Models for Data-Efficient Reinforcement Learning on Real-World Applications

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von
Andreas Dörr
aus Saarbrücken

Hauptberichter:	Prof. Dr. Marc Toussaint
Mitberichter:	Prof. Dr. Sebastian Trimpe
Tag der mündlichen Prüfung:	07.10.2021

Institut für Parallele und Verteilte Systeme
der Universität Stuttgart

2021

Models for Data-Efficient Reinforcement Learning on Real-World Applications

Andreas Doerr
2021

Models for Data-Efficient Reinforcement Learning on Real-World Applications

Acknowledgements

Great colleagues and friends accompanied my journey into the world of machine learning, robotics, and reinforcement learning. What I have learned and discovered, what I have developed and implemented, and what I have published is most often rooted in their support, discussions, and motivation. I want to thank all of you, my supervisors and colleagues, my discussion partners and friends who made this work happen.

Coming from engineering and control, the Machine Learning and Robotics Lab at the University of Stuttgart paved my way into the exciting world of intelligent systems. I am grateful for the supervision by Prof. Marc Toussaint and Dr. Nathan Ratliff, who guided me into the world of inverse reinforcement learning, motion planning and robotics with their stunning grasp of mathematical concepts.

Getting to know the people at the Max Planck Institute for Intelligent Systems reinforced my decision to pursue research in the direction of autonomous learning. I especially enjoyed the deep understanding, technical discussions, and strive for novel solutions, which I experienced at Prof. Stefan Schaal's Autonomous Motion Department. Thank you, Stefan, for setting up the AMD and facilitating our freedom to pursue our projects and topics! Thanks to all my colleagues, Jeanette, Nathan, Ludo, Sebastian, Felix, Friedrich, my office-mates Alonso and Dominik, and many others at the AMD and MPI-IS.

I found a second perspective on learning systems with Dr. Sebastian Trimpe and his Intelligent Control Systems group, where control theory and machine learning ideas are combined. I want to thank you for setting me up to conduct thorough and trustworthy research. I appreciate your great eye for details, control background, and continuous discussions to figure out what is going on in our methods and experiments.

In the last years, I found my second home in the Bosch Center for Artificial Intelligence. Thank you, Yasser, for pulling off this AI organization at Bosch and bringing me on board. With the colleagues and friends at Bosch, I got to experience a great mix of academic research, industrial products, and development in-between. Thank you, Duy and Chris, for supervising my work and for pushing and inspiring my projects when I was stuck. Thanks to Heiner, Martin, Stefan, and all the others from BCAI, CR, and our Bosch partners! I appreciate the fruitful discussions with Prof. Marc Toussaint and his external views and advice on my projects!

Finally, this list would not be complete without my friends and family outside academia who supported my projects, listened to my rehearsal talks, or proofread my papers and publications (thanks, Barbara!). I thank my parents, who encouraged and supported my scientific efforts right from the beginning. Thank you to you, Sandra and Emily, for facilitating and being with me as my family in crazy last-minute deadlines, when struggling with experiments, or going to conferences worldwide!

Andreas Doerr
Stuttgart, May 2021

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Large-scale deep *Reinforcement Learning* is strongly contributing to many recently published success stories of *Artificial Intelligence*. These techniques enabled computer systems to autonomously learn and master challenging problems, such as playing the game of *Go* or complex strategy games such as *StarCraft* on human levels or above. Naturally, the question arises which problems could be addressed with these Reinforcement Learning technologies in *industrial applications*. So far, machine learning technologies based on (semi-)supervised learning create the most visible impact in industrial applications. For example, image, video or text understanding are primarily dominated by models trained and derived autonomously from large-scale data sets with modern (deep) machine learning methods. Reinforcement Learning, on the opposite side, however, deals with temporal decision-making problems and is much less commonly found in the industrial context. In these problems, current decisions and actions inevitably influence the outcome and success of a process much further down the road.

This work strives to address some of the core problems, which prevent the effective use of Reinforcement Learning in industrial settings. Autonomous learning of new skills is always guided by existing priors that allow for generalization from previous experience. In some scenarios, non-existing or uninformative prior knowledge can be mitigated by vast amounts of experience for a particular task at hand. Typical industrial processes are, however, operated in very restricted, tightly calibrated operating points. Exploring the space of possible actions or changes to the process naively on the search for improved performance tends to be costly or even prohibitively dangerous.

Therefore, one reoccurring subject throughout this work is the emergence of priors and model structures that allow for efficient use of all available experience data. A promising direction is *Model-Based Reinforcement Learning*, which is explored in the first part of this work. This part derives an automatic tuning method for one of the most common industrial control architectures, the PID controller. By leveraging all available data about the system's behavior in learning a system dynamics model, the derived method can efficiently tune these controllers from scratch.

Although we can easily incorporate all data into dynamics models, real systems expose additional problems to the dynamics modeling and learning task. Characteristics such as non-Gaussian noise, latent states, feedback control or non-i.i.d. data regularly prevent using off-the-shelf modeling tools. Therefore, the second part of this work is concerned with the derivation of modeling solutions that are particularly suited for the reinforcement learning problem.

Despite the predominant focus on model-based reinforcement learning as a promising, data-efficient learning tool, this work's final part revisits model assumptions in a separate branch of reinforcement learning algorithms. Again, generalization and, therefore, efficient learning in model-based methods is primarily driven by the incorporated model assumptions (e.g., smooth dynamics), which real, discontinuous processes might heavily violate. To this end, a model-free reinforcement learning is presented that carefully reintroduces prior model structure to facilitate efficient learning without the need for strong dynamic model priors.

The methods and solutions proposed in this work are grounded in the challenges experienced when operating with real-world hardware systems. With applications on a humanoid upper-body robot or an autonomous model race car, the proposed methods are demonstrated to successfully model and master their complex behavior.

Zusammenfassung

Hinter zahlreichen aktuellen Erfolgen von *Künstliche Intelligenzer* (KI) stecken Methoden des *Deep Reinforcement Learning* (DRL). Hiermit wurden Probleme gelöst, die bisher dem menschlicher Intelligenz vorbehalten waren. Computer können nun in komplexen Problemen, wie z.B. dem *Go* Spiel oder Strategie-Spielen wie „StarCraft", menschliche Performance erreichen oder sogar übertreffen. In industriellen Anwendungen ist das Potential dieser Methoden hingegen weitaus weniger ausgeschöpft. In diesem Feld sind Erfolge von KI vorwiegend auf *(semi-)supervised* Lernverfahren zurückzuführen. In den Bereichen der Bild-, Video- oder Text-Verarbeitung haben *Deep-Learning* (DL) Methoden klassische Ansätze fast vollständig verdrängt. Im Gegensatz dazu beschäftigt sich *Reinforcement Learning* (RL) mit zeitlichen Entscheidungsprozessen. In diesen Problemen können aktuelle Entscheidungen langfristige Auswirkungen auf die Performance eines Systems haben.

