Exploiting model-based reinforcement learning for non-prehensile manipulation of deformable objects

handed in
MASTER'S THESIS

B.Sc. Berkol Görür

born on the 03.07.1992
living in:
Schleißheimer 14
80333 Munich
Tel.: 15777 - 2875453

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor:	Dr.-Ing. Matteo Saveriano
Start:	03.09.2018
Intermediate Report:	14.02.2018
Delivery:	20.05.2019

TECHNISCHE UNIVERSITÄT MÜNCHEN
Human-centered Assistive Robotics
UNIV.-PROF. DR.-ING. DONGHEUI LEE

May 27, 2019

M A S T E R ' S T H E S I S
for
Berkol Görür
Student ID 03681015, Degree EI

Exploiting model-based reinforcement learning for non-prehensile manipulation of deformable objects

Problem description:

Non-prehensile manipulation actions, like pushing or folding, aim at modifying the pose of an object without firmly grasping it. Non-prehensile manipulation is a hard task and, especially when the object to manipulate is deformable, it is far from being fully solved [1]. This is because the task at hand is hard to model and the complex model makes the defign of a control policy non trivial. Reinforcement learning (RL), or leaning by self-practice, is an appealing approach widely used in robotics to learn complex skills. In particular, model-based RL approaches [2, 3], combined with non-linear optimal control approaches like the Iterative Linear Quadratic Gaussian (ILQG) [4], seem well-suited to execute non-prehensile manipulation tasks.
In this Master thesis work, the student has to implement a learning based approach for non-prehensile manipulation of deformable objects. The approach starts with a simplified task model, e.g. considering a rigid body, and improves both the task model and the control policy with data from real executions. The effectiveness of the developed approach is demonstrated with experiments on a real robot.

Tasks:

- Literature research on model-based reinforcement learning and non-prehensile manipulation
- Combination of PI-REM [2] and ILQG [4]
- Application of model-based RL to non-prehensile manipulation of a deformable object
- Evaluation on a real robot

Bibliography:

[1] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile Dynamic Manipulation: A Survey", in *Robotics and Automation Letters*, 2018.
[2] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-Efficient Control Policy Search using Residual Dynamics Learning", in *International Conference on Intelligent Robots and Systems*, 2017.
[3] S. Levine and P. Abbeel, "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics", in *Neural Information Processing Systems*, 2014.
[4] Y. Tassa, N. Mansard, and E. Torodov, "Control-limited differential dynamic programming", in *International Conference on Robotics and Automation*, 2014.

Supervisor:	Dr.-Ing. Matteo Saveriano
Start:	03.09.2018
Intermediate Report:	14.02.2019
Delivery:	20.05.2019

(D. Lee)
Univ.-Professor

Abstract

Representation of uncertain system dynamics as Gaussian Process (GP) is an effective approach to alleviate the model-bias problem in reinforcement learning tasks. While Probabilistic Inference for Learning Control (PILCO) algorithm leverages the Gaussian Process regression by incorporating the uncertainty in decision making process by means of a parametrized policy function, it neglects the expert knowledge about the system dynamics. Policy Improvement with Residual Model Learning (PIREM) combines the expert knowledge about the approximate dynamics of real system with the probabilistic representational capacity of GP regression by modelling the residual dynamics between the real and approximate as a Gaussian Process.

The approaches mentioned above rely on the use of an external optimizer to update policy parameters based on the gradient information of the return function. The goal of this project is to replace this policy improvement step with iterative Linear Quadratic Regulator (iLQR) method which computes a locally linear policy through a series of recursive backward and forward-pass steps. Experiments show that combination of iLQR with PIREM significantly reduces the computation complexity of policy parameter stage and allows faster convergence to the optimal policy with less rollouts.

The algorithms are tested on balancing tasks involving the control of an inverted pendulum and manipulation of a deformable ball on the plate. While all algorithms can find an optimal policy in simulation environments, only closed-loop PIREM-iLQR is capable of computing an optimal policy for the real ball-and-plate system within a reasonable amount of rollouts. Inconsistency in the dynamics of manipulated ball, delay in control signals, inaccuracies arising from linear extrapolation used under certain conditions, absence of sparse GP regression and ineffectiveness of feedback terms received from the approximate model are the possible causes for partial failure of the real experiments.

Contents

Chapter 1

Introduction

In reinforcement learning (RL) tasks, the agent aims to improve the current strategy to maximize the received rewards upon a series of interaction with the environment. While many *model-free* RL algorithms offer effective approaches to find the optimal policy in the absence of knowledge on system dynamics, they require large amount of data and time-consuming trials. Especially for robotic applications, this *sample inefficiency* adversely affects the feasibility of experiments due to factors such as wears and tears on the hardware, unsafe exploratory behaviour and the need for constant human intervention during trials [KABP13]. While the issue of sample inefficiency can be alleviated through expert demonstrations and more accurate simulations, considering system dynamics during planning may significantly reduce the required number of interactions with the physical environment. *Model-based* methods exploit the available data by learning a generalized model of the system and take this model into account while developing a new policy [SB98].
The major drawback of model-based algoritms is so-called *model-bias* problem. In case the model is not accurate enough, algorithms fails to converge to the optimal policy [AS97]. Probabilistic Inference for Learning Control (PILCO) algorithm [DFR15] proposes a method to mitigate this problem by representing the system dynamics as a distribution over continous functions by means of Gaussian Process (GP). Hence the system uncertainity is encoded as Gaussian at the regions with sparse data. Natural adaptation of model uncertainity at the long-term prediction and policy update stages provides robustness againt model errors. Based on the gradient information of the total cost, policy parameters are updated by means of external optimizers.
While PILCO does not involve any external expert knowledge, Policy Improvement with REsidual Model learning (PIREM) algorithm enables us to explicitly incorporate a rough model of the system into transition dynamics. Rather than representing the whole system as a Gaussian process, PIREM models the residual difference between the actual model and partially known model called *approximate* model as GP. The initial policy obtained from the approximate model may provide a plausible starting trajectory and hence accelerate the convergence towards the optimal pol-

icy. Learning of the residual dynamics may considerably improve the performance in scenarios where unmodelled factors such as sensor noise, friction or dynamics of the manipulated object play a major role in the overall behaviour of the system.
The purpose of this project is to speed up the policy improvement process by incorporating iterative Linear Quadratic Regulator (iLQR) method into PILCO and PIREM algorithms which normally use optimization techniques like CG or L-BFGS to update policy parameters. Iterative Linear Quadratic Regulator yields locally optimal time-varying feedforward and feedback terms [TMT14]. These terms are computed via recursive application of backward and forward pass through local approximation of cost and dynamics functions without any parametrization.
Nonprehensile dynamic manipulation is regarded as one of the most complicated manipulation tasks in the realm of robotics. Due to the lack of a general theoretical framework, the field of nonprehensile dynamic manipulation has been a major area of research in the last decades. Although considerable improvements have been achieved with the advancements in sensing and actuation, non-smooth dynamics of the system prevent engineers from developing a unified method without splitting the main task into multiple subtasks.
Dynamic manipulation can be described as changing the system state under the influence of forces due to accelerations besides the kinematic, static and quasi -static forces, such as balancing a rolling object on a plate [RLS18]. This leads to the definition of further manipulation categories: Kinematic manipulation encompasses tasks which can be analysed solely by the kinematics of the effector. In other words, the state of manipulated object only depends on the motion of the robotic hand. Pick-and-place tasks are typical example of kinematic manipulation. On the other hand, static manipulation is subject to both kinematics and static forces, which can be exemplified by the frictional force between a gripper and the object providing a firm interaction between both sides. The third class of manipulation that constitutes a dynamic manipulation is quasi-static manipulation, where kinetic frictional force in sliding contacts plays a big role. Pushing an object on a table is an instance of quasi-statics manipulation.
Objects of dexterous manipulation are subjects to two kinds of constraints: *form closure* and *force closure* [PT08]. A grasping operation exhibits force closure if it can withstand any external disturbance. If infinitesimal motion of the manipulated object is prevented, the manipulation is also subject to form closure. A manipulation task is *nonprehensile* if the object is subject to unilateral constraints which ensure form or force closure in certain directions [ML93]. Pushing an object with the fingertip is a common case of nonprehensile dynamic manipulation. The hand is able to resist any external force exerted towards it in the axis of contact, however it cannot counteract any disturbance causing the object to deviate to both sides. Another example can be holding an object on the palm of a hand. While the hand can resist any force perpendicular to the palm surface, the object can easily be lifted up. Folding clothes, cooking in a pan, carrying objects on a tray and many surgery operations are further examples of nonprehensile manipulation.

Figure 1.1: Carrying objects on a tray is an example of nonprehensile manipulation. While the system can resist downward forces perpendicular to the tray, objects can be easily lifted without any resistance. Image retrieved from: http://www.montessoricircle.com/montessori-and-movement.

While some tasks are inherently nonprehensile such as unscrewing a bottle cap, many tasks can be achieved in both prehensile and nonprehensile manner. A ball can be displaced both by pushing and pick-and-place action. Besides, some tasks can also be achieved by a sequence of prehensile and nonprehensile manipulations. Throwing a ball with one hand and catching it with the other one during juggling is a nonprehensile action (throwing) followed by a prehensile one (grasping the ball). Nonprehensile manipulations offer ease of control owing to increased workspace and degrees of freedom. Since the manipulated object does not have to be grasped within a certain domain of configuration space, any contact surface can be used to interact with it. This enables the manipulator to achieve the task with a larger degree of freedom with enhanced dexterity. For instance, a bottle can be unscrewed in many alternative ways. Additionally, one certain task can be accomplished with different alternatives of nonprehensile manipulation primitives.
Advantages of nonprehensile manipulation comes at the expense of increased complexity in planning and control due to complex system dynamics. Change in the contact status during nonprehensile manipulation causes non-smooth dynamics, which is the main obstacle to find a unified framework valid for all tasks. To deal with non-smooth system behaviour, the whole manipulation task is usually split into nonprehensile *manipulation primitives*. Each primitive is subsequently performed with ad-hoc controllers. Manipulation primitives covered in the literature include throwing, dynamics catching, batting, juggling, dribbling, pushing, sliding, rolling etc. A high-level controller is employed to switch between these primitives during the manipulation process [ML93]. Carrying objects on a tray is an example of nonprehensile manipulation. While the system can resist downward forces perpendicular to the tray, objects can be easily lifted without any resistance.

1.1 Related Works

Fu and Levine [FLA15] present a model-based reinforcement algorithm which embeds the prior experience into a neural network dynamics model and allows the use of model predictive control planning by adapting it through continuous refitting of a locally linear dynamics model. Simple linear models that are already capable of representing the coarse local dynamics of the system for a large span of manipulation tasks are enhanced with neural networks. The policy is determined by means of iterative linear quadratic regulator. Linear regression fit is achieved by fitting a Gaussian model to the dataset and normal inverse-Wishart prior on this Gaussian to obtain a prior model. The parameters of normal-inverse-Wishart-distribution are updated with the collected data during the course execution. While prior parameters can be obtained by fitting a Gaussian mixture model over the obtained data, neural networks have proven to exhibit much more powerful representational capacity for complex state spaces. Fu and Levine employ a two layered neural network architecture which maps steps and actions from the current and previous time step to an acceleration term that is integrated via semi-implicit integration rule to predict the states of the next time step.

In order to ensure the validity of linear models for the new controller, Levine et al. [LWA15] extends the idea of trajectory optimization by augmenting the regular cost term with an entropy maximization term and thus limiting the deviation of the new trajectory from the previous trajectory distribution. In addition, based on the principle that data points at nearby time steps and prior iterations are induced by roughly similar dynamics, number of points used for training are reduced by clustering the data via GMMs. Each cluster is modelled by piecewise linear-Gaussian dynamics and used as prior by finding the weighted mean and covariance of averaged cluster points.

Since local time-varying linear models fail to represent system dynamics in the presence of discontinuities, Levine, Waager and Abbeel [CKY+16] present a method that combines path integrals (PI2) with guided policy search (GPS). In order to sustain learning on new task instances, a sampling approach named mirror descent guided policy search (MDGPS) alternating between solving the optimization for the local policy and global policy by minimizing the KL-divergence is proposed. PI2 is used only to learn the feedforward terms, hence the number of policy samples is reduced in proportion to learned parameter. Montgomery et al. [MAF+16] describes an approach that allows the use of random initial states where samples are drawn from the global neural network policy instead of local policies. Subsequently, these samples are clustered to perform local policy for each cluster. The local policies serve as a supervisor to update the global policy in the supervised learning phase. Since deep reinforcement algorithms require large training times due to sample-complexity, off-policy deep Q-function derived algorithms, namely Deep Deterministic Policy Gradient (DDPG) and Normalized Advantage Function (NAF) can be trained in a parallel manner across multiple computers [GHLL16].

DDPG adapts actor-critic method, while NAF employs a special form of Q-function to allow closed-form solutions similar to discrete action models.
Lowrey et al. [LKD+18] offers a modified version of natural policy gradient algorithm that is applied on a simulated model and later transferred to the real system. Additionally, it is also demonstrated that training with an ensemble of models provide robustness against inaccuracies due to modelling errors. Besides alleviating difficulties arising from the parametrization of the nonlinear physical system, ensemble methods with conservative policies allow more appropriate data collection to determine the actual model.
In order to make use of the known approximate model of the actual system, Abbeel et al. [AQN06] presents a hybrid algorithm where the policy is evaluated on the real system and the derivate with respect to the policy parameters are computed based on the simulator. At each iteration the model is updated by adding a time-dependent bias term corresponding to the discrepancy between the real and original model. Hence, the derivatives are computed along the correct state-action trajectory.

1.2 Structure of this Work

The second chapter provides the fundamental theoretical background behind the algorithms employed in this thesis. In the third chapter, the results of the simulated and physical experiments are presented. Within the same chapter, the outcomes of the experiments are discussed. Chapter 4 concludes the thesis.

Chapter 2

Theory and Methods

This chapter presents the necessary theoretical background that constitutes the backbone of the algorithms implemented in this thesis. Section 2.1 gives a brief overview of the fundamental notions and approaches in the field of reinforcement learning. Section2.2 introduces Probabilistic Inference for Learning Control (PILCO) algorithm which is followed by a broad explanation of sparse Gaussian Process in section 2.3. The chapter continues with the presentation of Policy Improvement with REsidual Model Learning (PIREM) method in section 2.4. The contribution of this thesis is the combination of PILCO and PIREM methods with iterative Linear Quadratic Regulator controller, which is described in Section 2.5. The chapter concludes with the explanation of two contributions of this thesis, namely PILCO-iLQR and PIREM-iLQR algorithms 2.6.

2.1 Reinforcement Learning

Reinforcement learning is a process of discovering the optimal **policy** to obtain maximum cumulative **reward** through a series of trial-and-error search. A policy can be defined as a function mapping **states** to **actions** to be taken in the next time step. A reward signal indicates how good or bad the action is based on the state of the **agent** and **environment**.
The notions of action, state and model are commonly shared by the fields of reinforcement learning and optimal control theory. However, optimal control requires a perfect model of the system which yields the next state given the current state and action. This property of optimal control leads to wrong results if the system dynamics is inaccurate. On the contrary, reinforcement learning directly utilizes the state measurements and the reward obtained from the environment [Pow12]. The rest of this section is build upon the Sutton and Barto [SB98] and Kober et al. [KABP13].
The fundamental assumption behind the majority of reinforcement algorithms is the adoption of *Markov Decision Process* in desicion-making process. The Markov property implies that the reward-state pair at the next time step (s_t, r_t) solely depends

on the states (s_t) and actions (a_t) at the current time instant. Therefore, rather than considering the past state and action values at the previous time steps, the policy takes the current state and action into account to maximize the cumulative reward. The cumulative reward function to be maximized can be organized in various ways. It can contain information about the average reward within a finite horizon, sum of discounted reward values over an infinite horizon or simply the sum of reward values over a finite horizon.

Due to the absence of explicit training data, an agent in a reinforcement learning task needs to explore the environment with a series of trials. System dynamics and reward mechanism of the environment need to be discovered by visiting states that the agent is uncertain about. However, in order to obtain a stable policy, the agent may also maintain the current behaviour as it becomes more certain about the optimality of its behaviour. Such balance between discovery and adherence to the existing strategy is called *exploration* and *exploitation*. The problem emerging in many reinforcement learning tasks is the exponential increase in the required amount of data to properly explore the state and action spaces. For instance, if 10 datapoints are necessary to cover the one-dimensional state space, the number of datapoints to cover two-dimenisonal state space with the same density of data becomes 100. For n-dimensional space, this number becomes 100^n. This phenomenon is referred to as the *curse of dimensionality*.

