol{x}_n$	A multidimensional data point

$\mathrm{r}(\boldsymbol{s}, \boldsymbol{a})$	State and action dependent reward function
$\Phi(t)$	Basis function
$\mathcal{P}(\cdot)$	Gaussian probability density function.
$\mathrm{TM}\left(\boldsymbol{\theta}^1_{init}, \ldots, \boldsymbol{\theta}^L_{init}\right)$	Variable representing the Tree-model with a parameter vector $\boldsymbol{\theta}^l$ for each layer
$d^{\mathcal{I}}$ and $d^{\mathcal{O}}$	Dimensions of the input and output parameters
$f(t)$	Time dependent forcing function used in Dynamic Movement Primitives (DMP) (See Section 2-1)
f_i	Force component used in DMP (See Section 2-1)
I	Number of roll-outs considered for one update
$k^{\mathcal{P}}$	Virtual stiffness
k^v	Virtual damping coefficient
l	Indicates the Layer of a Tree-model
M	The number of roll-outs considered during convergence detection
N	Number of samples
S_{init}	Number of roll-outs before first update
D	Diagonal matrix
d	number of dimensions.
Λ	Diagonal matrix containing the eigenvalues λ
L	Lower triangular matrix
μ_k	Mean of the k-th Gaussian probability density function.
$\mathcal{P}(x)$	Gaussian probability density function.
ϕ_i	Parameter describing part of the correlation matrix $\boldsymbol{\Sigma}^{\mathcal{IO}}$.
π_k	the prior probability of Gaussian probability density function k.
Σ_k	Covariance of the k-th Gaussian probability density function.
T	Lower Unitriangular matrix (Matrix with unit values on the diagonal)
V	Matrix of which the columns are eigenvectors