In dieser Arbeit werden einige Kernprobleme von RL im industriellen Bereich adressiert. Grundsätzlich sind zum Lernen Annahmen erforderlich, um aus bisheriger Erfahrung Rückschlüsse auf ähnliche Ereignisse zu ziehen. In manchen Anwendungen sind hierfür große Mengen an Daten verfügbar, sodass Lernen trotz schwacher Annahmen über die zugrundeliegenden Prozesse möglich wird. Typische industrielle Prozesse sind diesbezüglich jedoch oft stark limitiert. Sie operieren in klar eingegrenzten Arbeitspunkten und Abweichungen, die zum Lernen erforderlich wären, sind selten möglich, teuer oder sogar schädlich.

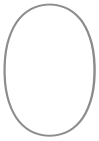
Wesentliche Fragestellung in dieser Arbeit ist somit welche Annahmen effizientes Lernen ermöglichen. Eine vielversprechende Richtung ist *modellbasiertes RL* (MBRL). Der erste Teil dieser Arbeit stellt eine Methode vor, die es ermöglicht eine der meistverwendetsten Regler Architekturen, den PID Regler, optimal für ein bestehendes Problem anzupassen. Mit diesen Methoden können alle verfügbaren Daten in einem Modell der Systemdynamik zusammengeführt werden. Hiermit kann der gewünschte Regler autonom und praktisch ohne Vorwissen ideal eingestellt werden.

Obwohl Modelle der Systemdynamik alle verfügbaren Daten nutzen können, gibt es in der Praxis zahlreiche Probleme bei der Modellierung realer Systeme. Typisch sind unbeobachtete Zustände, komplexes stochastisches und korreliertes Verhalten, die oft den Grundannahmen der Modelle widersprechen. Im zweiten Teil dieser Arbeit werden daher Methoden zur Modellierung abgeleitet, die auf diese Probleme und auf die Verwendung im Kontext von RL zugeschnitten sind.

Ähnlich wie in MBRL, wird effizientes Lernen auch in anderen RL Klassen hauptsächlich durch die verfügbaren Annahmen, wie z.B. Glattheit, bestimmt. Diese Annahmen können bei diskontinuierlichen Systemen (z.B. Roboter die mit Gegenständen in ihrer Umgebung interagieren) unzutreffend sein und somit den Lernprozess behindern. Als Lösung zu diesem Problem beschäftigt sich der letzte Teil dieser Arbeit mit den *Modell freien* RL Methoden. Mit geeigneten Annahmen wird auch hier daten-effizienteres Lernen ermöglicht. Hierzu stellt dieser Arbeit eine neue Methode vor, die mit möglichst wenig Modell-Annahmen erfolgreich neues Verhalten erlernen kann.

Die Methoden und Lösungen, die in dieser Arbeit vorgeschlagen werden, sind abgeleitet aus den Anforderungen und Besonderheiten realer Hardware. Am Beispiel eines humanoiden Roboters oder eines autonomen Modellrennfahrzeugs, werden die Fähigkeiten der vorgeschlagenen Algorithmen demonstriert komplexes Verhalten realer System zu modellieren und zu beherrschen.

Prologue



Introduction

Learning from interactions with the environment surrounding us is one of the most natural ways for humans to extend their abilities and learn how to reach their goals. Reinforcement Learning (RL) as a problem formulation tries to capture this situation in its most general setting. An *agent* is interacting with an unknown environment by choosing from all possible action opportunities, observing some effect, and reiterating with a potential new action given its improved understanding of the environment. A reward signal indicates some notion of goal fulfillment to the agent, such that ultimately, the agent aims at maximizing its long-term, accumulated reward.

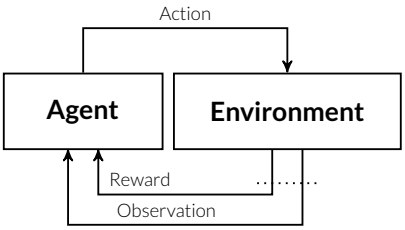


Figure 1: Visualization of the typical reinforcement learning interaction loop between an agent and its environment.

RL and, in particular, Deep Reinforcement Learning (DRL), which employs deep Neural Networks (NNs) for learning, achieved some impressive results in the past. Games are commonly utilized as a benchmark and learning environment to assess the performance of the computer's reinforcement learning capabilities. Starting from the early RL successes in backgammon [151], both the available compute and the reinforcement learning tools evolved to a point where automatically learned strategies could reach super-human performance on more complex games, such as the ATARI video games [98]. In 2016 Deepmind's AlphaGo defended Lee Sedol, one of the best players in the game of go [138] and most recently, OpenAI's RL agent defeated the world champing in the 5 vs. 5 Dota 2 game [15].

[151] Tesauro, "Temporal difference learning and TD-Gammon," 1995

[98] Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller, "Playing atari with deep reinforcement learning," 2013

[138] Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, and Lanctot, "Mastering the game of Go with deep neural networks and tree search," 2016

[15] Berner, Brockman, Chan, Cheung, Dębiak, Dennison, Farhi, Fischer, Hashme, and Hesse, "Dota 2 with large scale deep reinforcement learning," 2019

However, many of the reported successes of RL are achieved in well-understood environments, which can be reasonably accurately and cheaply simulated. Besides the previously mentioned game-environments [151, 98, 138, 15] this scheme applies as well to learning the control of physical systems, such as robots [11, 1]. The required millions of system interactions, which can be obtained in large-scale, distributed simulations, are typically infeasible to obtain on real systems due to wear-and-tear, time-, safety-, or cost-constraints.

[11] Barth-Maron, Hoffman, Budden, Dabney, Horgan, Tb, Muldal, Heess, and Lillicrap, "Distributed distributional deterministic policy gradients," 2018

[1] Akkaya, Andrychowicz, Chociej, Litwin, McGrew, Petron, Paino, Plappert, Powell, and Ribas, "Solving rubik's cube with a robot hand," 2019

In all the previously mentioned problems, RL was able to derive high-performance behavioral strategies in complex, NP-hard problems, which cannot be solved optimally. Instead of manually engineering heuristic strategies based on proxy objectives, RL automatically strives to find an approximately optimal strategy. This benefit is conceptually comparable to the breakthroughs in computer vision or natural language processing, where machine learning systems largely replaced and superseded manual feature engineering. At the same time, RL methods are able to derive solutions that are robust to uncer-

tainties in the underlying system [153] and scale to high-dimensional input/output spaces [79]. A critical question in RL is how to leverage these benefits in real-world industrial applications.