The objective in reinforcement learning is to find an optimal policy π^* that maximizes the total reward. Defining a probability function $\pi(s,a) = P(a|s)$ and letting μ^π represent the state distribution based on the environment dynamics when applying the policy π, the optimization problem can be formulated as:

$$\begin{aligned} \max_{\pi} J(\pi) &= \sum_{s,a} \mu^\pi(s)\pi(s,a)R(s,a) \\ \text{s.t. } \mu^\pi(s') &= \sum_{s,a} \mu^\pi(s)\pi(s,a)T(s,a,s'), \forall s' \in S \\ 1 &= \sum_{s,a} \mu^\pi(s)\pi(s,a) \\ \pi(s,a) &\geq 0, \forall s \in S, a \in A \end{aligned} \tag{2.1}$$

where the transition probability function $T(s,a,s')$ denotes the likelihood of reaching the state s' by taking the action a at state s. Solving this problem in primal form as shown above is referred to *policy search*, whereas solution in dual form is named as *value-function based approach*. Solving this problem in dual form by using Lagrange multiplier $V^\pi(s')$ and finding the extrema with respect to $\mu^\pi(s)\pi(s,a)$ gives

$$R(s,a) + \sum_{s'} V^\pi(s')\,T(s,a,s') - V^\pi(s) = 0. \tag{2.2}$$

Collecting the terms except $V^{\pi}(s)$ on the right side of the equation yields

$$V^{*}(s) = \max_{a^{*}} \left[R\left(s, a^{*}\right) + \gamma \sum_{s'} V^{*}\left(s'\right) T\left(s, a^{*}, s'\right) \right]. \tag{2.3}$$

The problem boils down to finding the optimal action a^{*} for each state. The Lagrange multiplier $V^{\pi}\left(s'\right)$ is called the value function. While policy search algorithms attempt to directly update the policy, **value function methods** try to approximate the value function and use it to find the optimal policy in an indirect manner. Rather than operating with the value function, many prominent algorithms consider so-called state-action value function $Q^{\pi}(s, a)$, which is a measure for the utility of taken the action a at state s:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s'} V^{\pi}\left(s'\right) T\left(s, a, s'\right) \tag{2.4}$$

For discrete state-action spaces with low dimensions, the value function and corresponding actions can be stored in tables. However, large and contionues states require sophisticated dimensionality reduction methods. Value function based algorithms which aim to find the optimal policy by value function evaluation can be classified into three categories: Dynamic Programming-based Methods, Monte Carlo Methods and Temporal Difference Methods.

Dynamic Programming-based Methods are model-based approaches that employ the state transition probabilites $T\left(s', a, s\right)$ and reward function $R(s, a)$ to estimate the value function. Policy iteration and value iteration are the most well-known algorithms under this category. Policy iteration switches between two stages named policy evaluation and policy improvement. After random initialization of the policy, the value of each state is recalculated depending on the current value of the successor state, immediate reward and the policy. The value of each state is updated with the same procedure until steady values are obtained. Subsequently, the policy greedly picks the optimum action based on the new value functions. The alternation between policy evaluation and improvement is repeated until the policy converges.

Monte Carlo Methods propose an alternative model-free approach to determine the value function at the policy evaluation stage. The value functions are updated by considering the amount of visits to each state and the total reward received from the first visit until the end of the episode. The most simplistic version of this approach can be to set the value function equal to the total reward divided by the number of visits.

Temporal Difference Methods differ from the Monte Carlo methods in terms of the horizon of the reward. While Monte Carlo methods need to wait until the end of episode, temporal difference methods solely consider the immediate reward and the value of the successor state to evaluate the policy. Being a model-free method, temporal difference methods can be seen as a combination of Monte Carlo and dynamic programming-based approaches. Within this framework, the value of a state

can be updated as follows:

$$V(s) \leftarrow V(s) + \alpha \left(R(s,a) + \gamma V\left(s'\right) - V(s)\right) \tag{2.5}$$

where α is the learning rate and (s' is the successor state. This version of the algorithm which contains information about the next step is called TD(0). A variant of the same method which considers all time steps until the end of the episode is named TD(λ). Defining a return value which contains the sum of n future rewards with a decaying weight

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V\left(S_{t+n}\right) \tag{2.6}$$

and a weighting factor $(1-\lambda)\lambda^{n-1}$ for each of the n-step return, the total weighted sum of n-step returns can be expressed as

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}. \tag{2.7}$$

Similar to TD(0), the value function can be updated in the direction of the forward λ-return:

$$V(s) \leftarrow V(s) + \alpha \left(G_t^{\lambda} + \gamma V\left(s'\right) - V(s)\right) \tag{2.8}$$

Just as value functions, state-action values $Q(s,a)$ can be updated with temporal difference methods as well. *SARSA* algorithm estimates the new $Q(s,a)$ value towards the maximum $Q\left(s',a'\right)$ at the following time instant:

$$Q'(s,a) \leftarrow Q(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q\left(s',a'\right) - Q(s,a)\right) \tag{2.9}$$

Instead of estimating the value function first, **policy search** methods intent to update policy parameters directly. Compared to value function approaches, they exhibit more stable convergence behaviors with local optimization around the current policy and facilitate the integration of expert knowledge through policy initialization. Since a small variation in the policy may lead to significant change in the value functions, the new trajectory may quickly diverge from the previous one and cause unforeseen instabilities in the real robot. Policy-based approaches reduce this risk with gradual and local updates in the policy parameters. In addition, they allow for stochastic policies and easy adaption of constraints such as control limits or upper bounds of the change in the state. Policy functions can be parametrized with fewer parameters compared to value functions used in value-based approaches, which may reduce space consumption and computational complexity.
One of the generic policy-search algorithms algorithms is called Monte Carlo Policy Gradient or *REINFORCE* [Wil92]. In order to find the policy that maximizes the expected return, policy parameters θ can be updated in the direction of steepest ascent in expected return:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J. \tag{2.10}$$

Note that the episodic return is the sum of expected immediate returns with the trajectory distribution $P^\theta(\tau) = P(\tau|\theta)$:

$$J^\theta = \sum_\tau P^\theta(\tau) J^\tau \tag{2.11}$$

The derivative of the episode distribution can be reformulated as

$$\nabla_\theta P^\theta(\tau) = P^\theta(\tau) \nabla_\theta \log P^\theta(\tau) \tag{2.12}$$

where the last term $\nabla_\theta \log P^\theta(\tau)$ is called the *score function*. Therefore the derivative of the expected return with respect to the policy parameters is calculated as

$$\begin{aligned} \nabla_\theta J^\theta &= \sum_\tau \nabla_\theta P^\theta(\tau) J^\tau = \sum_\tau P^\theta(\tau) \nabla_\theta \log P^\theta(\tau) J^\tau \\ &= E\left\{ \nabla_\theta \log P^\theta(\tau) J^\tau \right\}. \end{aligned} \tag{2.13}$$

Since the immediate reward does not depend on the future actions, the return in the above equation can be replaced with $Q^\pi(s, a)$. The variance in policy update can be reduced by substracting a *baseline* function without changing the expected value. A family of algorithms named *actor-critic* methods aims to combine the benefits of value-function based and policy-based approaches. While the *critic* attempts to update the action-state function $Q(s, a)$, actor updates the policy parameters based on the new values suggested by the critic.
Reinforcement learning in the context of robotics brings about additional challenges due to high-dimensional, continuous states and actions [KABP13]. In addition, measurements generally obtain noise and system states are partially observable. Hence, the agent cannot know its precise state and the reinforcement learning algorithm ought to be modelled as partially observed and combined with filtering methods. Therefore, it is usually applicable to store the uncertainty information as a mean and variance value instead of pure measured data.
Experiments on real robots are generally expensive and tedious. Small errors in modelling of the robot may accumulate throughout an episode, or *rollout*. Thus, policy functions should be under-modelled to provide robustness against system uncertainty. A further aspect of challenges in robotic tasks is called *reward shaping*, which implies the specification a proper reward function to achieve fast learning algorithms to minimize the real-world costs. Many algorithms used in robotic applications are model-based and learning systems often utilise policy search methods rather than value function-based approaches.

2.2 Probabilistic Inference for Learning Control (PILCO)

Many reinforcement learning approaches require large number of trials to learn. Due to this data inefficiency, huge amounts of interactions are necessary to find the optimum policy for systems with high dimensions and complicated dynamics. Hence,

learning in real robotic systems becomes impractical and time consuming without additional expertise, simulators, prior knowledge on the system, pre-defined policies and extracting more information on existing data.

PILCO [DFR15] presents a model-based algorithm to extract useful information from existing data. System uncertainity is explicitly incorporated into trajectory planning and control actions by modelling the system dynamics with a nonparametric Gaussian process. Since existing data might be generated by various underlying system dynamics, strictly relying on a single deterministic model leads to inaccurate long-term predictions beyond the domain of training data. This problem is called model-bias and adaptation of Gaussian processes into transition dynamics can alleviate it by taking all acceptable functions into account during decision making process.

The PILCO algorithm can be decomposed into three modules: Initially, a probabilistic GP model is learnt based on the data obtained after applying control signals via randomly initialized policy parameters. Subsequently, long-term prediction $p(\boldsymbol{x}_1|\pi), \ldots, p(\boldsymbol{x}_T|\pi)$ of the state is received via approximate inference given an initial state. The expected long-term cost $J^\pi(\boldsymbol{\theta})$ is evaluated during approximate inference steps to update the parameters θ through gradient-based policy improvement by calculating $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$. The PILCO algorithm is summarized in 2.2.

Given the system dynamics

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}, \quad \boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_w) \tag{2.14}$$

with states $\mathbf{x} \in \mathbb{R}^n$ and controls $\mathbf{u} \in \mathbb{R}^m$, independent and identically distributed Gaussian noise $\mathbf{w}$ and unknown system transition function $\mathbf{f}$, the goal of PIREM is to find an optimum policy $\pi : \boldsymbol{x} \mapsto \pi(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{u}$, such that the following long-term cost is minimized:

$$J^\pi(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathrm{E}_{\boldsymbol{x}_t}\left[c(\boldsymbol{x}_t)\right], \quad \boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \tag{2.15}$$

where π is is the policy function parametrized by θ.

2.2.1 Gaussian Process Regression

A regression is the statistical process of estimating a function h that maps inputs $\mathbf{x}_i \in \mathbb{R}^D$ to the observations $y_i = h(\phi, \mathbf{x}_i) + \varepsilon_i \in \mathbb{R}$ with a noise term ε_i. To be more precise, the aim of regression is to find the set of parameters ϕ^* that describes the relation between the input and corresponding observations. Gaussian process regression is a a non-parametric regression method, where the values of the observed datapoints act as the parameters themeselves. As more data collected, the number

Algorithm 1 Probabilistic Inference for Learning Control

procedure PILCO
init: Sample controller parameters $\theta \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Apply random control signals and record data
repeat
  Learn GP hyperparameters via evidence maximization
  repeat
    Approximate inference for policy evaluation, get $J^\pi(\boldsymbol{\theta})$
    Gradient-based policy improvement by getting $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$
    Update parameters θ
  until convergence; **return** θ^*
  Set $\pi^* \leftarrow \pi\left(\boldsymbol{\theta}^*\right)$
  Apply π^* to the system and record data
until task learned
end procedure

of parameters grows and the computational cost for estimating the function value for a given query point increases accordingly. A Gaussian process can be formally defined as follows:

Definition 1. A *Gaussian process* is a collection of random variables, any number of which have a consistent joint Gaussian distribution [RW05].
Within the PILCO framework, system is modelled as Gaussian process, which estimates the temporal difference $\Delta_t = x_{t+1} - x_t$ given the input tuple $(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Considering zero prior mean, the covariance function, or the kernel, is defined as

$$k\left(\tilde{\boldsymbol{x}}_p, \tilde{\boldsymbol{x}}_q\right) = \sigma_f^2 \exp\left(-\frac{1}{2}\left(\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{x}}_q\right)^\top \boldsymbol{\Lambda}^{-1}\left(\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{x}}_q\right)\right) + \delta_{pq}\sigma_w^2 \tag{2.16}$$

where $\tilde{\boldsymbol{x}} := \left[\boldsymbol{x}^\top \boldsymbol{u}^\top\right]^\top$ and $\Lambda := \operatorname{diag}\left(\left[\ell_1^2, \ldots, \ell_{D+F}^2\right]\right)$. The posterior GP hyperparameters Λ, σ_f^2 and σ_w^2 are learned by evidence maximization, where Λ is a diagonal matrix representing the length-scales, σ_f^2 denotes the signal variance and σ_w^2 signifies the noise variance. Specifying the hyper-parameters with a vector $\boldsymbol{\theta}$, evidence maximization aims to find the parameters that maximize the marginal likelihood of the observed datapoints through a gradient ascent method:

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \tag{2.17}$$

Given data points at the current instant, the marginal distribution of the successor state is Gaussian distributed:

$$p\left(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t\right) = \mathcal{N}\left(\boldsymbol{x}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}\right) \tag{2.18a}$$

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{x}_t + \mathrm{E}_f\left[\boldsymbol{\Delta}_t\right], \quad \boldsymbol{\Sigma}_{t+1} = \operatorname{var}_f\left[\boldsymbol{\Delta}_t\right] \tag{2.18b}$$

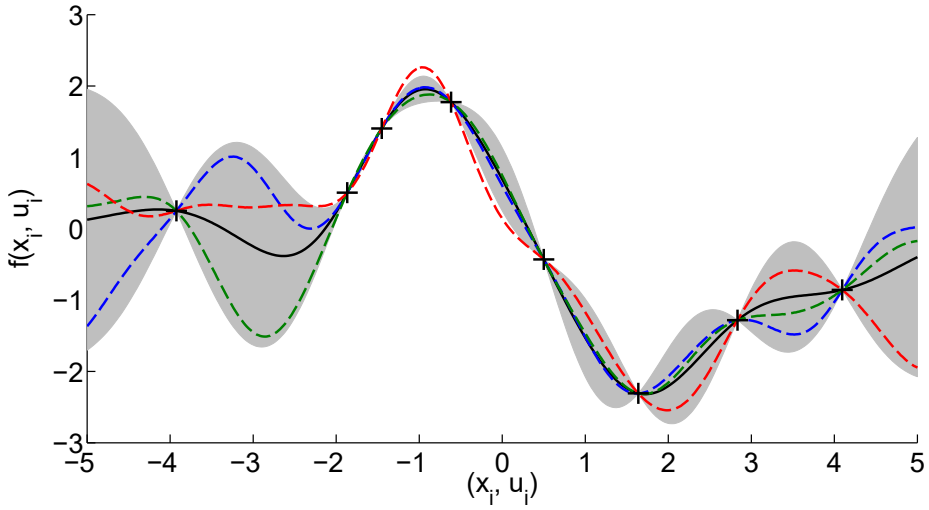


Figure 2.1: Since observed data points can be generated by many possible functions, probabilistic modelling of system dynamics is a reasonable way of expressing uncertainity. [DFR15] .

where the mean and covariance are equal to

$$\mathbb{E}_f\left[\boldsymbol{\Delta}_t\right] = m_f\left(\tilde{\boldsymbol{x}}_t\right) = \boldsymbol{k}_*^\top \left(\boldsymbol{K} + \sigma_w^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} = \boldsymbol{k}_*^\top \boldsymbol{\beta} \tag{2.19a}$$

$$\operatorname{var}_f\left[\boldsymbol{\Delta}_t\right] = k_{**} - \boldsymbol{k}_*^\top \left(\boldsymbol{K} + \sigma_w^2 \boldsymbol{I}\right)^{-1} k_* \tag{2.19b}$$

with $k_* := k\left(\tilde{X}, \tilde{x}_t\right), k_{**} := k\left(\tilde{x}_t, \tilde{x}_t\right)$, $\boldsymbol{\beta} := \left(\boldsymbol{K} + \sigma_w^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}$, and $\boldsymbol{K}$ as a kernel matrix with entries $K_{ij} = k\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j\right)$. For multidimensional targets, conditionally independent Gausian processes are trained for each dimension.

2.2.2 Long-term predictive distribution

In order to compute and minimize the total cost, long-term probability distribution of the state trajectory $p\left(\boldsymbol{x}_1|\pi\right), \ldots, p\left(\boldsymbol{x}_T|\pi\right)$ needs to be obtained. To evaluate this, the joint probabilty distribution of the state-control pair at the previous time step has to be propagated. Since the control $\boldsymbol{u}_t = \pi\left(\boldsymbol{x}_t, \boldsymbol{\theta}\right)$ is a function of state, a joint distribution $p\left(\tilde{\boldsymbol{x}}_t\right) = p\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right)$ can be approximated as a Gaussian distribution. To compute the state at the next time step, the probability for the temporal difference needs to computed, which requires the integral

$$p\left(\boldsymbol{\Delta}_t\right) = \iint p\left(f\left(\tilde{\boldsymbol{x}}_t\right)|\tilde{\boldsymbol{x}}_t\right) p\left(\tilde{\boldsymbol{x}}_t\right) \mathrm{d}f \mathrm{d}\tilde{\boldsymbol{x}}_t \tag{2.20}$$

Since the integral with respect to the random variable $\tilde{\boldsymbol{x}}_t$ and the random function f is analytically intractable, it is approximated by a Gaussian via a method called moment matching as shown in Fig. 2.2 . After the mean $\boldsymbol{\mu}_\Delta$, covariance $\boldsymbol{\Sigma}_\Delta$ and

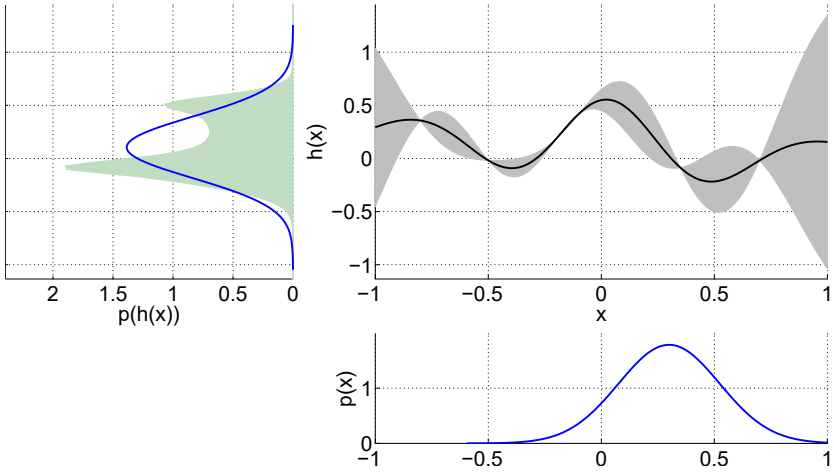


Figure 2.2: Estimation from an uncertain input. Prediction from uncertain state-control pair (lower figure) to obtain long-term predictive trajectory (upper-right figure) is achieved by approximating the probability $p(\boldsymbol{\Delta}_t)$ by a Gaussian. The green field illustrates the true computationally intractable distribution (upper-left figure), while the blue curve represents the approximate distribution obtained via moment matching [DFR15].

the covariance between $\boldsymbol{\Delta}_t$ and $\boldsymbol{x}_t$ are evaluated with moment matching [CGLR03], the mean and the variance terms for the next time step can be written as

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\mu}_\Delta \tag{2.21}$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_\Delta + \operatorname{cov}\left[\boldsymbol{x}_t, \boldsymbol{\Delta}_t\right] + \operatorname{cov}\left[\boldsymbol{\Delta}_t, \boldsymbol{x}_t\right] \tag{2.22}$$

Adapting the law of iterated expectations, the predictive mean for target dimensions $a = 1, \ldots, D$ can be calculated as

$$\begin{aligned} \mu_\Delta^a &= \mathrm{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathbb{E}_{f_a}\left[f_a\left(\tilde{\boldsymbol{x}}_t\right)|\tilde{\boldsymbol{x}}_t\right]\right] = \mathbb{E}_{\tilde{\boldsymbol{x}}_t}\left[m_{f_a}\left(\tilde{\boldsymbol{x}}_t\right)\right] \\ &= \int m_{f_a}\left(\tilde{\boldsymbol{x}}_t\right)\mathcal{N}\left(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t\right)\mathrm{d}\tilde{\boldsymbol{x}}_t = \boldsymbol{\beta}_a^\top \boldsymbol{q}_a, \\ \boldsymbol{\beta}_a &= \left(\boldsymbol{K}_a + \sigma_{w_a}^2\right)^{-1}\boldsymbol{y}_a \end{aligned} \tag{2.23}$$

where $\boldsymbol{q}_a \in \mathbb{R}^n$ is obtained by multiplication and intergration over Gaussian functions:

$$\begin{aligned} q_{a_i} &= \int k_a\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_t\right)\mathcal{N}\left(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t\right)\mathrm{d}\tilde{\boldsymbol{x}}_t \\ &= \sigma_{f_a}^2\left|\tilde{\boldsymbol{\Sigma}}_t\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{I}\right|^{-\frac{1}{2}}\exp\left(-\frac{1}{2}\boldsymbol{\nu}_i^\top\left(\tilde{\boldsymbol{\Sigma}}_t + \boldsymbol{\Lambda}_a\right)^{-1}\boldsymbol{\nu}_i\right) \end{aligned} \tag{2.24}$$

with $\boldsymbol{\nu}_i$ equal to the centralized training inputs $(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t)$. The elements of the covariance matrix $\boldsymbol{\Sigma}_\Delta \in \mathbb{R}^{D\times D}$ are computed via the covariance decomposition

formula:

$$\begin{aligned}\sigma_{aa}^2 &= \mathbb{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathrm{var}_f\left[\Delta_a|\tilde{\boldsymbol{x}}_t\right]\right] + \mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}\left[\Delta_a^2\right] - \left(\boldsymbol{\mu}_\Delta^a\right)^2 \\ \sigma_{ab}^2 &= \mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}\left[\Delta_a\Delta_b\right] - \boldsymbol{\mu}_\Delta^a\boldsymbol{\mu}_\Delta^b, \quad a \neq b\end{aligned} \tag{2.25}$$

Since target dimensions are conditionally independent, the term $\mathbb{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathrm{var}_f\left[\Delta_a|\tilde{\boldsymbol{x}}_t\right]\right]$ vanishes at the off-diagonal terms. Using the law of iterated expectations, the term $\mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}\left[\Delta_a\Delta_b\right]$ can be obtained as

$$\begin{aligned}\mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}\left[\Delta_a\Delta_b\right] &= \mathrm{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathbb{E}_f\left[\Delta_a|\tilde{\boldsymbol{x}}_t\right]\mathbb{E}_f\left[\Delta_b|\tilde{\boldsymbol{x}}_t\right]\right] \\ &= \int m_f^a\left(\tilde{\boldsymbol{x}}_t\right) m_f^b\left(\tilde{\boldsymbol{x}}_t\right) p\left(\tilde{\boldsymbol{x}}_t\right) \mathrm{d}\tilde{\boldsymbol{x}}_t\end{aligned} \tag{2.26}$$

The mean values in the integral are known from the GP prediction procedure. Thus, standard multiplication and intergration over Gaussian functions gives

$$\begin{aligned}&\mathbb{E}_{f,\tilde{\boldsymbol{x}}_t}\left[\Delta_a\Delta_b\right] = \boldsymbol{\beta}_a^\top\boldsymbol{Q}\boldsymbol{\beta}_b \\ &Q := \int k_a\left(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{X}}\right)^\top k_b\left(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{X}}\right) p\left(\tilde{\boldsymbol{x}}_t\right) \mathrm{d}\tilde{\boldsymbol{x}}_t\end{aligned} \tag{2.27}$$

The elements of $Q \in \mathbb{R}^{n\times n}$ are equal to

$$Q_{ij} = |\boldsymbol{R}|^{-\frac{1}{2}} k_a\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{\mu}}_t\right) k_b\left(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{\mu}}_t\right) \exp\left(\frac{1}{2}\boldsymbol{z}_{ij}^\top\boldsymbol{T}^{-1}\boldsymbol{z}_{ij}\right) \tag{2.28}$$

where

$$\begin{aligned}&\boldsymbol{R} := \tilde{\boldsymbol{\Sigma}}_t\left(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}\right) + \boldsymbol{I}, \boldsymbol{T} := \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1} + \tilde{\boldsymbol{\Sigma}}_t^{-1} \\ &\boldsymbol{z}_{ij} := \boldsymbol{\Lambda}_a^{-1}\boldsymbol{\nu}_i + \boldsymbol{\Lambda}_b^{-1}\boldsymbol{\nu}_j\end{aligned} \tag{2.29}$$

The additional term at the diagonal entries can be similarly found as

$$\mathrm{E}_{\tilde{\boldsymbol{x}}_t}\left[\mathrm{var}_f\left[\Delta_a|\tilde{\boldsymbol{x}}_t\right]\right] = \sigma_{f_a}^2 - \mathrm{tr}\left(\left(\boldsymbol{K}_a + \sigma_{w_a}^2\boldsymbol{I}\right)^{-1}\boldsymbol{Q}\right) + \sigma_{w_a}^2 \tag{2.30}$$

The cross-covariance term $\mathrm{cov}\left[\boldsymbol{x}_t, \boldsymbol{\Delta}_t\right]$ in 2.22 is equavalent to the cross-covariance between the temporal state difference $\boldsymbol{\Delta}_t = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$ and the state-action pair $\tilde{x}_t$, wihch can be formulated as

$$\mathrm{cov}\left[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t\right] = \mathrm{E}_{\tilde{\boldsymbol{x}}_t, f}\left[\tilde{\boldsymbol{x}}_t\boldsymbol{\Delta}_t^\top\right] - \tilde{\boldsymbol{\mu}}_t\boldsymbol{\mu}_{\boldsymbol{\Delta}}^\top \tag{2.31}$$

The mean of the state difference $\boldsymbol{\mu}_\Delta$ is computed in 2.23, while $\tilde{\mu}_t$ is obtained from the joint probability of the state-action pair based on the policy function. Applying the Adam's law, the first term $\mathbb{E}_{\tilde{\boldsymbol{x}}_t, f}\left[\tilde{\boldsymbol{x}}_t\Delta_t^a\right]$ can be given as

$$\begin{aligned}\mathrm{E}_{\tilde{\boldsymbol{x}}_t, f}\left[\tilde{\boldsymbol{x}}_t\Delta_t^a\right] = \mathrm{E}_{\tilde{\boldsymbol{x}}_t}\left[\tilde{\boldsymbol{x}}_t\mathbb{E}_f\left[\Delta_t^a|\tilde{\boldsymbol{x}}_t\right]\right] &= \int \tilde{\boldsymbol{x}}_t m_f^a\left(\tilde{\boldsymbol{x}}_t\right) p\left(\tilde{\boldsymbol{x}}_t\right) \mathrm{d}\tilde{\boldsymbol{x}}_t \\ &= \int \tilde{\boldsymbol{x}}_t\left(\sum_{i=1}^n \beta_{a_i} k_f^a\left(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_i\right)\right) p\left(\tilde{\boldsymbol{x}}_t\right) \mathrm{d}\tilde{\boldsymbol{x}}_t\end{aligned} \tag{2.32}$$

By reformulating the covariance as Gaussian and swapping the integral and summation, the above equation can be expressed as

$$\begin{aligned}&\mathbb{E}_{\tilde{\boldsymbol{x}}_t,f}\left[\tilde{\boldsymbol{x}}_t\Delta_t^a\right]\\&=\sum_{i=1}^{n}\beta_{a_i}\int\tilde{\boldsymbol{x}}_t c_1\mathcal{N}\left(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{x}}_i,\boldsymbol{\Lambda}_a\right)\mathcal{N}\left(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t,\tilde{\boldsymbol{\Sigma}}_t\right)\mathrm{d}\tilde{\boldsymbol{x}}_t\end{aligned}\tag{2.33}$$

where c_1 is equivalent to $\sigma_{f_a}^2(2\pi)^{\frac{D+F}{2}}\left|\Lambda_a\right|^{\frac{1}{2}}$. The product of two Gaussians generates another Gaussian $c_2^{-1}\mathcal{N}\left(\tilde{x}_t|\psi_i,\Psi\right)$ with

$$\begin{aligned}c_2^{-1}&=(2\pi)^{-\frac{D+F}{2}}\left|\boldsymbol{\Lambda}_a+\tilde{\boldsymbol{\Sigma}}_t\right|^{-\frac{1}{2}}\\&\quad\times\exp\left(-\frac{1}{2}\left(\tilde{\boldsymbol{x}}_i-\tilde{\boldsymbol{\mu}}_t\right)^{\top}\left(\boldsymbol{\Lambda}_a+\tilde{\boldsymbol{\Sigma}}_t\right)^{-1}\left(\tilde{\boldsymbol{x}}_i-\tilde{\boldsymbol{\mu}}_t\right)\right)\\\boldsymbol{\Psi}&=\left(\boldsymbol{\Lambda}_a^{-1}+\tilde{\boldsymbol{\Sigma}}_t^{-1}\right)^{-1},\quad\boldsymbol{\psi}_i=\boldsymbol{\Psi}\left(\boldsymbol{\Lambda}_a^{-1}\tilde{\boldsymbol{x}}_i+\tilde{\boldsymbol{\Sigma}}_t^{-1}\tilde{\boldsymbol{\mu}}_t\right)\end{aligned}\tag{2.34}$$

The integral yields the expected value of the new Gaussian when the terms independent of $\widetilde{x}_t$ are pulled out of the integration. After carrying out the integration and a couple of simplifications, the cross-covariance can be expressed as

$$\operatorname{cov}\tilde{\boldsymbol{x}}_t,f\left[\tilde{\boldsymbol{x}}_t,\Delta_t^a\right]=\sum_{i=1}^{n}\beta_{a_i}q_{a_i}\tilde{\boldsymbol{\Sigma}}_t\left(\tilde{\boldsymbol{\Sigma}}_t+\boldsymbol{\Lambda}_a\right)^{-1}\left(\tilde{\boldsymbol{x}}_i-\tilde{\boldsymbol{\mu}}_t\right)\tag{2.35}$$

2.2.3 Gradients for Policy Improvement

Proper selection of cost and policy function allows us to compute the smooth gradients of $\mathrm{d}J^{\pi}(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$ in order to minimize the total expected cost. The derivative of the cost with respect to the policy parameters is computed by means of a series of chain rule. By defining $\xi_t:=\mathbb{E}_{\boldsymbol{x}_t}\left[c\left(\boldsymbol{x}_t\right)\right]$, the first step to drive the derivative is

$$\begin{aligned}\frac{\mathrm{d}J^{\pi}(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}}&=\sum_{t=1}^{T}\frac{\mathrm{d}\boldsymbol{\xi_t}}{\mathrm{d}\boldsymbol{\theta}}\\\frac{\mathrm{d}\boldsymbol{\xi_t}}{\mathrm{d}\boldsymbol{\theta}}&=\frac{\mathrm{d}\boldsymbol{\xi_t}}{\mathrm{d}p\left(\boldsymbol{x}_t\right)}\frac{\mathrm{d}p\left(\boldsymbol{x}_t\right)}{\mathrm{d}\boldsymbol{\theta}}:=\frac{\partial\boldsymbol{\xi_t}}{\partial\boldsymbol{\mu}_t}\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\theta}+\frac{\partial\boldsymbol{\xi_t}}{\partial\boldsymbol{\Sigma}_t}\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\theta}.\end{aligned}\tag{2.36}$$

Note that the mean $\boldsymbol{\mu}_t$ and covariance of the current state $(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)$ depend on the policy parameters $\boldsymbol{\theta}$ and previous time step's mean and covariance values $(\boldsymbol{\mu}_{t-1},\boldsymbol{\Sigma}_{t-1})$. This leads us to the following expansion of the above equation:

$$\begin{aligned}\frac{\mathrm{d}p\left(\boldsymbol{x}_t\right)}{\mathrm{d}\boldsymbol{\theta}}&=\frac{\partial p\left(\boldsymbol{x}_t\right)}{\partial p\left(\boldsymbol{x}_{t-1}\right)}\frac{\mathrm{d}p\left(\boldsymbol{x}_{t-1}\right)}{\mathrm{d}\boldsymbol{\theta}}+\frac{\partial p\left(\boldsymbol{x}_t\right)}{\partial\boldsymbol{\theta}}\\\frac{\partial p\left(\boldsymbol{x}_t\right)}{\partial p\left(\boldsymbol{x}_{t-1}\right)}&=\left\{\frac{\partial\boldsymbol{\mu}_t}{\partial p\left(\boldsymbol{x}_{t-1}\right)},\frac{\partial\boldsymbol{\Sigma}_t}{\partial p\left(\boldsymbol{x}_{t-1}\right)}\right\}\end{aligned}\tag{2.37}$$

The steps that are going to be shown below are applied on the mean $\boldsymbol{\mu}_t$, yet these are perfectly valid for $\boldsymbol{\Sigma}_t$ as well. The derivative of $\boldsymbol{\mu}_t$ with respect to the oplicy parameters can be formulated as

$$\frac{\mathrm{d}\mu_t}{\mathrm{d}\theta} = \frac{\partial \mu_t}{\partial \mu_{t-1}} \frac{\mathrm{d}\boldsymbol{\mu}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\Sigma}_{t-1}} \frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\theta}} \tag{2.38}$$

The term $\mathrm{d}p\left(\boldsymbol{x}_{t-1}\right)/\mathrm{d}\boldsymbol{\theta}$ is already computed in the previous time step. The derivative with respect to previous moments $\partial \boldsymbol{\mu}_t / \partial p\left(\boldsymbol{x}_{t-1}\right)$ can be calculated from equations obtained by moment matching. The last term $\partial \boldsymbol{\mu}_t / \partial \boldsymbol{\theta}$ can be analytically obtained from equations governing the nonlinear policy:

$$\frac{\partial \mu_t}{\partial \theta} = \frac{\partial \boldsymbol{\mu}_\Delta}{\partial \boldsymbol{\mu}_u} \frac{\partial \boldsymbol{\mu}_u}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{\mu}_\Delta}{\partial \boldsymbol{\Sigma}_u} \frac{\partial \boldsymbol{\Sigma}_u}{\partial \boldsymbol{\theta}} \tag{2.39}$$

2.2.4 Policy

In order to obtain the long-term state prediction, the policy should enable us to evaluate the joint state-control probability $p\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right)$. In addition, the control limits arising from the constraints of the physical system needs to be taken into account. If the control signal proposed by the preliminary policy $\tilde{\pi}\left(\boldsymbol{x}_t\right)$ is beyond the accepted range $[-\boldsymbol{u}_{\max}, \boldsymbol{u}_{\max}]$, the policy function should bring it within this range. This can be achieved by a squashing function which maps the outputs of the preliminary function into the desired range. The squashing function can be chosen as

$$\sigma(x) = \frac{9}{8}\sin(x) + \frac{1}{8}\sin(3x) \in [-1, 1]. \tag{2.40}$$

which is a suitable function for the analytical computation of the moments for the normally distributed state values. The output of the squashing function can be subsequently multiplied with the desired upper limits of control signals to extend the normalized limits.
The preliminary policy can be expressed as

$$\tilde{\pi}\left(\boldsymbol{x}_*\right) = \sum_{i=1}^{N} k\left(\boldsymbol{m}_i, \boldsymbol{x}_*\right)\left(\boldsymbol{K} + \sigma_\pi^2 \boldsymbol{I}\right)^{-1} \boldsymbol{t} = k\left(\boldsymbol{M}, \boldsymbol{x}_*\right)^\top \boldsymbol{\alpha} \tag{2.41}$$

where $\boldsymbol{x}_*$ is the current state, $\sigma_\pi^2 = 0.01$. The term $k\left(\boldsymbol{m}_i, \boldsymbol{x}_*\right)$ specifies the distance between the input and the center of a Gausssian basis function with mean $\boldsymbol{m_i}$:

$$k\left(\boldsymbol{x}_p, \boldsymbol{x}_q\right) = \exp\left(-\frac{1}{2}\left(\boldsymbol{x}_p - \boldsymbol{x}_q\right)^\top \boldsymbol{\Lambda}^{-1}\left(\boldsymbol{x}_p - \boldsymbol{x}_q\right)\right) \tag{2.42}$$

Therefore the preliminary policy is nothing but summation of N number of Gaussian basis functions The parameters that are updated throughout the learning process are $\boldsymbol{t}_i$, $\boldsymbol{\lambda}_i$ and $\boldsymbol{m}_i$ per target dimensions.