The RL problem and the RL methods are a superset of many related, more specific problem and solution instances. For example, one can find planning, optimal control, and system identification in typical model-based RL concepts. Development and progress in RL can, therefore, originate from at least two perspectives.

A first class of algorithms and concepts is concerned with improving the individual RL sub-problems and adapting these sub-methods towards the overall setting of reinforcement learning. For example, different notions and update rules exist to judge the value of specific actions from data by taking more or less long-term dependencies into account [28]. Similarly, novel modeling schemes have been presented to account for uncertainty due to missing data [106]. In this work, we will present two novel schemes for modeling a system's dynamics, which improve the typical dynamics modeling and system identification in RL's direction in Part II. By exploiting the subsequent RL problem's structure, we can tailor these methods to obtain more data-efficient learning in the RL setting.

As mentioned previously, the improvement of individual methods strives to make the best use of all available data to solve the RL problem without sacrificing its generality. In particular, for RL on a real-world system, the second angle of attack is to narrow down the most general RL framework by including specific structure and assumptions valid for this particular domain or problem class. In this work, Sec. 7 discusses an example of a general-purpose, model-free RL method made significantly more data-efficient by exploiting an assumption over the nature of available actions. Similarly, the method presented in Sec. 10 achieves data-efficient learning with minimal model assumptions for a class of problems, where smoothness assumptions in action space are valid.

[153] Tobin, Fong, Ray, Schneider, Zaremba, and Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017

[79] Lange, Riedmiller, and Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," 2012

[28] Dann, Neumann, and Peters, "Policy evaluation with temporal differences: A survey and comparison," 2014

[106] Murray-Smith, Sbarbaro, Rasmussen, and Girard, "Adaptive, cautious, predictive control with Gaussian process priors," 2003

1

Outline of this Work

This work is centered around the idea of identifying and improving model-assumptions to enable efficient reinforcement learning in industrial domains. Typical questions which are discussed throughout this work are (i) how to make the most out of available data? (ii) where to incorporate prior knowledge? and (iii) what model assumptions can and should we make? This work comprises three main parts with the following contributions. More extensive details on the scientific contributions can be found in the introduction sections of the respective parts.

Model-Based Reinforcement Learning for Tuning of PID Controllers

The first part of this work introduces an extended version of the Probabilistic Inference for Learning COntrol (PILCO) [31] framework. In particular, this extended version allows for the tuning of multivariate and coupled PID controllers, which can be typically found in many industrial applications. It is demonstrated how PID control architectures can be seamlessly integrated into the probabilistic interpretation of the model-based policy search methodology in PILCO. The efficiency of the resulting method is demonstrated for robotic and automotive applications. Work presented in this part has been previously published in

> A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, and S. Trimpe. "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2017

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

Learning Models for Model-Based Reinforcement Learning

In the second part of the thesis, two novel methods *Multi-Step Gaussian Processes (MSGP)* [39] and *Probabilistic Recurrent State-Space-Models (PR-SSM)* [40] are presented. Both aim at learning improved long-term predictive models of the system dynamics from interaction data. These methods are derived with the requirements of model-based RL in mind. They address typical problems of real-world systems, such as noise distribution and non-observed states, and leverage the structure

[39] Doerr, Daniel, Nguyen-Tuong, Marco, Schaal, Toussaint, and Trimpe, "Optimizing Long-term Predictions for Model-based Policy Search," 2017

[40] Doerr, Daniel, Schiegg, Nguyen-Tuong, Schaal, Toussaint, and Trimpe, "Probabilistic Recurrent State-Space Models," 2018

of the subsequent learning problem to improve model performance. Work presented in this part has been previously published in

> A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, M. Toussaint, and S. Trimpe. "Optimizing Long-term Predictions for Model-based Policy Search." In: *Conference on Robot Learning (CORL)*. 2017, pp. 227–238

> A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. "Probabilistic Recurrent State-Space Models." In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1280–1289

Model Assumptions in Model-Free Reinforcement Learning

The (seemingly) abundance of (model) assumptions in model-free RL leads to impressive flexibility in learning novel skills but at the same-time catastrophic data-efficiency. This final part of this work reintroduces mild model-assumptions into model-free RL to recover some of the data-efficiency experiences in model-based methods. In this part, *Deep Deterministic Off-Policy Gradients (DD-OPG)*, an off-policy, trajectory-based policy gradient method is introduced that leverages all available trajectory data. A single model-assumption in action space is required to learn from previous experience and judge the influence of past data. Work presented in this part has been previously published in

> A. Doerr, M. Volpp, M. Toussaint, S. Trimpe, and C. Daniel. "Trajectory-based off-policy deep reinforcement learning." In: *International Conference on Machine Learning (ICML)*. 2019, pp. 1636–1645

Conclusion

This work concludes with a summary of the novel contributions presented in this thesis. Indications of the main problems and solutions leading to the novel methods are given. A short outlook is provided into some fundamental directions for future work.

2

Preliminaries

The following sections introduce the main problem classes, nomenclature and preliminaries, which the three main chapters of this work are based on. In particular, this section provides an overview on the reinforcement learning problem in Sec. 2.1 and the types and (dis)advantages of typical reinforcement learning methods in Sec. 2.2. The connection between RL and (optimal) control, as well as the nomenclature in the control literature is outlined in Sec. 2.3. Finally, a brief introduction into Gaussian Processes and Gaussian Process Regression is given, which is a main building-block of the probabilistic models utilized throughout this work.

2.1 The Reinforcement Learning Problem

In the following section, we formalize the general Reinforcement Learning problem as it is most commonly used in related work and throughout this work. Problems very much similar to the RL problem are considered from the control community as well using a different notation. This section introduces the RL perspective, whereas Sec. 2.3 presents the problems typically encountered in the classical control setting and the respective notation.

Most of today's RL literature is based on the repeated interactions between an agent and and environment at discrete points in time as depicted in Fig. 2. In problem settings, where all information about the current state of the system is revealed to the agent, the RL problem is formalized in the framework of a Markov Decision Processes (MDPs). A state s is said to fulfill the Markov property, if given the state, all relevant information is available to predict the future development of the system. In most real systems, only partial information about the system's state is available from sensor measurements, e.g., cameras, accelerometers or lidar readings. This situation is described by Partially Observable Markov Decision Processes (POMDPs), where the system receives a (partial) *observation* o from the system. The following RL formalism mainly focuses on the MDP problem, with some pointers to POMDPs at the end of this section. A comprehensive introduction into RL can be found in [146].