The long-term cost function is the sum of expected returns at each time instant. Hence, given a normally distributed input $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$, the predictive mean of the preliminary policy $\tilde{\pi}(\boldsymbol{x}_*)$ needs to be calculated:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}*}[\tilde{\pi}(\boldsymbol{x}_*)] &= \boldsymbol{\alpha}_a^\top \mathbb{E}_{\boldsymbol{x}_*}[k(\boldsymbol{M}, \boldsymbol{x}_*)] \\ &= \boldsymbol{\alpha}_a^\top \int k(\boldsymbol{M}, \boldsymbol{x}_*)\, p(\boldsymbol{x}_*)\, \mathrm{d}\boldsymbol{x}_* = \boldsymbol{\alpha}_a^\top \boldsymbol{r}_a\end{aligned} \tag{2.43}$$

where

$$\begin{aligned}r_{a_i} = &\left|\boldsymbol{\Sigma}_* \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{I}\right|^{-\frac{1}{2}} \\ &\times \exp\left(-\frac{1}{2}(\boldsymbol{\mu}_* - \boldsymbol{m}_i)^\top (\boldsymbol{\Sigma}_* + \boldsymbol{\Lambda}_a)^{-1} (\boldsymbol{\mu}_* - \boldsymbol{m}_i)\right)\end{aligned} \tag{2.44}$$

with $i = 1, \ldots, N$ denoting the number of radial basis functions and $a = 1, \ldots, F$ designating the policy dimension. The covariance between policy dimension a and b can be computed as

$$\begin{aligned}&\mathrm{cov}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*), \tilde{\pi}_b(\boldsymbol{x}_*)] \\ &= \mathrm{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)\, \tilde{\pi}_b(\boldsymbol{x}_*)] - \mathrm{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)]\, \mathrm{E}_{\boldsymbol{x}_*}[\tilde{\pi}_b(\boldsymbol{x}_*)].\end{aligned} \tag{2.45}$$

The terms $\mathrm{E}_{\boldsymbol{x}_*}[\tilde{\pi}(\boldsymbol{x}_*)]$ are already computed in 2.43, hence we are left with the term $\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)\, \tilde{\pi}_b(\boldsymbol{x}_*)]$:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}_*}[\tilde{\pi}_a(\boldsymbol{x}_*)\, \tilde{\pi}_b(\boldsymbol{x}_*)] &= \boldsymbol{\alpha}_a^\top \mathbb{E}_{\boldsymbol{x}_*}\left[k_a(\boldsymbol{M}, \boldsymbol{x}_*)\, k_b(\boldsymbol{M}, \boldsymbol{x}_*)^\top\right] \boldsymbol{\alpha}_b \\ &= \boldsymbol{\alpha}_a^\top \boldsymbol{Q} \boldsymbol{\alpha}_b\end{aligned} \tag{2.46}$$

The entries of $\boldsymbol{Q} \in \mathbb{R}^{N \times N}$ can be found as

$$\begin{aligned}Q_{ij} &= \int k_a(\boldsymbol{m}_i, \boldsymbol{x}_*)\, k_b(\boldsymbol{m}_j, \boldsymbol{x}_*)\, p(\boldsymbol{x}_*)\, \mathrm{d}\boldsymbol{x}_* \\ &= k_a(\boldsymbol{m}_i, \boldsymbol{\mu}_*)\, k_b(\boldsymbol{m}_j, \boldsymbol{\mu}_*)\, |\boldsymbol{R}|^{-\frac{1}{2}} \exp\left(\frac{1}{2} \boldsymbol{z}_{ij}^\top \boldsymbol{T}^{-1} \boldsymbol{z}_{ij}\right) \\ \boldsymbol{R} &= \boldsymbol{\Sigma}_* \left(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}\right) + \boldsymbol{I}, \quad \boldsymbol{T} = \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1} + \boldsymbol{\Sigma}_*^{-1} \\ \boldsymbol{z}_{ij} &= \boldsymbol{\Lambda}_a^{-1}(\boldsymbol{\mu}_* - \boldsymbol{m}_i) + \boldsymbol{\Lambda}_b^{-1}(\boldsymbol{\mu}_* - \boldsymbol{m}_j).\end{aligned} \tag{2.47}$$

Given the Gaussian of the current state, the procedure to compute the distribution of the successor state can be summarized as follows: Initially, the distribution of control signal $p(\boldsymbol{u}_{\max} \sigma(\tilde{\pi}(\boldsymbol{x}_t)))$ is analytically computed. As the next step, the joint state-control distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is obtained. This distribution is used to perform moment matching for the GP distribution of the change in state $p(\boldsymbol{\Delta}_t)$. Finally, the probability distribution of the next state $p(\boldsymbol{x}_{t+1})$ is computed.

2.2.5 Cost Function

Due to its mathematical property allowing a descent balance between exploration and exploitation, PILCO employs a saturating cost function in form

$$c(\boldsymbol{x}) = 1 - \exp\left(-\frac{1}{2\sigma_c^2} d\left(\boldsymbol{x}, \boldsymbol{x}_{\text{ target}}\right)^2\right) \in [0,1] \tag{2.48}$$

which behaves like a quadratic function in the vicinity of target $\boldsymbol{x}_{\text{ target}}$ and converges to 1 as the geometric distance d grows. The parameter specifies the width of the cost function which is visualized in Fig. 2.3. The expected values of the immediate cost can be computed by averaging the saturated cost over the probability distribution of $\boldsymbol{x}$:

$$\begin{aligned} &\mathrm{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] = \int c(\boldsymbol{x}) p(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \\ &= 1 - \int \exp\left(-\tfrac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}_{\text{ target}}\right)^{\top} \boldsymbol{T}^{-1}\left(\boldsymbol{x} - \boldsymbol{x}_{\text{ target}}\right)\right) p(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \end{aligned} \tag{2.49}$$

where $\boldsymbol{T}$ is the covariance of the unnormalized Gaussian with diagonal entries σ_c^2 or zero. Therefore, for a given $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ the average running cost is equal to

$$\begin{aligned} \mathrm{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] =& 1 - \left|\boldsymbol{I} + \boldsymbol{\Sigma}\boldsymbol{T}^{-1}\right|^{-1/2} \\ &\times \exp\left(-\frac{1}{2}\left(\boldsymbol{\mu} - \boldsymbol{x}_{\text{ target}}\right)^{\top} \tilde{\boldsymbol{S}}_1 \left(\boldsymbol{\mu} - \boldsymbol{x}_{\text{ target}}\right)\right) \end{aligned} \tag{2.50}$$

where the term $\tilde{S}_1$ is represents $T^{-1}\left(I + \Sigma T^{-1}\right)^{-1}$.

Saturated cost function in Fig. 2.3 promotes exploratory behaviours at the inital rollouts. Unvisited states with higher uncertainity generate lower expected cost due to the their wide tail at the region around target state. Hence the algorithm avoids certain states far from the target value by choosing uncertain states. As system dynamics are explored, the algorithm favors values with less variance around the target state, which is defined as exploitation in the field of reinforcement learning.

2.3 Sparse Gaussian Process

Non-parametric structure of Gaussian processes leads to a high training cost of $\mathcal{O}\left(N^3\right)$ and prediction cost of $\mathcal{O}\left(N^2\right)$, where N is the number of data points. Snelson and Ghahramini [SG06] present a new Gaussian process regression model to scale down the computational complexity by substituting N data points with fewer M psudo-input points. Thus, the training cost is reduced to $\mathcal{O}\left(M^2 N\right)$, while the prediction cost is decreased to $\mathcal{O}\left(M^2\right)$.
While the majority of previous studies tends to choose a subset of the obtained data based on an information criterion, active set selection undermines the learning procedure of kernel hyperparameters. The root cause for this is the non-smooth change

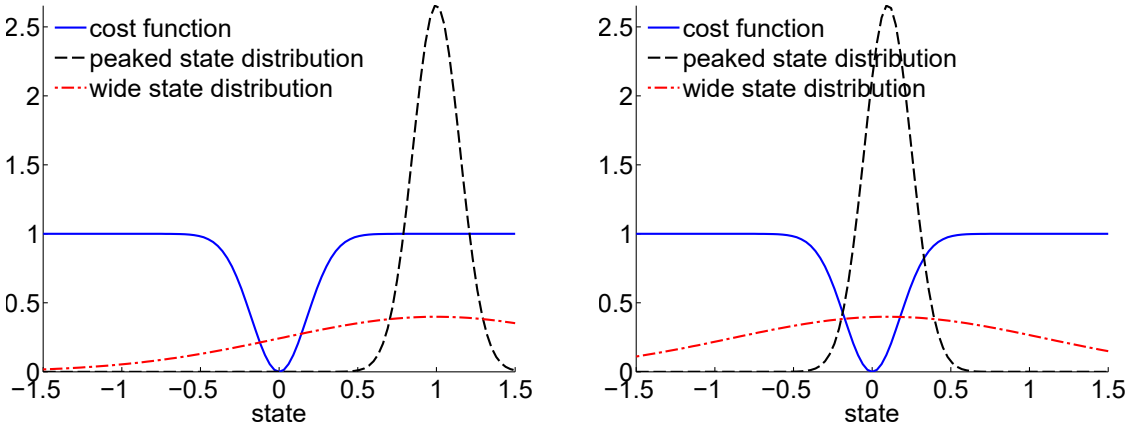


Figure 2.3: Saturated cost function promotes exploration of unvisited states at the early stages of the learning. States with high variance have larger tails at the target value, which leads to smaller expected costs compared to more certain states with less variance. As system dynamics are captured by the Gaussan process, the policy opts for states with higher confidence over the uncertain states.

in the gradient direction of the marginal likelihood, which impedes the convergence towards the optimal kernel parameters. To ovecome this problem, a single joint optimizer for generation of pseudo-inputs and finding corresponding hyperparameters is developed. The hyperparameters are determimed by the the location of the active set which consists of artificial pseudo-inputs instead of a subset of the existing data. As mentioned in the previous section, the distribution of the target value at a query point x with a dataset D comprising N data points $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$ and corresponding target values $\mathbf{y} = \{y_n\}_{n=1}^N$ is equal to

$$p(y|\mathbf{x}, \mathcal{D}, \boldsymbol{\theta}) = \mathcal{N}\left(y|\mathbf{k}_\mathbf{x}^\top \left(\mathbf{K}_N + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}, K_{\mathbf{xx}} - \mathbf{k}_\mathbf{x}^\top \left(\mathbf{K}_N + \sigma^2\mathbf{I}\right)^{-1}\mathbf{k}_\mathbf{x} + \sigma^2\right) \quad (2.51)$$

where $[\mathbf{k}_\mathbf{x}]_n = K(\mathbf{x}_n, \mathbf{x})$ and $K_{\mathrm{xx}} = K(\mathrm{x}, \mathrm{x})$. For pseudo dataset $\overline{D}$ of size M with pseudo inputs $\overline{\mathbf{X}} = \{\overline{\mathbf{x}}_m\}_{m=1}^M$ and corresponding targets $\overline{\mathbf{f}} = \{\overline{f}_m\}_{m=1}^M$, the target distribution for a single data point can be expressed as

$$p(y|\mathbf{x}, \overline{\mathbf{X}}, \overline{\mathbf{f}}) = \mathcal{N}\left(y|\mathbf{k}_\mathbf{x}^\top \mathbf{K}_M^{-1}\overline{\mathbf{f}}, K_{\mathbf{xx}} - \mathbf{k}_\mathbf{x}^\top \mathbf{K}_M^{-1}\mathbf{k}_\mathbf{x} + \sigma^2\right). \quad (2.52)$$

Note that the noise variance $\sigma^2\mathbf{I}$ is not included in the equation, because the pseudo-targets are not real observations $\overline{\mathbf{y}}$, but latent function values $\mathbf{f}$. The joint likelihood of the whole data is therefore equal to

$$p(\mathbf{y}|\mathbf{X}, \overline{\mathbf{X}}, \overline{\mathbf{f}}) = \prod_{n=1}^N p\left(y_n|\mathbf{x}_n, \overline{\mathbf{X}}, \overline{\mathbf{f}}\right) = \mathcal{N}\left(\mathbf{y}|\mathbf{K}_{NM}\mathbf{K}_M^{-1}\overline{\mathbf{f}}, \boldsymbol{\Lambda} + \sigma^2\mathbf{I}\right) \quad (2.53)$$

where $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda})$, $\lambda_n = K_{nn} - \mathbf{k}_n^\top \mathbf{K}_M^{-1}\mathbf{k}_n$ and $[\mathbf{K}_{NM}]_{nm} = K(\mathbf{x}_n, \overline{\mathbf{x}}_m)$.
To avoid seperate maximization this likelihood with respect to $\overline{X}$ and $\overline{f}$ separately,

the above likelihood can be integrated out with respect to the pseudo-targets by introducing the Gaussian prior:

$$p(\bar{\mathbf{f}}|\overline{\mathbf{X}}) = \mathcal{N}\left(\bar{\mathbf{f}}|\mathbf{0}, \mathbf{K}_M\right) \tag{2.54}$$

Hence, the posterior of the pseudo-targets $\bar{\mathrm{f}}$ can be expressed as

$$p(\bar{\mathbf{f}}|\mathcal{D}, \overline{\mathbf{X}}) = \mathcal{N}\left(\bar{\mathbf{f}}|\mathbf{K}_M\mathbf{Q}_M^{-1}\mathbf{K}_{MN}\left(\mathbf{\Lambda} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}, \mathbf{K}_M\mathbf{Q}_M^{-1}\mathbf{K}_M\right) \tag{2.55}$$

where $\mathbf{Q}_M = \mathbf{K}_M + \mathbf{K}_{MN}\left(\mathbf{\Lambda} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{K}_{NM}$. Predictive distribution given a query point x_* be computed by averaging over posterior over $\bar{\mathrm{f}}$:

$$p\left(y_*|\mathbf{x}_*, \mathcal{D}, \overline{\mathbf{X}}\right) = \int \mathrm{d}\bar{\mathbf{f}} p\left(y_*|\mathbf{x}_*, \overline{\mathbf{X}}, \bar{\mathbf{f}}\right) p(\bar{\mathbf{f}}|\mathcal{D}, \overline{\mathbf{X}}) = \mathcal{N}\left(y_*|\mu_*, \sigma_*^2\right), \tag{2.56}$$

where

$$\begin{aligned}\mu_* &= \mathbf{k}_*^\top\mathbf{Q}_M^{-1}\mathbf{K}_{MN}\left(\mathbf{\Lambda} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}\\ \sigma_*^2 &= K_{**} - \mathbf{k}_*^\top\left(\mathbf{K}_M^{-1} - \mathbf{Q}_M^{-1}\right)\mathbf{k}_* + \sigma^2.\end{aligned}$$

The operation $\Lambda+\sigma^2\mathbf{I}$ and $\mathbf{K}_{MN}\left(\mathbf{\Lambda} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{K}_{NM}$ to find $\mathbf{Q}_M$ consititutes the major portion of the total computational cost, which is $\mathcal{O}\left(M^2N\right)$.
Hyperparameters $\Theta = \{\boldsymbol{\theta}, \sigma^2\}$ and pseudo-inputs $\overline{X}$ can be found by performing gradient ascent to maximize the marginal likelihood:

$$\begin{aligned}p(\mathbf{y}|\mathbf{X}, \overline{\mathbf{X}}, \mathbf{\Theta}) &= \int \mathrm{d}\bar{\mathbf{f}}\, p(\mathbf{y}|\mathbf{X}, \overline{\mathbf{X}}, \bar{\mathbf{f}}) p(\bar{\mathbf{f}}|\overline{\mathbf{X}})\\ &= \mathcal{N}\left(\mathbf{y}|\mathbf{0}, \mathbf{K}_{NM}\mathbf{K}_M^{-1}\mathbf{K}_{MN} + \Lambda + \sigma^2\mathbf{I}\right)\end{aligned} \tag{2.57}$$

where the terms $\mathbf{K}_M$, $\mathbf{K}_{MN}$ and λ depend on the parameters $\overline{\mathbf{X}}$ and $\mathbf{\Theta}$.

2.4 Policy Improvement with Residual Model Learning (PI-REM)

Reinforcement learning in robotics and control brings about two handicaps: Continous and high dimensional state and action spaces require huge amount of interactions with the real environment which may cause wear and tear on the real robot. This is mostly overcome limiting the dimensionality of the search space by discrete parametrization of the policy. The second issue with RL methods is the need for large amount of rollouts due to the slow convergence property of the policy. Initialization of the policy through expert knowledge is a commonly used approach to reduce the number of trials.
Policy Improvement with Residual Model Learning (PI-REM) algorithm is a model-based method which utilizes the expert knowledge about the coarse dynamics of the

system. It accelerates the process of learning by combining the real sensory data with simulation results [SYFL17]. The key feature of the algorithm is to model the discrepancy between the real model and the known model *approximate model* as a Gaussian process. While representation via Gaussian Process reduces the model-bias problem with the integration of uncertainity, initial policy computed by means of the approximate model provides a plausible initial policy for the complicated real system. Besides enabling fast convergence towards an optimal policy, introducing approximate policy at the early stages reduces the risk of damages and instabilities caused by random initial policy. PILCO algorithm assumes that the real system is Gaussian distributed, on the other hand PI-REM models the residual difference between the real and simpler approximate model as Gaussian process. The superposition of the residual and approximate model alleviates the problems arising from unknown or mathematically complicated dynamics such as sensor noise, friction coefficient or deformations in the manipulated objects. Given a dynamical system

$$\boldsymbol{x}_{t+1}^r = \boldsymbol{f}^a\left(\boldsymbol{x}_t^r, \boldsymbol{u}_t^r\right) + \boldsymbol{f}^u\left(\boldsymbol{x}_t^r, \boldsymbol{u}_t^r\right) + \boldsymbol{w} \tag{2.58}$$

where the function $\boldsymbol{f}^a(\cdot,\cdot)$ denotes the fully described approximate model and $f^u(\cdot,\cdot)+w$ representing the unknown dynamics with noise, the objective of the Gaussian process is to capture the dynamics of the unknown model $f^u(\cdot,\cdot)+w$.

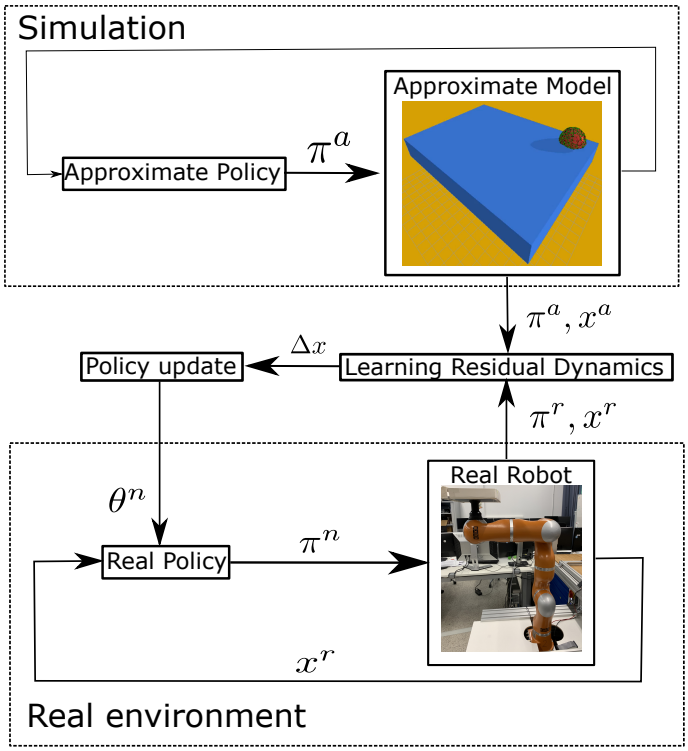


Figure 2.4: Overview of the PI-REM algorithm.

Using the definition $\tilde{\boldsymbol{x}}_t^a = (\boldsymbol{x}_t^a, \boldsymbol{u}_t^a)$, $\tilde{\boldsymbol{x}}_t^r = (\boldsymbol{x}_t^r, \boldsymbol{u}_t^r)$, $\Delta_t = (\Delta x_t, \Delta u_t)$ with $\Delta x_t = x_t^r - x_t^a$ and $\Delta u_t = u_t^r - u_t^a$, the real dynamics can be reformulated as

$$\boldsymbol{f}^r\left(\tilde{\boldsymbol{x}}_t^r\right) = \boldsymbol{f}^r\left(\tilde{\boldsymbol{x}}_t^a + \Delta_t\right) = \boldsymbol{f}^a\left(\tilde{\boldsymbol{x}}_t^a + \Delta_t\right) + \boldsymbol{f}^u\left(\tilde{\boldsymbol{x}}_t^a + \Delta_t\right) + \boldsymbol{w} \tag{2.59}$$

Employing zero-order Taylor expansion yields

$$\boldsymbol{f}^r\left(\tilde{\boldsymbol{x}}_t^a + \Delta_t\right) = \boldsymbol{f}^a\left(\tilde{\boldsymbol{x}}_t^a\right) + \boldsymbol{r}^a\left(\Delta_t\right) + \boldsymbol{f}^u\left(\tilde{\boldsymbol{x}}_t^a\right) + \boldsymbol{r}^u\left(\Delta_t\right) + \boldsymbol{w} \tag{2.60}$$

where $\boldsymbol{r}^a\left(\Delta_t\right)$ and $\boldsymbol{r}^u\left(\Delta_t\right)$ denote the residual terms beyond the zero-order expansion. Hence, the discrepancy between the real and approximate system can be formulated as

$$\begin{aligned}
\Delta\boldsymbol{x}_{t+1} &= \boldsymbol{f}^r\left(\tilde{\boldsymbol{x}}_t^a + \Delta_t\right) - \boldsymbol{f}^a\left(\tilde{\boldsymbol{x}}_t^a\right) && \text{(2.61a)}\\
&= \boldsymbol{f}^a\left(\tilde{\boldsymbol{x}}_t^a\right) + \boldsymbol{f}^u\left(\tilde{\boldsymbol{x}}_t^a\right) + \boldsymbol{r}\left(\Delta_t\right) + \boldsymbol{w} - \boldsymbol{f}^a\left(\tilde{\boldsymbol{x}}_t^a\right) && \text{(2.61b)}\\
&= \boldsymbol{f}^u\left(\tilde{\boldsymbol{x}}_t^a\right) + \boldsymbol{r}\left(\Delta_t\right) + \boldsymbol{w} = \hat{\boldsymbol{f}}^u\left(\tilde{\boldsymbol{x}}_t^a, \Delta_t\right) && \text{(2.61c)}
\end{aligned}$$

As can be seen in the equation, the next state is predicted from $\tilde{\boldsymbol{x}}_t^a$ and Δ_t. Thus, the Gaussian process is trained with precomputed inputs $\{\tilde{\boldsymbol{x}}_t^a, \Delta_t\}_{t=0}^{N-1}$ and training targets $\{\boldsymbol{y}_t = \Delta\boldsymbol{x}_{t+1} - \Delta\boldsymbol{x}_t\}_{t=0}^{N-1}$. The training procedure is repeated after the end of each rollout by taking the elementwise difference between the datasets $\mathcal{X}^a = \{\boldsymbol{x}_t^a, \boldsymbol{u}_t^a\}_{t=0}^N$ and $\mathcal{X}^r = \{\boldsymbol{x}_t^r, \boldsymbol{u}_t^r\}_{t=0}^N$. The remaining steps including learning the GP hypermarameters, approximate inference via moment matching, gradient based policy improvement and parameter update is performed as presented in the PILCO algorithm. While PILCO performes a long-term prediction with the Gaussian distribution $p\left(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t\right)$, PIREM applies the same steps based on the Gaussian distribution $p\left(\Delta\boldsymbol{x}_{t+1}|\tilde{\boldsymbol{x}}_t^a, \Delta_t\right) = \mathcal{N}\left(\Delta\boldsymbol{x}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}\right)$. Given training inputs $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1^a, \Delta_1, \ldots, \tilde{\boldsymbol{x}}_N^a, \Delta_N]$, training targets $\boldsymbol{y} = [y_1, \ldots, y_N]^{\mathrm{T}}$ and a new input $\tilde{\boldsymbol{x}}^* = \left[\left(\tilde{\boldsymbol{x}}_t^{a,*}\right)^{\mathbf{T}}\left(\Delta_t^*\right)^{\mathrm{T}}\right]^{\mathrm{T}}$, the predictive mean abd variance are computed as follows:

$$\begin{aligned}
\boldsymbol{\mu}_{t+1} &= \Delta\boldsymbol{x}_t + \boldsymbol{K}_{\tilde{x}^*\tilde{X}}\left(\boldsymbol{K}_{\tilde{X}\tilde{X}} + \sigma_n^2\boldsymbol{I}\right)^{-1}\boldsymbol{y}\\
\boldsymbol{\Sigma}_{t+1} &= k\left(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*\right) - \boldsymbol{K}_{\tilde{x}^*\tilde{X}}\left(\boldsymbol{K}_{\tilde{X}\tilde{X}} + \sigma_n^2\boldsymbol{I}\right)^{-1}\boldsymbol{K}_{\tilde{X}\tilde{x}^*}
\end{aligned} \tag{2.62}$$

where $\boldsymbol{K}_{\tilde{x}^*\tilde{X}} = k\left(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{X}}\right)$, $\boldsymbol{K}_{\tilde{X}\tilde{x}^*} = \boldsymbol{K}_{\tilde{x}^*\tilde{X}}^{\mathrm{T}}$ with $k(\cdot,\cdot)$ being the squared exponential covariance function

$$k\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j\right) = \sigma_k^2 \exp\left(-\frac{1}{2}\left(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j\right)^{\mathrm{T}}\boldsymbol{\Lambda}^{-1}\left(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j\right)\right) \tag{2.63}$$

Similar to PILCO, the hyperparameters $\boldsymbol{\Lambda}$, σ_k^2 and σ_n^2 are obtained via evidence maximization by using the training data. The parametrized policy is a function of input states $\mathbf{x}$ and parameters $\boldsymbol{\theta}$ and can be represented as sum of radial basis functions in the form

$$\pi(x, \theta) = \sum_i^D k\left(x^*, c_i\right)\left(K_{CC} + \sigma_\pi^2 I\right)^{-1} y_\pi \tag{2.64}$$

where $\boldsymbol{C} = [c_1, \ldots, c_D]$ denotes the center of the basis functions in 2.64. The parameters that are updated throughout the learning are $\boldsymbol{\theta} = [\boldsymbol{C}, \boldsymbol{\Lambda}, \boldsymbol{y}_\pi]$. The variance σ_π^2 is set to 0.01 based on previous emprical results. Just like in the PILCO algorithm, the cost function consitituting the total return is chosen as the saturating version:

$$c\left(\boldsymbol{x}_t^r\right) = 1 - \exp\left(-\frac{1}{2\sigma_c^2}\left\|\boldsymbol{x}_t^r - \boldsymbol{x}_g\right\|^2\right) \in [0, 1] \tag{2.65}$$

Minimization of the total cost requires the predictive distribution of the state, which can be mathematically formulated as $p\left(\Delta\boldsymbol{x}_1|\boldsymbol{\pi}\right), \ldots, p\left(\Delta\boldsymbol{x}_N, |\boldsymbol{\pi}\right)$. One-step prediction from previous state and input is computed as

$$p\left(\boldsymbol{y}_t\right) = \iint p\left(\hat{\boldsymbol{f}}^u\left(\boldsymbol{\xi}_t\right)|\boldsymbol{\xi}_t\right) p\left(\boldsymbol{\xi}_t\right) \mathrm{d}\hat{\boldsymbol{f}}^u \mathrm{d}\boldsymbol{\xi}_t \tag{2.66}$$

where $\boldsymbol{y}_t = \Delta\boldsymbol{x}_{t+1} - \Delta\boldsymbol{x}_t$, denoting the temporal difference of the residual state and $\xi_t = \left[\left(\tilde{\boldsymbol{x}}_t^a\right)^{\mathrm{T}}\left(\Delta_t\right)^{\mathrm{T}}\right]^{\mathrm{T}}$. Since this computation is intractable, the predictive distribution is computed via moment matching with the approximation of Gaussian distribution over $\boldsymbol{y}_{t-1}$.
Since the states of the new system is the residual difference between the real and approximate model, the cost function in 2.65 can be rewritten by defining a new goal state $\Delta\boldsymbol{x}_{g,t} = \boldsymbol{x}_g - \boldsymbol{x}_t^a$. As shown below, minimization of the new cost function drives the states of the real system to the target states by forcing the real robot to follow the same trajectory calculated by the approximate model:

$$\begin{aligned} c\left(\boldsymbol{x}_t^r\right) &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\left\|\boldsymbol{x}_t^r - \boldsymbol{x}_g\right\|^2\right) \\ &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\left\|\Delta\boldsymbol{x}_t + \boldsymbol{x}_t^a - \boldsymbol{x}_g\right\|^2\right) \\ &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\left\|\Delta\boldsymbol{x}_t - \left(\boldsymbol{x}_g - \boldsymbol{x}_t^a\right)\right\|^2\right) \\ &= 1 - \exp\left(-\frac{1}{2\sigma_c^2}\left\|\Delta\boldsymbol{x}_t - \Delta\boldsymbol{x}_{g,t}\right\|^2\right) = c\left(\Delta\boldsymbol{x}_t\right) \end{aligned} \tag{2.67}$$

2.5 Iterative Linear Quadratic Regulator

Trajectory optimization is the process of finding a series of states and controls to locally minimize a given cost function. While direct methods explicitly represent the states, shooting methods just parametrize the controls. The states are computed with forwards integration and the are not directly present in the optimization state, which is the reason why shooting methods are also named as indirect method [VSB92]. System dynamics are contained in the optimization, obtained tranjectories are strictly feasable. Differential dynamic programming (DDP) [MAY66] is

Algorithm 2 Policy Improvement with REsidual Model learning

procedure PI-REM
init: Learn a policy $\boldsymbol{\pi^a}$ for the approximate model
Set $\boldsymbol{\pi^n} = \boldsymbol{\pi^a}$
while task learning is not achieved **do**
  Apply $\boldsymbol{\pi^n}$ to the real robot and record data
  Compute the training set $\mathcal{X}^d = \mathcal{X}^r - \mathcal{X}^a$
  Compute the corresponding target set $\Delta x_{g,t} = x_g - x_t^a$
  Update the GP hyperparameters via evidence maximization
  repeat
    Approximate inference for policy evaluation, get $J^\pi(\boldsymbol{\theta})$
    Gradient-based policy improvement by getting $\mathrm{d}J^\pi(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$
    Update parameters θ
  until convergence; **return** θ^*
  Set $\pi^n \leftarrow \pi(\boldsymbol{\theta}^*)$
end procedure

a second-order shooting method allowing quadratic convergence for systems with smooth dynamics. In case second-order derivatives of the dynamics are cancelled and only the first-order terms remain, the algorithm is called iterative Linear-Quadratic Regulator (iLQR) [TMT14].

Considering discrete-time dynamics, a system can be modeled as:

$$\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i), \tag{2.68}$$

where i denotes the time index of the system with states $\mathbf{x} \in \mathbb{R}^n$ and controls $\mathbf{u} \in \mathbb{R}^m$. A trajectory $\{\mathbf{X}, \mathbf{U}\}$ represents a sequence of states $\mathbf{X} \triangleq \{\mathbf{x}_0, \mathbf{x}_1 \ldots, \mathbf{x}_N\}$ with controls $\mathbf{U} \triangleq \{\mathbf{u}_0, \mathbf{u}_1 \ldots, \mathbf{u}_{N-1}\}$.

The total cost is represented by J as a sum of running costs ℓ and final cost ℓ_f between index 0 and N:

$$J(\mathrm{x}_0, \mathrm{U}) = \sum_{i=0}^{N-1} \ell(\mathrm{x}_i, \mathbf{u}_i) + \ell_f(\mathrm{x}_N) \tag{2.69}$$

As mentioned at the beginning, shooting methods rely only on the sequence of controls $\mathbf{U}$ to rollout the state trajectory from the initial state x_0. Thus, the objective is to find the optimum control sequence minimizing the total cost:

$$\mathbf{U}^\star \triangleq \underset{\mathbf{U}}{\operatorname{argmin}} J(\mathbf{x}_0, \mathbf{U}) \tag{2.70}$$

The *cost-to-go* J_i denotes the sum of costs from time index i to N:

$$J_i\left(\mathrm{x}, \mathrm{U}_i\right) = \sum_{j=i}^{N-1} \ell\left(\mathrm{x}_j, \mathrm{u}_j\right) + \ell_f\left(\mathrm{x}_N\right) \tag{2.71}$$

The *Value* at time i stands for the minimum cost-to-go receieved by applying the optimal control sequence:

$$V_i(\mathbf{x}) \triangleq \min_{\mathbf{U}_i} J_i\left(\mathbf{x}, \mathbf{U}_i\right) \tag{2.72}$$

The value at the final time step is equal to running cost ℓ_f. Expending the equation above yields:

$$V(\mathrm{x}) = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + V'(\mathbf{f}(\mathbf{x}, \mathbf{u}))\right] \tag{2.73}$$

where V' denotes the value at the next time step.