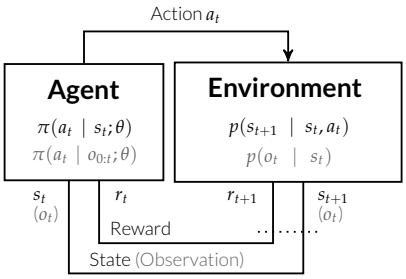


Figure 2: Visualization of the discrete time interaction loop between an agent and its environment in a typical reinforcement learning problem.

[146] Sutton and Barto, *Reinforcement learning: An introduction*, 1998

Markov Decision Process

The Markov Decision Process (MDP) is defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, p_0)$. The agent interacts with the environment by taking an action a_t from a set of all possible actions $\mathcal{A}$ in timestep t. The environment is in state s_t out of all possible states $\mathcal{S}$. Following the (probabilistic) dynamics of the environment, the distribution of the next state s_{t+1} is distributed according to $p(s_{t+1} \mid s_t, a_t)$. The agent typically starts in a fixed initial state $s_0 = \delta(s)$ or a distribution thereof, e.g., $s_0 \sim p_0(s)$.

The agent decides on future actions based on the current state of the system s_t according to its *policy*. This policy can either be a deterministic mapping of states to actions

$$a_t = \mu(s_t)\,, \tag{1}$$

or a distribution of possible actions, where one concrete action instance is sampled from

$$a_t \sim \pi(\cdot \mid s_t) \tag{2}$$

For the purpose of learning, policies are most commonly represented by parametrized function approximators. In cases with finitely many states and actions, tabular policies are possible, such that each parameter corresponds to one table entry. In more complex environments with larger number or potentially infinitely many states and actions, continuous function approximators are used. For example, linear feature-based [118], radial basis function networks (RBF) [33], or (deep) neural networks (DNNs) [98] are typically employed. We denote a policy with parameters θ as μ_θ in the deterministic case or π_θ in case of a probabilistic policy.

From the interaction of an agent with the environment as depicted in Fig. 2, a continuous stream of states and actions is generated, which we call a trajectory τ.

$$\tau = (s_0, a_0, s_1, a_1, \ldots) \tag{3}$$

In cases where a parametrized policy is executed, we denote the trajectory as τ_θ to indicate the dependency from the policy parameters. Given the initial state distribution $\rho_0(s_0)$, the system dynamics $p(s_{t+1} \mid s_t, a_t)$ and policy $\pi_\theta(a_t \mid s_t)$, the distribution over trajectories of length T is given by

$$p(\tau \mid \theta) = \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1} \mid s_t, a_t)\pi(a_t \mid s_t) \tag{4}$$

The graphical model of the agent-environment interaction and the resulting trajectory distribution is shown in Fig. 3. Computing the tra-

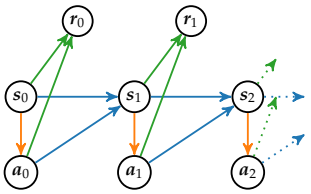


Figure 3: Graphical model of the Markov Decision Process. Transitions are due to the policy $\pi(a_t \mid s_t; \theta)$ (⟶), system dynamics $p(s_{t+1} \mid s_t, a_t)$ (⟶), and reward function $r(s_t, a_t)$ (⟶).

[118] Peters and Schaal, "Reinforcement learning of motor skills with policy gradients," 2008

[33] Deisenroth, Fox, and Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," 2015

[98] Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller, "Playing atari with deep reinforcement learning," 2013

jectory distribution is non-trivial and typically not possible in closed-form. This is due to the complex distributions arising from typical (unknown) non-linear transition models and non-linear policies. Throughout this thesis we will explore different techniques to efficiently but approximately compute this integral (e.g. using moment matching in Part I and chapter 7 or sampling in chapter 8). Alternatively, methods will be discussed that directly estimate statistics based on this distribution using trajectory samples only (cf., chapter 10).

The agent receives an (immediate) reward r_t in each timestep according to the reward function $r_t = r(s_t, a_t)$[1]. The goal of the agent is to maximize the cumulative reward which is called *return* $R(\tau)$.

[1] Reward functions might be defined as well as $r(s_t, a_t, s_{t+1]})$ or $r(s_t)$ depending on the context.

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \tag{5}$$

Depending on the problem, a finite or infinite horizon T might be employed. The discounting factor $\gamma \in (0, 1]$ is a design element in RL to decide how much short term reward is favored over long term rewards.

Based on the trajectory distribution (4) and a trajectory's return (5), the goal of the RL agent can be formalized as maximizing the expected return as given by

$$J(\theta) = \int_{\tau} p(\tau \mid \theta) R(\tau) d\tau = \mathop{\mathbb{E}}_{\tau \sim p(\tau|\theta)} [R(\tau)] \tag{6}$$

Thus, the RL strives to find the optimal policy $\pi^* = \pi(\theta^*)$, which solves

$$\theta^* = \arg\max_{\theta} J(\theta) \,. \tag{7}$$

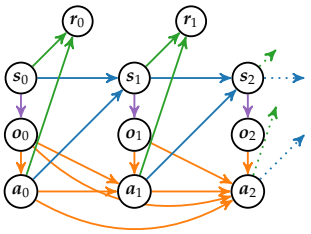


Figure 4: Graphical model of a Partially Observable Markov Decision Process. Additionally to the MDP in Fig. 3, the observation model $p(o_t \mid s_t)$ (⟶) is depicted. As indicated in the model, the policy in a POMDP is potentially influenced by all previous observations and actions $\pi(a_t \mid o_{0:t}, a_{0:t-1}; \theta)$ (⟶).

Partially Observable Markov Decision Process

In many real-world, e.g., industrial problems, full state information cannot be achieved due to the cost of sensors or complexity of the system. Instead, only some parts of the system are measured, e.g., a vehicle position. Other states, such as the vehicle's velocity, need to be inferred. This situation is captured by *Partially Observable Markov Decision Processes* (POMDPs), where the agent only obtains observations o_t of the full state s_t according to some conditional observation distribution $p(o_t \mid s_t)$. The graphical model of a agent-environment interaction in the POMDP setting is depicted in Fig. 4. Typical problems arising from this abundance of true, Markovian state information in real-world applications are discussed in Sec. 6.3.

In contrast to the MDP case, the agent's policy typically needs to incorporate information from previous observations and actions, to recover the missing state information. Thus, the action distribution is typically conditioned on the full, past interaction, i.e. $\pi(a_t \mid o_{0:t-1}, a_{0:t-1})$. In some POMDP algorithms, the notion of a belief state b_t is introduced, which captures all information from previous interactions, i.e., the agent maintains a belief of what the actual system's state s_t could be in form of a internal believe state distribution $p(b_t \mid o_{0:t-1}, a_{0:t-1})$. Similarly, methods in this work will derive a latent state, which captures the underlying true state-information. These model-learning methods are discussed in Sec. 7 and Sec. 8.

2.2 Classes of Reinforcement Learning Methods

This section presents an overview of different classes of RL algorithms. The main modeling problem in each of the RL classes is highlighted, which raises the question of where and how to employ model assumptions to learn efficiently. Each class of RL methods poses unique challenges and opportunities to incorporate model knowledge when applied to a real, industrial problem. The main part of this work will be concerned with model assumptions in the first two classes of RL methods.

Most of today's Reinforcement Learning algorithms are designed around the MDP or POMDP formalism as defined in Chap. 2.1. Therefore, the input to these algorithms is given by time-discrete interaction-data-tuples (s_t, a_t, r_t, s_{t+1}). Furthermore, these algorithms are usually aiming to derive an optimal policy $\pi^*(a \mid s; \theta)$. This policy is a state-feedback controller, which decides on an action a (or a distribution thereof) for each system state s. Such a policy is parametrized by parameters $\theta \in \mathcal{R}^p$. Thus, all the discussed methods belong to the domain of Policy Search (PS) methods [34]. The RL problem is reduces from finding the optimal functional dependency to locating the optimal set of policy parameters θ^*.

[34] Deisenroth, Neumann, and Peters, "A survey on policy search for robotics," 2013

Ultimately, all RL algorithms need to incorporate the available interaction data into a learning rule to select an improved policy. Historically, a broad range of methods evolved around different possible model assumptions. Most notably, three main branches of work correspond to three different modeling problems in which data can be incorporated. A simplified view of this RL methodology cosmos is depicted in Fig. 5[2].

[2] Credit for RL landscape overview and visualization to Prof. Marc Toussaint

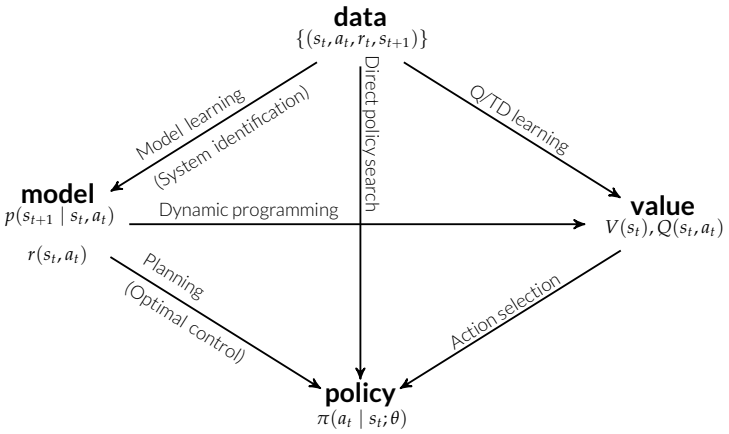


Figure 5: Simplified depiction of the main algorithm categories in the reinforcement learning realm. Given interaction data (state, action, reward, next state) from an agent with an environment, the ideal policy should be inferred.

Model-Based RL

A first branch of RL methods centers around a (probabilistic) model $\hat{p}$ of the system's true dynamics, e.g., $p(s_{t+1} \mid s_t, a_t)$ in case of an MDP. This regime is most related to the field of optimal control since, subsequently, typical techniques such as model-based planning and optimization of finite-horizon costs/reward are employed. A significant portion of methods in this branch of control literature deals with elementary models, such as linear or linear Gaussian [20] models. This restriction enables analytic, closed-form solutions to policy search problem. One such example is the Linear Quadratic Regulator (LQR) [3] dealing with linear models and quadratic cost terms. Many mechanical systems are designed to exhibit close to linear behavior around a specific operating point. Therefore, strong smoothness assumptions such as linear models or standard variants of GP models demonstrate impressive generalization capabilities from only a few data-points.

[20] Burl, *Linear optimal control: H (2) and H (Infinity) methods*, 1998

[3] Anderson and Moore, *Optimal control: linear quadratic methods*, 2007

Restricting the class of models to linear models is, however, a too strong assumption for many non-linear, real-world problems. Therefore, a multitude of different, more flexible, models have been proposed for model-based RL. Ranging from linear-Gaussian-models [83], to Gaussian processes [35], and deep neural network models [54], the amount of required data to learn a new task clearly increases with the flexibility of the employed models. A good overview of current model-based RL methods is provided in [99] and a benchmark with state-of-the-art MBRL methods can be found in [80]. In this work, methods in the realm of model-based RL are discussed in part I and part II. The first discusses model-based RL as a data-efficient tool for tuning PID control architectures in the context of continuous control. In this part (cf. Sec. 3.3), a more detailed overview of model-based RL and related work is provided. The second part focuses on exploiting

[83] Levine and Koltun, "Guided Policy Search," 2013

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

[54] Gal, McAllister, and Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," 2016

[99] Moerland, Broekens, and Jonker, "Model-based reinforcement learning: A survey," 2020

[80] Langlois, Zhang, Zhang, Abbeel, and Ba, "Benchmarking model-based reinforcement learning," 2019

the RL problem's structure to improve the performance of learned dynamics models for this specific use-case.

Model-Free RL

The second branch of methods directly operates on the expected return J (cf. (5)) from executing a policy π on the system. Therefore, these methods model in the space of $J(\theta)$, which is the expected return given the specific policy parameters. Typical representatives of these methods are Evolutionary Strategies (ES) [150], Bayesian Optimization (BO) [90], and Policy Gradient (PG) [169] methods. Similar to the model-based methods, it is again eminent how strong model-assumptions are and how well these assumptions fit the true dependency $J(\theta)$ for data-efficient learning. Irrespectively, these methods are inherently restricted in their data-efficiency by only incorporating summary statistics (i.e., the expected return) of all available interaction data. There exist, however, methods to alter the utilized summary statistic and reintroduce structure from the MDP/POMDP problem to improve on data-efficiency. Examples of that can be found predominantly in PG [67] methods, but also in BO [170]. Part II of this work deals with model-assumptions in the so-called model-free world of PG methods.