At this point, we will define a term $Q(\delta\mathbf{x}, \delta\mathbf{u})$ representing the change in the value due to small perturbations $\delta\mathbf{x}$ and $\delta\mathbf{u}$ in the state-control pair:

$$Q(\delta\mathbf{x}, \delta\mathbf{u}) = \ell(\mathbf{x}+\delta\mathbf{x}, \mathbf{u}+\delta\mathbf{u}) - \ell(\mathbf{x}, \mathbf{u}) + V'(\mathbf{f}(\mathbf{x}+\delta\mathbf{x}, \mathbf{u}+\delta\mathbf{u})) - V'(\mathbf{f}(\mathbf{x}, \mathbf{u})) \tag{2.74}$$

The purpose of DDP is to find the smallest perturbation in the control signal such that the change in the value function is minimized alongside the trajectory. This would guarantee the convergence to the optimal projection and prevent deviations from it. Expanding $Q(\delta\mathbf{x}, \delta\mathbf{u})$ usingTaylor's second order expansion gives:

$$Q(\partial\mathbf{x}, \partial\mathbf{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \partial\mathbf{x} \\ \partial\mathbf{u} \end{bmatrix}^T \begin{bmatrix} 0 & \mathbf{Q}_\mathbf{x}^T & \mathbf{Q}_\mathbf{u}^T \\ \mathbf{Q}_\mathbf{x} & \mathbf{Q}_\mathbf{xx} & \mathbf{Q}_\mathbf{xu} \\ \mathbf{Q}_\mathbf{u} & \mathbf{Q}_\mathbf{ux} & \mathbf{Q}_\mathbf{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \partial\mathbf{x} \\ \partial\mathbf{u} \end{bmatrix} \tag{2.75}$$

Combining last two terms yields:

$$Q_\mathrm{x} = \ell_\mathrm{x} + \mathrm{f}_\mathrm{x}^\top V'_\mathrm{x} \tag{2.76a}$$
$$Q_\mathrm{u} = \ell_\mathrm{u} + \mathrm{f}_\mathrm{u}^\top V'_\mathrm{x} \tag{2.76b}$$
$$Q_\mathrm{xx} = \ell_\mathrm{xx} + \mathrm{f}_\mathrm{x}^\top V'_\mathrm{xx} \mathrm{f}_\mathrm{x} + V'_\mathrm{x} \cdot \mathrm{f}_\mathrm{xx} \tag{2.76c}$$
$$Q_\mathrm{ux} = \ell_\mathrm{ux} + \mathrm{f}_\mathrm{u}^\top V'_\mathrm{xx} \mathrm{f}_\mathrm{x} + V'_\mathrm{x} \cdot \mathrm{f}_\mathrm{ux} \tag{2.76d}$$
$$Q_\mathrm{uu} = \ell_\mathrm{uu} + \mathrm{f}_\mathrm{u}^\top V'_\mathrm{xX} \mathrm{f}_\mathrm{u} + V'_\mathrm{x} \cdot \mathrm{f}_\mathrm{uu} \tag{2.76e}$$

where the last terms require the product of a tensor with a vector, which are neglected in iLQR. Hence, the rest of the terms can be computed by using raw data where the derivatives can be computed by means of finite-difference method. Minimizing $Q(\delta\mathbf{x}, \delta\mathbf{u})$ with respect to $\delta\mathbf{u}$ provides an open-loop term $\mathbf{k}$ and a closed-loop term $\mathbf{K}$:

$$\delta\mathbf{u}^\star(\delta\mathbf{x}) = \underset{\delta\mathbf{u}}{\operatorname{argmin}} Q(\delta\mathbf{x}, \delta\mathbf{u}) = \mathbf{k} + \mathbf{K}\delta\mathbf{x} \tag{2.77}$$

where

$$\mathbf{k} \triangleq -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} \tag{2.78a}$$

$$\mathbf{K} \triangleq -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}} \tag{2.78b}$$

Plugging these terms back to the second order expansion 2.75 gives

$$\Delta V = -\frac{1}{2} \mathbf{k}^{\mathrm{T}} Q_{\mathrm{uu}} \mathbf{k} \tag{2.79a}$$

$$V_{\mathrm{x}} = Q_{\mathrm{x}} - \mathbf{K}^{\mathrm{T}} Q_{\mathrm{uu}} \mathbf{k} \tag{2.79b}$$

$$V_{\mathrm{xx}} = Q_{\mathrm{xx}} - \mathbf{K}^{\top} Q_{\mathrm{uu}} \mathbf{K} \tag{2.79c}$$

The terms in 2.78 and 2.79 are updated recursively backwards in time by initializing the value function as the terminal cost. After the *backward pass* is completed, the locally linear policy $\delta\mathbf{u}$ is evaluated by computing the state trajectory:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 \tag{2.80a}$$

$$\hat{\mathbf{u}}_i = \mathbf{u}_i + \alpha \mathbf{k}_i + \mathbf{K}_i \left(\hat{\mathbf{x}}_i - \mathbf{x}_i \right) \tag{2.80b}$$

$$\hat{\mathbf{x}}_{i+1} = \mathbf{f} \left(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i \right) \tag{2.80c}$$

The backward and forward passes are repeated consecutively until convergence to a locally optimal trajectory is reached. The term α is a parameter called backtracking search parameter and prevents possible divergence of the the new trajectory by retaining it inside the region of validity. It is determined by picking the optimal value which generates the least total cost along the trajectory among a set of gradually decreasing candidates between 0 and 1. In other words, the cumulative cost is evaluated with various numbers of α within the range between 0 and 1 that is kept constant during forward pass. Finally, the one with minimum total cost is chosen. Backtracking introduces additional cost and can be discarded for some real time applications.

In order to ensure a descent direction, Hessian matrices should be positive definite. In case the Hessian becomes negative semi-definite, Tikhonov regularization terms are added to the diagonals of before the inversion of $Q_{\mathbf{uu}}$ in 2.78. The regularization terms are determined through Levenberg-Marquardt method which is detailed in [TL05].

Due to physical constraints of the system such as limited joint torques, joint ranges and further kinematical features, additional inequality constraints should be taken into account for solving the optimization problem. While inequality constraints on the states can be implicitly managed by smooothing hard constraints [TT10], box constraints on control signals require a different approach. A straighforward method could be naive clamping of the control signal. However, clamped signals may cause increased cost, hence divergence. Clmaping can also performed in the computation of open-loop signal $\mathbf{k}$ during the forward-pass phase or by setting the corresponding rows of $\mathbf{K}$ equal to zero to to inactivate the feedback. These methods have yet

experimentally shown to be inefficient [TMT14].
A further solution could be to input the control to a squahing function $s(\mathbf{u})$. To prevent the control signal reaching large or small values and thus being stuck on the plateau, the cost function should consider the original control $\mathbf{u}$ rather than the clamped $s(\mathbf{u})$. The drawback of this approach lies in the nonlinearity of the sigmoid function: Due to local quadratic approximation in the backward pass, higher order terms produce adverse impact on convergence.
The autors of [TMT14] propose a new algorithm called projected Newton quadratic programming algortihm to compute the optimal feedback gain $\mathbf{k}$ as the result of the optimization task:

$$\begin{aligned} \mathrm{k} = \underset{\delta\mathbf{u}}{\operatorname{argmin}} & \frac{1}{2}\delta\mathbf{u}^\top Q_{\mathbf{uu}}\delta\mathbf{u} + Q_{\mathbf{u}}^\top\delta\mathbf{u} \\ \text{subject to } & \underline{\mathbf{b}} \leqslant \mathbf{u} + \delta\mathbf{u} \leqslant \overline{\mathbf{b}} \end{aligned} \tag{2.81}$$

where $\overline{\mathbf{b}}$ and $\underline{\mathbf{b}}$ denote the upper and lower limits of the control signal. The free dimenisons of the matrix $Q_{\mathbf{uu}}$ obtained from this solution is used to evaluate the feedback term $\mathbf{K}_\mathrm{f} = -Q_{\mathbf{uu},\mathrm{f}}Q_{\mathbf{ux}}$. The remaining rows of $\mathbf{K}$ are set to zero whose indices correspond to clamped controls. The above problem can be generically reformulated as follows:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\top\mathbf{H}\mathbf{x} + \mathbf{q}^\top\mathbf{x} \\ \text{subject to} \quad & \underline{\mathbf{b}} \leqslant \mathbf{x} \leqslant \overline{\mathbf{b}} \end{aligned} \tag{2.82}$$

Defining an initial value $\mathbf{x} = [\![\mathbf{x}]\!]_\mathrm{b}$ and gradient $\mathbf{g} = \nabla_\mathbf{x} f = \mathbf{q} + \mathbf{H}\mathbf{x}$, the set of free and clamped indices can be formulated as

$$c(\mathbf{x}) = \left\{ j \in 1 \dots n \;\middle|\; \begin{array}{ll} \mathbf{x}_j = \mathbf{b}_j, & \mathbf{g}_j > 0 \\ \mathbf{x}_j = \overline{\mathbf{b}}_j, & \mathbf{g}_j < 0 \end{array} \right\} \tag{2.83a}$$

$$f(\mathbf{x}) = \{ j \in 1 \dots n | j \notin c \} \tag{2.83b}$$

The vectors $\mathbf{x}$, $\mathbf{g}$ and matrix $\mathbf{H}$ can be rearranged according to their free and clamped indices:

$$\mathbf{x} \leftarrow \begin{bmatrix} \mathbf{x}_\mathrm{f} \\ \mathbf{x}_\mathrm{C} \end{bmatrix}, \mathbf{q} \leftarrow \begin{bmatrix} \mathbf{q_f} \\ \mathbf{q_c} \end{bmatrix}, \mathbf{H} \leftarrow \begin{bmatrix} \mathbf{H}_\mathrm{ff} & \mathbf{H}_\mathrm{fc} \\ \mathbf{H}_\mathrm{cf} & \mathbf{H}_\mathrm{cc} \end{bmatrix} \tag{2.84}$$

Thus, the gradient for free indices is

$$\mathbf{g}_\mathrm{f} = \nabla_{\mathrm{x_f}} f = \mathbf{q}_\mathrm{f} + \mathbf{H}_\mathrm{ff}\mathbf{X}_\mathrm{f} + \mathbf{H}_\mathrm{fc}\mathbf{x}_\mathrm{c} \tag{2.85}$$

The step size for the free indices is

$$\Delta\mathbf{x}_\mathrm{f} = -\mathbf{H}_\mathrm{ff}^{-1}\mathbf{g}_\mathrm{f} = -\mathbf{H}_\mathrm{ff}^{-1}\left(\mathbf{q}_\mathrm{f} + \mathbf{H}_\mathrm{fc}\mathbf{x}_\mathrm{c}\right) - \mathbf{x}_\mathrm{f} \tag{2.86}$$

Appending zero vector at the clamped indices gives

$$\Delta \mathrm{x} = \begin{bmatrix} \Delta \mathbf{x}_{\mathrm{f}} \\ \mathbf{0}_{\mathrm{c}} \end{bmatrix} \tag{2.87}$$

Thus, the updated $\hat{\mathbf{x}}$ subject to backtracking line-search is equal to

$$\hat{\mathbf{x}}(\alpha) = [\![\mathbf{x} + \alpha \Delta \mathbf{x}]\!]_{\mathbf{b}} \tag{2.88}$$

The line-search parameter α is decreased until the Armijo condition is met

$$\frac{f(\mathbf{x}) - f(\hat{\mathbf{x}}(\alpha))}{\mathbf{g}^{\top}(\mathbf{x} - \hat{\mathbf{x}}(\alpha))} > \gamma \tag{2.89}$$

where γ is commonly set to 0.1 based on empirical results in the literature.

2.6 PILCO-iLQR and PIREM-iLQR

The goal of this thesis is to incorporate iterative Linear Quadratic Regulator into policy optimization stage of the existing PILCO and PIREM algorithms, which rely on external optimizers such as CG or L-BFGS to update the policy parameters based on the gradient information of the return function. By retaining the approximate inference via moment-matching mechanism to predict the long term state trajectory, our approach updates the policy with a series of recursive backward and forward sweep steps, identical to the regular policy improvement procedure with deterministic functions. Through a series of interaction with the real environment, the hyperparameters of the Gausian Process are again updated by means of expectation maximization method at the end of each rollout. The only major distinction of our approach regarding the use of Gaussian Process (GP) is the assumption of zero covariance between control signals and states. The covariance information is used to compute joint state-control distribution and predict the change in the state. While analytic computation of these covariance terms is possible for PILCO thanks to the parametrized policy function that accepts the current state as input, we simply set them equal to zero at the prediciton stage.

PILCO-iLQR is in fact nothing but a usual iLQR method which adapts Gaussian Process regression as system transition function and attempts to obtain local linear approximation of the dynamics via finite difference method instead of explicit computation of first order derivatives. Finite difference method is a numerical approach to approximate the function derivative at a certain point which observes the change in the function value after a small amount of increase ϵ in the input:

$$u'(x) = \frac{u(x + \epsilon) - u(x)}{\epsilon} \tag{2.90}$$

One of the major findings of our work is that the uncertainity information about the current state has to be included into local approximation of cost and dynamics

functions by augmenting the state vector comprised of mean values with the elements of covariance matrix. In other words, a state vector with n dimensions used in conventional iLQR method needs to be augmented with n^2 additional covariance terms obtained from the predictive distribution of Gaussian process. Experiments on simple pendulum system demonstrate that both PILCO-iLQR and PIREM-iLQR algorithms fail to converge to an optimal policy if covariance terms are neglected in cost and dynamics functions.

The inclusion of covariance terms in the state vector brings about additional computational cost during evaluation of the derivative terms in the backward pass. In addition, the parameters weighting the cost of covariance terms need to be carefully selected and tuned. In order to achieve optimal policy, the covariance terms should not only be considered in the derivative of the dynamics function, but they should also be contained in the cost function. The target values of all variance terms are set to zero to obtain stable and certain trajectories. While setting too large weights on the variance terms may lead to a suboptimal policy and prevent the mean values from reaching their own target values, small weight degrades the convergence behaviour of the policy.

The incorporation of iLQR into PIREM is implemented in the similar way as PILCO-iLQR. However unlike PILCO-iLQR, iLQR method also provides an initial policy for the real system, which is computed based on the approximate model before the first rollout. Introduction of the approximate policy significantly reduces the number of required rollouts by providing closed-loop feedback terms and plausible initial trajectory if the approxiamte model is accurate enough. Similar to the plain PIREM, the new target set for the residual dynamics is recomputed before the policy improvement stage as explained in 2. After the trajectory predicted with GP regression successfully converges, the control signals generated with the policy proposed by PILCO-iLQR are superposed with the control signals generated by the approximate model during real rollouts.

The state-control pairs $\tilde{\boldsymbol{x}}_t^a = (\boldsymbol{x}_t^a, \boldsymbol{u}_t^a)$ that predict the next state together with the residual terms are kept unchanged in feedforward phase, because they are received from the approximate model and independent of the policy applied on the real robot. Since they remain constant during the trajectory optimization, derivate operations to compute the corresponding feedforward and feedback gains for these terms are discarded. In order to show that it is not only the closed-loop feedback terms of the approximate model that enable the agent to reach the target state, we distinguish between two versions of PIREM-iLQR: The *open-loop* version just utilizes the feedforward control signals genereated by the approximate model, whereas the *closed-loop* PIREM-iLQR uses these closed-loop terms from the approximate model as well.

Algorithm 3 Probabilistic Inference for Learning Control with Iterative Linear Quadratic Regulator

procedure PILCO-ILQR
init: Apply random control signals and record data
repeat
  Learn GP hyperparameters via evidence maximization
  repeat
    Local approximation: Obtain the first order derivatives of belief dynamics $f_{\mathbf{x}}$, $f_{\mathbf{u}}$ and quadratic approximation of the cost function $\ell_{\mathbf{x}}$,$\ell_{\mathbf{u}}$,$\ell_{\mathbf{xx}}$,$\ell_{\mathbf{uu}}$,$\ell_{\mathbf{xu}}$ via finite-difference method
    Backward pass: Recursively compute the value functions and obtain feed-forward and feedback terms
    Forward pass: Evaluate the locally linear policy forwards in time
  until trajectory convergence; **return** $\mathbf{k}_i$ and $\mathbf{K}_i$
  Apply the locally-linear policy to the real system and record data
until task learned
end procedure

Algorithm 4 Iterative Linear Quadratic Regulator with REsidual Model learning

procedure PIREM-ILQR
init: Learn the policy $\boldsymbol{\pi^a}$ for the approximate model through iLQR
Set $\boldsymbol{\pi^n} = \boldsymbol{\pi^a}$
while task learning is not achieved **do**
  Apply $\boldsymbol{\pi^n}$ to the real robot and record data
  Compute the training set $\mathcal{X}^d = \mathcal{X}^r - \mathcal{X}^a$
  Compute the corresponding target set $\Delta x_{g,t} = x_g - x_t^a$
  Update the GP hyperparameters via evidence maximization
  repeat
    Local approximation: Obtain the first order derivatives of belief dynamics $f_{\Delta\mathbf{x}}$, $f_{\Delta\mathbf{u}}$ and quadratic approximation of the cost function $\ell_{\Delta\mathbf{x}}$, $\ell_{\Delta\mathbf{u}}$, $\ell_{\Delta\mathbf{x}\Delta\mathbf{x}}$, $\ell_{\Delta\mathbf{u}\Delta\mathbf{u}}$, $\ell_{\Delta\mathbf{x}\Delta\mathbf{u}}$ via finite-difference method
    Backward pass: Recursively compute the value functions and obtain feed-forward and feedback terms
    Forward pass: Evaluate the locally linear policy forwards in time
  until trajectory convergence; **return** $\mathbf{k}_i^r$ and $\mathbf{K}_i^r$
  Set $\pi^n \leftarrow$ superposition of
end procedure

Chapter 3

Experiments and Results

In order to compare the validity of aforementioned algorithms,we conducted experiments on various tasks with gradually increasing difficulty levels. The first task is balancing of the inverted pendulum, which is implemented in MATLAB. Secondly, we simulated a ball-on-plate scenario where a deformable object is requested to be brought to the center of a plate by changing the inclination angle of the plate. The simulations are implemented in Bullet physics engine which offers an environment to test rigid and soft body interactions. Finally, the same ball-and-plate task is implemented with a real KUKA LWR arm actuator.