[150] Szita and Lörincz, "Learning Tetris using the noisy cross-entropy method," 2006

[90] Marco, Hennig, Bohg, Schaal, and Trimpe, "Automatic LQR Tuning Based on Gaussian Process Global Optimization," 2016

[169] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992

[67] Jie and Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," 2010

[170] Wilson, Fern, and Tadepalli, "Using trajectory data to improve Bayesian optimization for reinforcement learning," 2014

Value-Based RL

Instead of modeling the expected return for an initial state distribution $p(s_0)$, the third branch of RL algorithms builds upon models of the expected value for a specific state or state-action-tuple. These algorithms model the state or state-action value function. Similar considerations about the specification of model assumptions apply in this domain as well. However, it is much more involved to specify meaningful model-assumptions in this space. Assumptions for value-functions would need to include the long-term interplay of policy, system dynamics and reward function for arbitrary query points. Thus, most state-of-the-art value-function methods revert back to very flexible, but thus data-inefficient function approximators, such as tables (in discrete state/action spaces) [164] or neural networks (for large-scale, continuous problems) [98, 156]. Interestingly the relation between structure and model-assumptions in the system dynamics, cost, or policy space to structure and assumptions in the value function domain is largely unexploited by current state-of-the-art methods.

[164] Watkins and Dayan, "Q-learning," 1992

[98] Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller, "Playing atari with deep reinforcement learning," 2013

[156] Van Hasselt, Guez, and Silver, "Deep reinforcement learning with double q-learning," 2016

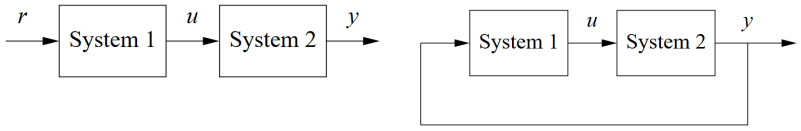
2.3 Reinforcement Learning and Control

This work builds heavily on concepts and methods in the field of model-based RL and model learning. Despite the renewed interest in these methods in the RL community, in particular associated with the recent surge of model-based deep RL, both fields are actually much more established in the control community. With finite-horizon optimal control [84] and system identification [88], two well established fields of research exist in the control community which capture very much similar problems. In this section, we try to briefly highlight the similarities but also the differences between the communities to enable a shared appreciation of existing but also novel contributions from both control and RL community.

[84] Lewis, Vrabie, and Syrmos, *Optimal control*, 2012

[88] Ljung, "System identification," 1998

Control is an engineering discipline concerned with the behavior of *dynamical systems* and methods to influence their behavior by choosing inputs to the systems appropriately. A dynamical systems is characterized by behavior which changes over time and is possibly influenced by external inputs. Typically, multiple dynamical systems are combined by feeding some output of one system to the input of the second system. This relation is depicted in Fig. 6. In cases where the output of the second system influences the input of the first system, a so called closed-loop is established. In contrast, if no connection exists, an open-loop system is formed.



(a) Open-loop system. (b) Closed-loop system.

Figure 6: Typical arrangement of two dynamical systems in control. (a) Open loop system: a reference r influences the first system, which in turn influences the output of the second system y by choosing an input u. (b) In a feedback arrangement, the output of the second system y additionally influences the input to the first system. Figure from [9].

In control, typically, the first system is a man-made controller which is designed to influence the behavior of some existing (physical) system. This arrangement is called feedback control, since changes in the behavior of the second system will be feedback into the controller (system 1) and thus influence the future control behavior in a circular pattern. This feedback loop is structurally identical to the typical RL learning loop depicted earlier in Sec. 2.1, where an agent with some policy acts to influence a (unknown) system.

The study of dynamical systems and control theory predates much of the work in RL. It is thus worthwhile to understand both fields and their perspectives on the problem. The following is an attempt to contrast the commonly used problem formulation in RL with the one from control. This is to get a better understanding of the respective nomenclature in both fields and make the similarities in the underlying problems apparent. It is remarked that different problem settings and formulations exist in both fields, which extend these basic prob-

lem setting. For example, the presented modeling schemes in Sec. 7 and Sec. 8 is based on previous work in auto-regressive and state-space models as known from the system identification community [17, 109].

[17] Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*, 2013

[109] Nelles, *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*, 2013

The field of control is concerned with analyzing properties of dynamical system, such as observability, reachability, or stability. To make statements about these properties, models of the system need to be available. Much of control is thus based on physical, first-principle models, which derive the behavior of dynamical systems as ordinary differential equations (ODEs). This perspective usually results in continuous time models in the form of

$$\frac{dx}{dt} = f(x,u), \quad y = h(x,u)\,. \tag{8}$$

In this equation, the state of a system is characterized by a vector x of state variables. The input to the system is similarly defined by a vector of input variables u. The continuous change in the system state can be described by a first-order ODE given the function $f(\cdot,\cdot)$. Finally, the actually observed quantities y are given based on some observation mapping $h(\cdot,\cdot)$. This model is called continuous-time, state-space model, since the current state of the system is fully captured in the state vector x. Despite the time continuous formulation, time discrete descriptions of a system can be derived given appropriate integration techniques, which results in

$$x_{t+1} = f_d(x_t,u_t), \quad y_t = h(x_t,u_t)\,. \tag{9}$$

with a time-discrete description of the system dynamics $f_d(\cdot,\cdot)$. From (9) it is straight forward to find the parallels to the POMDP system description in RL as defined in Sec. 2.1.

Not only the system description, but also the objective in classical control is very comparable to the one in RL. One branch of control deals with problems, where minimal cost should be incurred when operating a system over some time horizon T. This branch is called optimal control and the main objective is given by

$$J = \int_0^T \mathcal{L}(x(t),u(t),t]dt \tag{10}$$

$$s.t.$$

$$\dot{x}(t) = f(x(t),u(t))\,. \tag{11}$$

Again, the resemblance to the RL objective in 2.1 is apparent.

Despite the strong similarities between RL and control problem setting and objective, both fields tend to operate on different assumptions and thus emphasis different parts of the problem.

SYSTEM KNOWLEDGE - In RL, the system is typically unknown to the agent. The task of the agent is thus to explore the system sufficiently well to learn about how to operate the system and maximize its reward. This perspective results in the RL typical exploration-exploitation trade-off. The agent, lacking prior information about the system, needs to explore to solve the task, but at the same time needs to exploit its knowledge to choose optimal actions. If models of the system are incorporated in RL, these models are typically learned from data and tend to incorporate some probabilistic formulation to capture uncertainty about the true nature of the system and stochasticity that might be inherent to the system. In contrast, models in control are typically derived prior to the design of a controller. Despite the broad range of model types, ranging from white, gray, to black-box models, typically simplified first-principle-based models are utilized for control design.

OBJECTIVE - Control theory developed a broad range of tools to analyze and understand the behavior of a dynamical system. Objectives such as observability, reachability and stability can thus be analyzed using classical control theory, which are not directly accessible by the accumulated reward objective in RL. There is a trade-off between the optimization of an behavior for a specific situation (or range of situation) as in RL or optimal control and understanding fundamental properties of a system, given the assumption of an simplified, approximate model. RL might claim to solve the underlying problem by optimizing the *true* objective. However, specifying the real objective is a hard problem for real systems, leading to sub-problems like reward-shaping and research fields like inverse RL and learning from demonstration.

2.4 Gaussian Processes

A Gaussian Process (GP) [168] is a distribution over functions $f : \mathbb{R}^D \to \mathbb{R}$ that is fully defined by a mean function $m(\cdot)$ and covariance function $k(\cdot,\cdot)$. For each finite set of points $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ from the function's domain, the corresponding function evaluations $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)]$ are jointly Gaussian as given by

[168] Williams and Rasmussen, *Gaussian processes for machine learning*, 2005

$$p(\boldsymbol{f} \mid \boldsymbol{X}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{m}_X, \boldsymbol{K}_{X,X}), \tag{12}$$

with mean vector $\boldsymbol{m}_X$ having elements $m_i = m(\boldsymbol{x}_i)$ and covariance matrix $\boldsymbol{K}_{X,X}$ with entries $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Given observed function

values $\boldsymbol{f}$ at input locations $\boldsymbol{X}$, the GP predictive distribution at a new input location $\boldsymbol{x}^*$ is obtained as the conditional distribution

$$p(f^* \mid \boldsymbol{x}^*, \boldsymbol{f}, \boldsymbol{X}) = \mathcal{N}(f^* \mid \mu, \sigma^2), \tag{13}$$

with posterior mean and variance

$$\mu = m_{x^*} + \boldsymbol{k}_{x^*,X} \boldsymbol{K}_{X,X}^{-1} (\boldsymbol{f} - \boldsymbol{m}_X)\,, \tag{14}$$

$$\sigma^2 = k_{x^*,x^*} - \boldsymbol{k}_{x^*,X} \boldsymbol{K}_{X,X}^{-1} \boldsymbol{k}_{X,x^*}\,, \tag{15}$$

where $k_{A,B}$ denotes the scalar or vector of covariances for each pair of elements in $\boldsymbol{A}$ and $\boldsymbol{B}$. In this work, the squared exponential kernel with Automatic Relevance Determination (ARD) [168] with hyperparameters θ_{GP} is employed.

[168] Williams and Rasmussen, *Gaussian processes for machine learning*, 2005

Commonly, the GP prediction in (13) is obtained by conditioning on all training data $\boldsymbol{X}$, $\boldsymbol{y}$. To alleviate the computational cost, several sparse approximations have been presented [142]. By introducing P inducing GP targets $\boldsymbol{z} = [z_1, \ldots, z_P]$ at pseudo input points $\boldsymbol{\zeta} = [\boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_P]$, which are jointly Gaussian with the latent function f, the true GP predictive distribution is approximated by conditioning only on this set of inducing points,

[142] Snelson and Ghahramani, "Sparse Gaussian processes using pseudo-inputs," 2006

$$p(f^* \mid \boldsymbol{x}^*, \boldsymbol{f}, \boldsymbol{X}) \approx p(f^* \mid \boldsymbol{x}^*, \boldsymbol{z}, \boldsymbol{\zeta})\,, \tag{16}$$

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{m}_\zeta, \boldsymbol{K}_{\zeta,\zeta})\,. \tag{17}$$

The predicted function values consequently become mutually independent given the inducing points.

Part I

Model-Based Reinforcement Learning for Tuning of PID Controllers

3

Introduction

Studies about feedback control in the refining, chemicals, and pulp and paper industries reveled that 97% of the deployed controllers involved some sort of PID feedback control scheme [38]. Even though more complex control concepts, such as Model Predictive Control (MPC) are increasingly adopted in the industry [123], PID is clearly the predominant control algorithm in low-level, regulatory controllers.

[38] Desborough and Miller, "Increasing customer value of industrial control performance monitoring—Honeywell's experience," 2001

[123] Qin and Badgwell, "An overview of industrial model predictive control technology," 1997

Despite the low number of open parameters in PID control, tuning multiple coupled controllers in multi-input multi-output (MIMO) systems, can become tedious in practice. The goal in this first part is to leverage the benefits of Reinforcement Learning for fast and automatic tuning of PID controllers. In this work, we extend Probabilistic Inference for Learning COntrol (PILCO) [31], a framework for iterative improvement of controllers based on the expected finite horizon cost as predicted from a learned system model. This model is learned in a fully Bayesian setting using Gaussian Process Regression (GPR) as a non-parametric function estimator [76]. Gaussian Processes (GPs) as probability distributions over a function space provide a prediction and an uncertainty measure of the process dynamics. The uncertainty component can be utilized to implement cautious control [106] or trigger further exploration to improve the model knowledge [105]. As controller tuning is based on the full non-linear system model, this framework takes into account all process couplings for controller tuning. In contrast to some of the related PID tuning techniques, which are most commonly limited to SISO or multi-loop problems, this framework imposes no structural restrictions on the multivariate PID control design.

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[106] Murray-Smith, Sbarbaro, Rasmussen, and Girard, "Adaptive, cautious, predictive control with Gaussian process priors," 2003

[105] Murray-Smith and Sbarbaro, "Nonlinear adaptive control using non-parametric Gaussian process prior models," 2002

PILCO has been successfully applied to reinforcement learning tasks, such as the inverted pendulum swing-up [31], the control of low-cost manipulators [35], and some real-world applications like throttle valve control [18]. These examples focus mostly on non-linear state feedback policies. In industrial applications, however, it is desirable to obtain interpretable control designs, which is the case for PID control structures rather than for arbitrarily complex non-linear control designs.