3.1 Inverted Pendulum

The objective of the inverted pendulum task is to bring the pendulum from stable equilibrium point to the upright position. The system accepts applied torque as input and the state vector of the pendulum is defined as $\boldsymbol{x} = [\dot{\theta}, \theta]^{\mathrm{T}}$ where θ denotes the angle between the pendulum and vertical position as illustrated in 3.1. The dynamics governing the system behaviour can be expressed as follows:

$$\ddot{\theta}_t \left(\frac{1}{4} m l^2 + I \right) + \frac{1}{2} m l g \sin(\theta_t) = u_t - b\dot{\theta}_t \tag{3.1}$$

where $l = 1/12ml^2$ represents the moment of inertial of the pendulum around the midpoint, b denotes the friction coefficient and u is the torque signal. For PIREM and PIREM-iLQR algorithms, the real model differs from the approximate one by the friction term which is set to 0.1 for the approximate model and 0.7 for the real system. In order to ensure stability, the sampling rate is chosen to be 0.1 seconds. The horizon for algorithms is specified as 40 steps, which is equivalent to 4 seconds.

The simulation results are given in 3.1 and 3.2. While all of the algorithms are capable of finding an optimal policy to keep the pendulum at the target state $[0, -\pi]^{\mathrm{T}}$, PIREM-iLQR outperforms the rest of the algorithms in terns of required duration

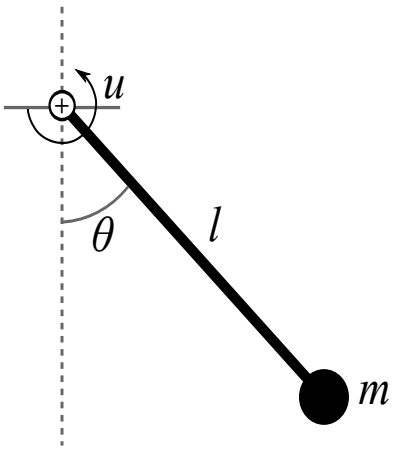


Figure 3.1: Inverted pendulum setup with torque as control signal

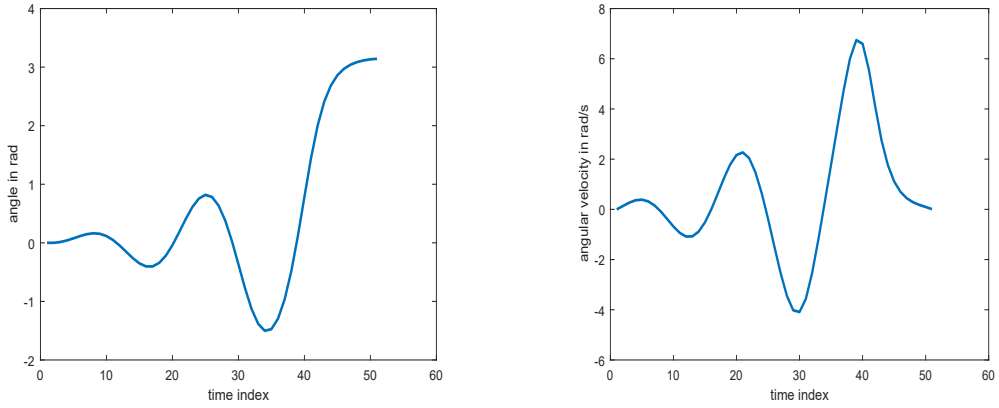
Algorithm	Real Rollouts (#)	Time for Policy Search (s)
PILCO	3	67
PIREM	3	118
PILCO-iLQR	3	62
PIREM-iLQR (open-loop)	3	52
PIREM-iLQR (closed-loop)	**2**	**20**

Table 3.1: Number of real rollouts and total time required to perform policy search with respect to chosen algorithm.

of real experience and average time spend to optimize the policy. Feedback terms proposed by the approximate model and the initial trajectory obtained by the initial policy significantly improves the convergence behaviour of the policy. In addition, it can said that the incorporation of iLQR into PIREM algorithm reduces the time elapsed during policy optimization by a considerable amount (almost by a factor of 4), whereas we cannot observe the same improvement for the PILCO algorithm. While the required time for policy search amounts 67 seconds for PILCO, it takes 62 seconds for the same amount of rollouts with PILCO-iLQR.

Algorithm	θ_f (deg)	$\dot{\theta}_f$ (deg/sec)
PILCO	1.4	180.1
PIREM	1.3	179.6
PILCO-iLQR	**1.4**	**180.2**
PIREM-iLQR (open-loop)	2.4	179.5
PIREM-iLQR (closed-loop)	1.6	179.8

Table 3.2: Final angle and angular velocity values.



(a) Angular Position (b) Angular Velocity

Figure 3.2: Angular position and velocity trajectory obtained with PIREM-iLQR after 2 rollouts.

3.2 Ball and Plate

The state space of the approximate model for the ball-and-plate scenario consists of the position ($\mathbf{x}$,$\mathbf{y}$), velocity ($\dot{\mathbf{x}}$,$\dot{\mathbf{y}}$) and inclination angle ($\theta_\mathbf{x}$,$\theta_\mathbf{y}$) of each dimension. The input of the system is chosen as the amount of change in the inclination angles ($\mathbf{\Delta}\theta_\mathbf{x}$,$\mathbf{\Delta}\theta_\mathbf{y}$). Setting the change in the angle as input allows us to enforce an upper limit on the rate of angle variation during the optimization process, which prevents unrealistic abrupt jumps in the angle values during real rollouts. Considering the limitations of the robot arm such as torque limits and kinematical structure during manipulation, maximum rate of angle change is specified as 6 degrees/second for both the Bullet simulations and real robot experiments.

The approximate model for PIREM and PIREM-iLQR algorithms in Bullet simulations relies on the assumption that a perfectly rigid ball starts rolling without any slipping behaviour as soon as the plate is tilted with an infinitesimal angle. Within this framework, angular acceleration α gained through the torque τ applied by the friction force F_f is proportional to the moment of inertia I of the rolling sphere, which is given as $2/5mr^2$ with r denoting the radius of the solid sphere and m representing the mass. Mathematically, this can be formulated as

$$\tau = I\alpha = I\frac{a}{r} = \frac{2}{5}mr^2\frac{a}{r} \tag{3.2}$$

where a denotes the linear acceleration along the inclined surface. The torque defined above is equivalent to the product of friction force F_f and the radius of the sphere.

Therfore, we can conclude the following relations:

$$\begin{aligned} F_f r &= \frac{2}{5} mra \\ mg\cos(\theta)\,\mu &= \frac{2}{5} mra \\ \mu &= \frac{2a}{5g\cos(\theta)} \end{aligned} \tag{3.3}$$

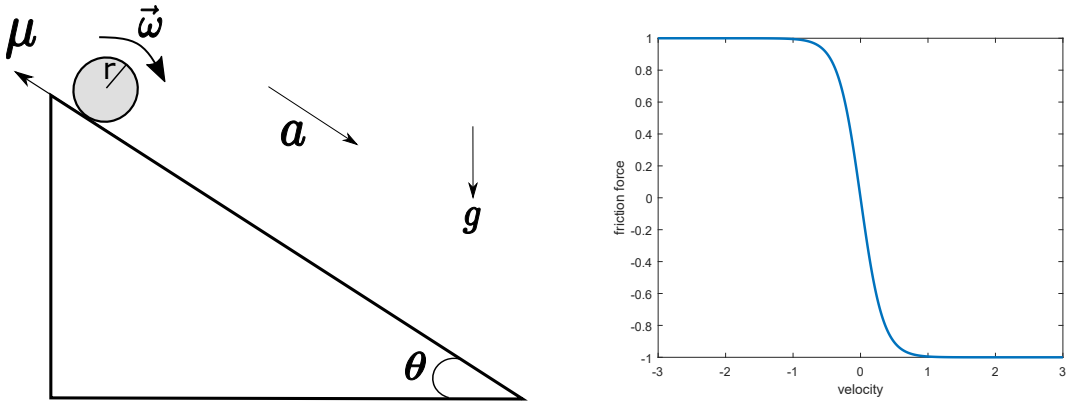
The linear acceleration produced by the net force exerted on the sphere along the plane surface can be computed as

$$g\sin(\theta) - g\cos(\theta) = a \tag{3.4}$$

Setting both a terms in the last two equations equal to each yields the following result:

$$a = \frac{5}{7} g\sin(\theta) \tag{3.5}$$

While the assumption of immediate movement under very small inclinations was accurate enough for the PIREM to learn the residual dynamics in Bullet environment, the minimum angle required to trigger motion was approximately 3 degrees for the real setup. Therefore we improved our approximate policy by modelling the friction force with a differentiable hyperbolic tangent as illustrated in Fig. 3.3. The asymptotes of the hyperbolic tangent function are computed based on the treshold angle required for rolling.



(a) Rolling motion without spinning

(b) Friction force

Figure 3.3: Overview of rolling without spinning dynamics adapted to build the approximate model in Bullet simulations (left). Friction force with respect to the ball velocity is represented as hyperbolic tangent function for experiments with real robot.

3.2.1 Simulation Results

Using Bullet physics environment, we conducted experiments for two and one dimensional cases. The limit of control signal is set to 6 degrees/second consistent with the real scenario. In order to attain the best performence with least computational complexity, the sampling frequency is defined as 40 Hz, which is the minimum frequency Bullet simulator allows. The horizon comprises of 100 time steps which correspends to 2.5 seconds with the current samplig frequency. The initial distance of the ball to the center of plate is set to 60 and 70 cm in x and y-axis, respectively. The radius of the ball is set to 10 cm.

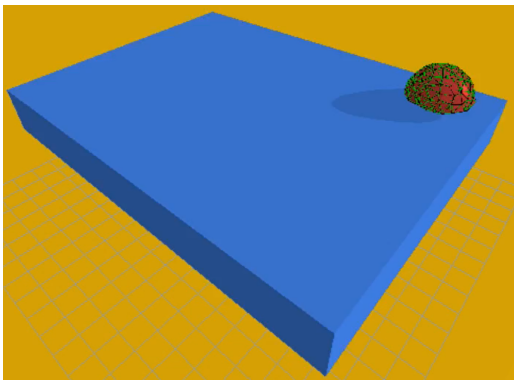


Figure 3.4: Experiment setup built with Bullet physics simulator.

Experiment results obtained from the ball-and-plate simulations share similarities with the results from the previous pendulum experiments: All the algorithms except PIREM-iLQR exhibit similar performance in terms of the number of performed rollouts. To drive the ball to the center of plate in two-axis at the same time, 10 rollouts have to be performed by these algorithms, while the closed-loop feedback terms obtained from the approximate model allowed PIREM-iLQR to achieve the task in two rollouts. A further similarity to the previous experiment is the contribution of iLQR in the polict improvement phase: The time elapsed to perform policy search is reduced from 320 minutes to 180 minutes by combining iLQR with the PIREM, whereas PILCO algorithm got even slower with the introduction of iLQR for both single and two-dimensional experiments. Analogous to the two dimensional setting, almost all the remaining algorithms except for the closed-loop PIREM-iLQR require 4 rollouts. The computational complexity of PILCO grows adversely again: PILCO involves a total computation time of 11 minutes to perform policy optimization, whereas PILCO-iLQR needs 21 minutes to perform backward and forwards sweeps to achieve the same task.

Algorithm	Real Rollouts (#)	Time for Policy Search (min)
PILCO	10	128
PIREM	10	320
PILCO-iLQR	10	168
PIREM-iLQR (open-loop)	10	181
PIREM-iLQR (closed-loop)	**2**	**10**

Table 3.3: Number of real rollouts and total time required to perform policy search for two dimensional case.

Algorithm	x-velocity (cm/s)	x-position (cm)	y-velocity (cm/s)	y-postion c(m)
PILCO	-1.3	3.2	-2.2	-0.6
PIREM	-0.2	0.84	-0.5	0.9
PILCO-iLQR	1.6	1.2	-0.1	1.5
PIREM-iLQR (open-loop)	0.45	1	0.02	1.4
PIREM-iLQR (closed-loop)	**-0.03**	**0.76**	**-0.26**	**0.51**

Table 3.4: Final position and velocity values for two-dimensional case.

Algorithm	Real Rollouts (#)	Time for Policy Search (min)
PILCO	4	11
PIREM	4	45
PILCO-iLQR	4	21
PIREM-iLQR (open-loop)	5	26
PIREM-iLQR (closed-loop)	**2**	**3**

Table 3.5: Number of real rollouts and total time required to perform policy search for one dimensional case.

Algorithm	velocity (cm/s)	position (cm)
PILCO	0	-1.28
PIREM	0.3	0.5
PILCO-iLQR	0.8	1
PIREM-iLQR (open-loop)	0.1	-1
PIREM-iLQR (closed-loop)	**0.02**	**0.4**

Table 3.6: Number of real rollouts and total time required to perform policy search for one dimensional case.

3.2.2 Experiments with the real robot

The real physical setup consists of a KUKA LWR robot arm controlled with PIREM-iLQR or PILCO-iLQR algorithm and a rectangular plate fenced with a cartoon barrier which is attached to the end effector as illustrated in Fig.3.5. The position of the ball is online tracked with a Asus Xtion depth sensor and the velocity is obtained by taking the difference beween the current and previous position values.

(a) Top view of the plate

(b) Side view of the KUKA LWR robot arm

Figure 3.5: Ball and plate system

As the ball hits the barrier, we assume that it keeps rolling down with an increasing velocity obtained via linear extrapolation based on the last velocity values before the collision. The extrapolation is performed offline after the termination of the real rollout. Positions values are corrected based on the extrapolated velocity values. In order to test the effecitivity of PILCO-iLQR and PIREM-iLQR algorithms, three kinds of rolling objects with different materials are used: a balloon filled with rice, a bouncy ball filled with liquid and a rigid sponge ball as shown in 3.6.

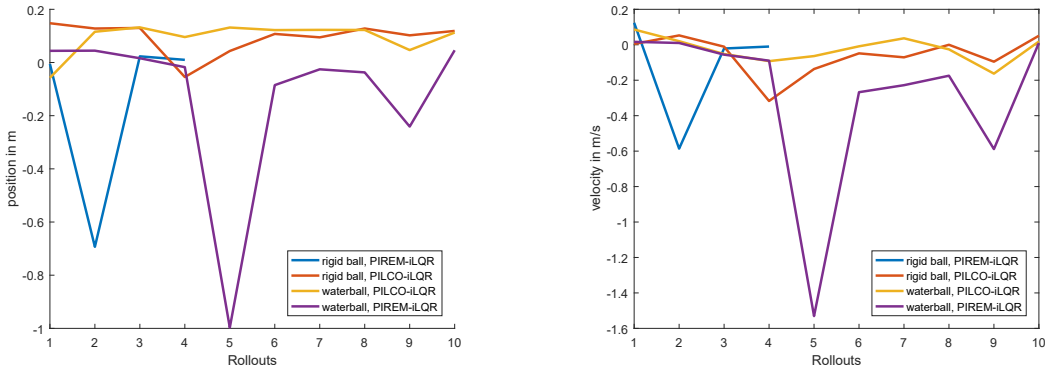
(a) Sponge ball

(b) Bouncy ball filled with water

(c) Balloon filled with rice

Figure 3.6: Kinds of balls used in real experiments

The position and velocity values at the final time step obtained by applying PIREM-iLQR and PILCO-iLQR on the bouncy water-filled and rigid sponge balls are shown in Fig. 3.7. Within 10 iterations, the only successful convergence to the local policy is observed with the rigid sponge ball after performing 4 rollouts with the closed-loop PIREM-iLQR. Due to the reasons that will be explained in the following section, the total time required for the policy search in 10 iterations takes in 8.5 hours in average. As more data is accumulated, computational complexity per rollout increases exponentially such that the eleventh iteration alone requires 2 hours of computation. Due to limited time remaining to conduct real robot experiments, we terminated the experiments after 10 rollouts.



(a) Position (b) Velocity

Figure 3.7: Results obtained by performing PILCO-iLQR and PIREM-iLQR algorithms on rigid and water filled ball with the real robot. (a) Final position over iteration index. (b) Final velocity over iteration index.

The evolution of the final position over the rollout index can be seen in 3.8 for the rigid ball controlled with closed-loop PIREM-iLQR. The policy fully converges after the fourth rollout and the standard deviation in the final position gradually decreases as the Gaussian Process is refined with the obtained data.

3.2.3 Discussion on the experiments with the physical robot

In this section, we will discuss the possible factors behind the (partial) failure of the experiments with the real robot. However, we would like to remind the reader that the real experiments had to be terminated at the elevent rollout due to enormous computational complexity at the policy search stage, which reaches 2 hours of computation time at the eleventh rollout alone.

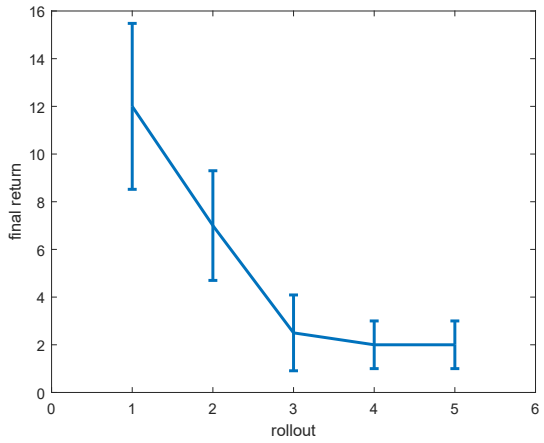


Figure 3.8: Evolution of the final distance to the center of plate along the x-axis (mean and standard deviation over 7 executions).

The reason for the drastic growth in the time required to perform policy search is the prediction cost of order $\mathcal{O}(N^2)$ where N represent the number of data points used for Gaussian process regression. While the sparse GP model used in simulations scales down the order of complexity by introducing pseudo inputs summarizing the existing data, non-smooth fluctutations in the real data inhibit the convergence of this algorithm and force us to use the whole data at the expense of time. We also attempted to train the GP with a smaller portion of data received from the recent rollouts which failed to capture the system dynamics properly.

A further element impeding the success of tested algorithms is the delay in the robot's controller. Fig 3.9 illustrates the delay factor in the commanded angles which adversely affect the efficiency of algorithms in the presence non-smooth feed-forward control signals. In fact, control signals proposed by PILCO-iLQR method are much more noisy than those of PIREM-iLQR as shown in Fig. 3.10 , which might be one of the key reasons for the failure of PILCO-iLQR with the rigid ball.

As mentioned earlier, we employ linear extrapolation to estimate the state values at the future time steps in case of a collision beween the ball and barrier around the plate as shown in 3.11. While the final distance to the plate center usually reaches values in the order of meters, a small change in balls behaviour at the initial time steps would inevitably lead to huge deviations in the estimated trajectories. Since total return function is dominated by the final states, extrapolation may bring about misguided policy updates. Another important side effect of this phenomenon is the generation of inaccurate training data for the Gaussian process regression.

The next factor that impinges the effectiveness of the tested algorithms is the in-

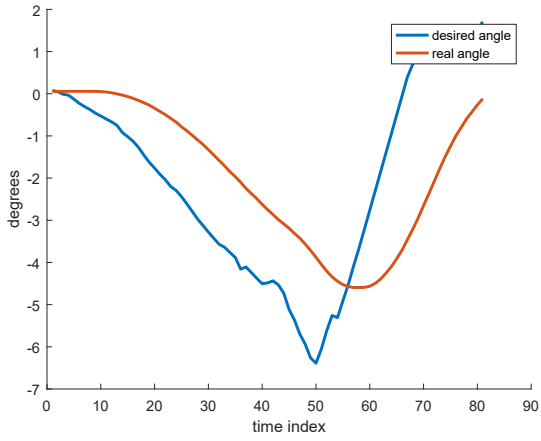


Figure 3.9: Desired inclination angles and the real inclination induced by the robot arm. Internal impedence controller of the robot arm introduces delay in commanded signals which renders the execution of turbalent feedforward control signals unattainable.

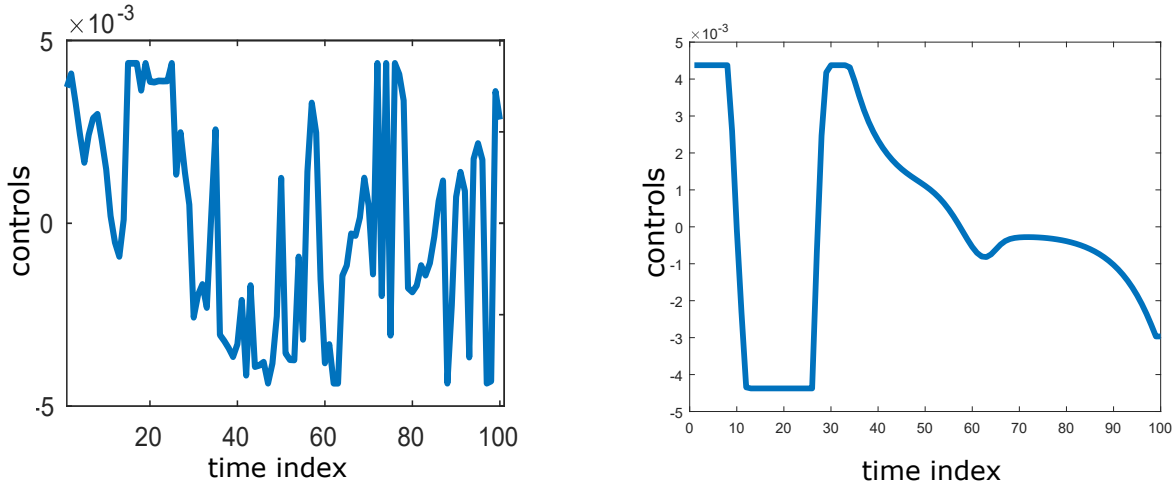
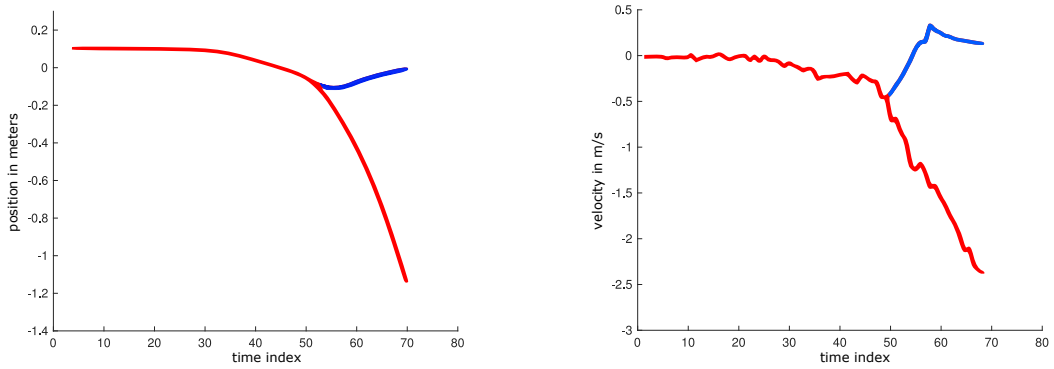


Figure 3.10: Feedforward control signals generated by PILCO-iLQR (left) and PIREM-iLQR (right). Delay in the commanded signal due to the internal impedence control mechanism of the robot arm adversely affects the applicability of the proposed signals.

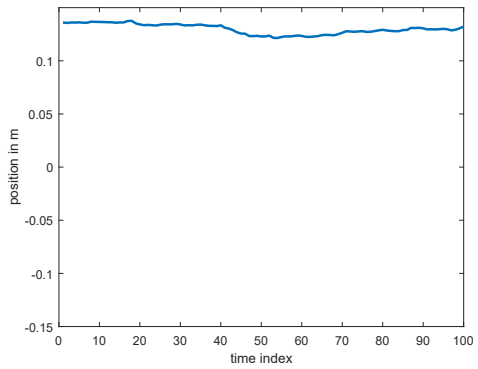
efficieny of closed-loop feedback terms computed by the initial policy optimization based on the approximate model. While feedback terms form the substantial part of the control signals in the simuation environment, the additive control signals produced by these terms are quite small compared to the feedforward terms proposed by PILCO-iLQR and PIREM-iLQR.



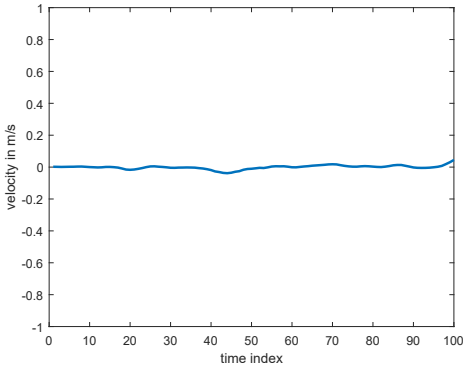
(a) Position (b) Velocity

Figure 3.11: Real values (blue line) and extrapolated values (red line) after collision with the barrier. A minor change in ball's state before the collision leads to vast deflections at the final states, which dominate the cost function and degrade the optimization process. Besides a misguided policy update, such variance at the inferred values yields inaccurate training data for the Gaussian process.

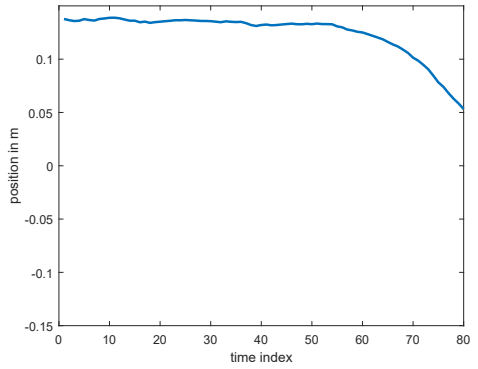
The most crucial cause behind the failure of real robot experiments is the inconsistent behaviour exhibited by the manipulated spherical objects. Fig. 3.12 demonstrates the irregularity in ball's behaviour with respect to our system definiton. Balloon filled with rice generates totally different trajectories under the same policy and roughly same initial position and velocity. Depending on the bumpiness of the contact surface, the ball may get stuck at the original position during the entire horizon of the rollout, or it can be located halfway between the target and initial position with ongoing rolling motion at the final time instant. It may also have reached and missed the target position towards the end of the rollout. Such dynamics of the manipulated object produces datapoints with nonsmooth changes for the Gaussian process regression and undermines the convergence of the optimal policy.



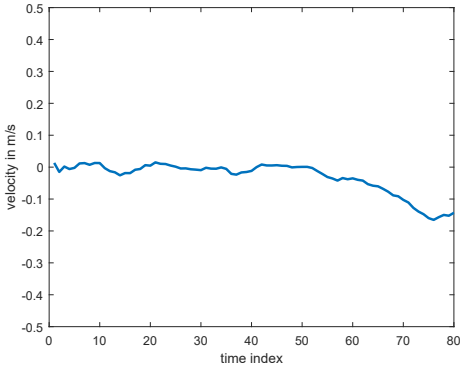
(a) Position of the ball stuck at the initial position.



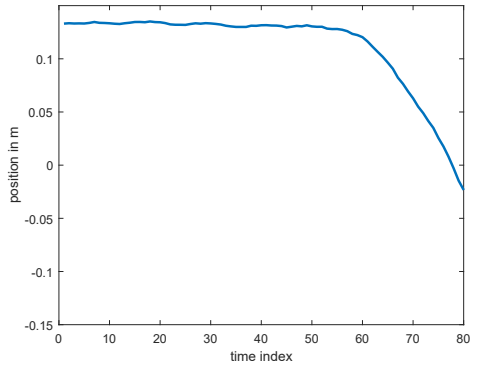
(b) Velocity of the ball stuck at the initial position.



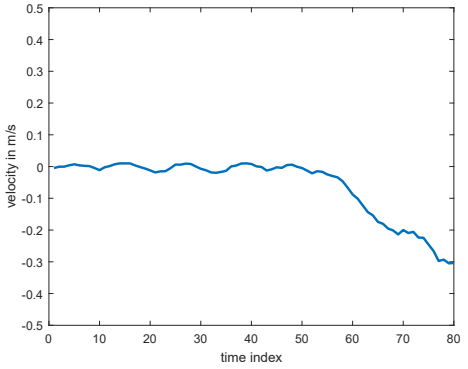
(c) Position of the ball located halfway at the final time instant.



(d) Velocity of the ball located halfway at the final time instant.



(e) Position of the ball that reached and missed the center.



(f) Velocity of the ball that reached and missed the center.

Figure 3.12: Inconsistency in the observed trajectories given the same policy and similar initial states. Balloon filled with rice may occasionally get stuck at the initial position throughout the whole episode (3.12(a) - 3.12(b)) if the instantenous contact surface is flat. It can also be located halfway between the target and initial position with ongoing rolling motion at the final time instant (3.12(c) - 3.12(d)) . The target may also be already reached and missed as the rollout terminates (3.12(e) - 3.12(f)).

Chapter 4

Conclusion

In this projected, we combined the plain PILCO and PIREM algorithms with the iterative Linear Quadratic Regulator method to observe better convergence behaviour towards the optimal policy. Experiments applied in simulation environment demonstrate that all of the discussed algorithms, namely PILCO, PIREM, PILCO-iLQR, open-loop PIREM-iLQR and closed-loop PIREM-iLQR, are capable of obtaining a (sub)optimal policy for balancing of inverted pendulum and soft ball-and-plate tasks. State trajectories generated by the these algorithms are quite similar to each other. While all algorithms except for the closed-loop PIREM-iLQR require the same amount of rollouts for the majority of tasks, closed-loop PIREM-iLQR outperforms the rest of the algorithms in terms of total amount of required interactions with the environment. While incorporation of iLQR significantly reduces the computational complexity of policy improvement for PIREM, the same improvement cannot be observed with PILCO. On the contrary, time spend for policy search per rollout slightly increases for ball-and-plate scenario when combining iLQR with PILCO.

The ball-and-plate experiment conducted with real robot arm is only successful with the rigid sponge ball controlled with closed-loop PIREM-iLQR method. The algorithm converged to a stable policy after the fourth rollout with an average final position error of 2 cm and a standard deviation of 2 cm along the x-axis over 7 executions. Inconsistency in the dynamics of manipulated ball, delay in control signals, inaccuracies arising from linear extrapolation used under certain conditions, absence of sparse GP regression and ineffectiveness of feedback terms received from the approximate model are the possible causes for partial failure of the real experiments.

List of Figures

Bibliography

[AQN06] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 1–8, New York, NY, USA, 2006. ACM. URL: http://doi.acm.org/10.1145/1143844.1143845, doi:10.1145/1143844.1143845.

[AS97] C. G. Atkeson and J. C. Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 3557–3564 vol.4, April 1997. doi:10.1109/ROBOT.1997.606886.

[CGLR03] J.Q. Candela, A Girard, J Larsen, and C.E. Rasmussen. Propagation of uncertainty in bayesian kernel models: Application to multiple step ahead forecasting. pages II – 701, 05 2003. doi:10.1109/ICASSP.2003.1202463.

[CKY+16] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. *CoRR*, abs/1610.00529, 2016. URL: http://arxiv.org/abs/1610.00529, arXiv:1610.00529.

[DFR15] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, Feb 2015. doi:10.1109/TPAMI.2013.218.

[FLA15] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. *CoRR*, abs/1509.06841, 2015. URL: http://arxiv.org/abs/1509.06841, arXiv:1509.06841.

[GHLL16] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633, 2016. URL: http://arxiv.org/abs/1610.00633, arXiv:1610.00633.

[KABP13] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 09 2013. doi:10.1177/0278364913495721.

[LKD+18] Kendall Lowrey, Svetoslav Kolev, Jeremy Dao, Aravind Rajeswaran, and Emanuel Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. *CoRR*, abs/1803.10371, 2018. URL: http://arxiv.org/abs/1803.10371, arXiv:1803.10371.

[LWA15] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *CoRR*, abs/1501.05611, 2015. URL: http://arxiv.org/abs/1501.05611, arXiv:1501.05611.

[MAF+16] William Montgomery, Anurag Ajay, Chelsea Finn, Pieter Abbeel, and Sergey Levine. Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. *CoRR*, abs/1610.01112, 2016. URL: http://arxiv.org/abs/1610.01112, arXiv:1610.01112.

[MAY66] By DAVID MAYNE. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

[ML93] Matthew T Mason and Kevin M Lynch. Dynamic manipulation. In *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, pages 152–159. IEEE, 1993.

[Pow12] Warren B. Powell. Ai, or and control theory: A rosetta stone for stochastic optimization. 2012.

[PT08] D. Prattichizzo and J. Trinkle. *Grasping*. Springer, 2008.

[RLS18] F. Ruggiero, V. Lippiello, and B. Siciliano. Nonprehensile dynamic manipulation: A survey. *IEEE Robotics and Automation Letters*, 3(3):1711–1718, July 2018. doi:10.1109/LRA.2018.2801939.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[SB98] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[SG06] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages

1257–1264. MIT Press, 2006. URL: http://papers.nips.cc/paper/2857-sparse-gaussian-processes-using-pseudo-inputs.pdf.

[SYFL17] Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee. Pi-rem: Policy improvement with residual model learning. 01 2017.

[TL05] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306 vol. 1, June 2005. doi:10.1109/ACC.2005.1469949.

[TMT14] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. 05 2014. doi:10.1109/ICRA.2014.6907001.

[TT10] Yuval Tassa and Emanuel Todorov. Stochastic complementarity for local control of discontinuous dynamics. In *Robotics: Science and Systems*, 2010.

[VSB92] Oskar Von Stryk and Roland Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, 37:357–373, 12 1992. doi:10.1007/BF02071065.

[Wil92] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992. URL: https://doi.org/10.1007/BF00992696, doi:10.1007/BF00992696.