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

[18] Bischoff, Nguyen-Tuong, Koller, Markert, and Knoll, "Learning Throttle Valve Control Using Policy Search," 2013

The main portion of the presentation in this part of the thesis has been previously published as

> A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, and S. Trimpe. "Model-based Policy Search for Automatic Tuning of Multivari-

ate PID Controllers." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2017

Additional experimental results from applying PILCO-PID to the problem of learning control of an model race car are based on work conducted in a master thesis project, which has been supervised in the context of this PhD research. The master thesis work on PILCO-PID for gain-scheduled PID control has been previously published in

M. Lefarov. "Model-Based policy Search for Learning Multivariate PID Gain Scheduling Control." MA thesis. University of Stuttgart, Apr. 2018

3.1 Contributions

In this part, we propose PILCO-PID as a novel, general framework for automatized and data-efficient tuning of multivariate PID controllers. In particular, the main benefits of the developed PILCO-PID framework are

- Applicability to non-linear MIMO systems with arbitrary couplings.
- Tuning of arbitrary multivariate PID structures.
- Consistent treatment of model uncertainty for PID tuning without prior system knowledge.
- No process model required.

The auto-tuning method is demonstrated in pole balancing experiments on Apollo, a complex robot platform as shown in Fig. 7. Extensions to GP-NARX system models are demonstrated to cope with typical problems on real systems, such as imperfect low-level tracking controllers and unobserved dynamics. As a second, real-world example, the PILCO-PID algorithm is demonstrated to efficiently learn to control a model race car, as shown in Fig. 8, close to its friction limits.

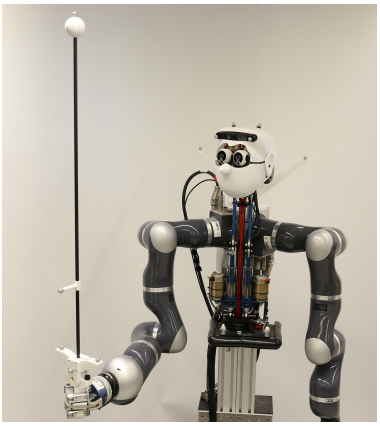


Figure 7: The humanoid upper-body robot, Apollo, is balancing an inverted pendulum. Using the proposed framework, coupled PID and PD controllers are trained to stabilize the pole in the central, upright position without requiring prior knowledge of the pendulum system dynamics.

Figure 8: The ability of PILCO-PID to automatically tune strongly coupled PID controllers on a highly non-linear system is demonstrated by learning to drive at the friction limits of a model race car.

3.2 Outline

This part starts by introducing the two fundamental building blocks of PILCO-PID: model-based reinforcement learning algorithms in Sec. 3.3 and PID control in Sec. 3.4. The main ideas and flavors of model-based reinforcement learning, related work and its relevance for data-efficient learning are presented in Sec. 3.3. This is an extension to the discussion on classes of RL methods in Sec. 2.2. PILCO as one specific representative of this type of data-efficient, model-based RL algorithm is detailed in Sec. 3.3.1. Section 3.4 is a primer on PID

control and tuning methods. Related work from the field of classical control in tuning PID control is discussed in Sec. 3.4.2.

The proposed multivariate PILCO-PID framework is developed in Sec. 4. Section 4.3 presents the results of applying the framework for tuning coupled PID controllers on the Apollo robot and on a model race car. Finally, this part is concluded with remarks and propositions for future work in Sec. 5.

3.3 Model-Based Reinforcement Learning

The following section is a short summary of directions and related work in the Model-Based Reinforcement Learning (MBRL) realm. In MBRL, we use all available interaction data (s_t, a_t, r_t, s_{t+1}), to learn a model $\hat{p}(s_{t+1} \mid s_t, a_t)$ of the true, underlying system dynamics $p(s_{t+1} \mid s_t, a_t)$. This model can then be utilized together with the policy $\pi(a_t \mid s_t; \theta)$ to simulate the system behavior over an horizon T. This simulation is started at a deterministic state s_0 or from a sample from the initial state distribution $s_0 \sim p(s_0)$. With these components, we can sample trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_t)$ by $a_t \sim \pi(a \mid s_t; \theta)$ and $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$. Alternatively, the full distribution of future states and actions can be computed as

$$p(s_{0:T}, a_{0:T}) = p(s_0) \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t) \pi(a_t \mid s_t; \theta) \,. \tag{18}$$

Model-based methods are currently gaining interest in the RL community due to empirical evidence of data efficiency [69, 99], [147, Ch. 8] over the predominant value or policy-gradient based RL methods (cf. Sec. 2.2). There are some clear advantages of constructing a model of the system dynamics to learn from the interaction data. All available data can be integrated into the model, no matter if data was collected on- or off-policy, i.e., with the current policy or any other policy. Similarly, data across multiple tasks can be incorporated as long as the underlying system remains unmodified. This is in contrast to policy gradient or value function methods, where additional effort is required to integrate off-policy data or data from a different tasks. Furthermore, much of domain-specific prior knowledge usually exist in the form of physical, i.e., first-principle dynamics models and can potentially be integrated and combined with data-based models. The model allows us to gain interpretable insights into future system behavior and is thus much more accessible to a human operator, e.g., in comparison to a value function estimate. Finally, but maybe most importantly, many physical systems exhibit great smoothness properties, thus enabling strong generalization in model-space given only a few data-points.

[69] Kaiser, Babaeizadeh, Milos, Osinski, Campbell, Czechowski, Erhan, Finn, Kozakowski, and Levine, "Model-based reinforcement learning for Atari," 2019

[99] Moerland, Broekens, and Jonker, "Model-based reinforcement learning: A survey," 2020

[147] Sutton and Barto, *Reinforcement learning: An introduction*, 2018

Algorithm 1: Schematic Depiction of Model-Based Reinforcement Learning

Input: Initial policy parameters θ_0
Empty data buffer $\mathcal{D}$
repeat
  Execute policy π_θ
  Update data buffer $\mathcal{D}$ with new trajectory τ
  Learn dynamics model from data $\mathcal{D}$
  repeat
    Simulate system with dynamics model and policy
    Compute expected return $J(\pi)$
    Improve policy π
  until Simulated performance converged (or max. iterations)
until Real performance converged (or max. iterations)

Despite these benefits of model-based RL, several problems limit the common adoption of model-based methods so far. Complex systems are typically much harder to model solely from observed data. Some of these problems will become apparent in the experiments conducted in this part. Solutions for improved model-learning techniques will be presented in Part II of this work together with a discussion on typical model learning obstacles in real-world systems in Sec. 6.3.

A schematic depiction of a typical MBRL algorithm is presented in Alg. 1. In most MBRL algorithms, there exist an outer loop, where we interact with the system to gather new data and where we improve the system dynamics model. In the inner loop, we subsequently improve the policy based on the available model- and data-knowledge. This scheme is then iterated till convergence. Based on this general scheme, many different MBRL algorithms have been established. The main differentiating factors are a) what kind of model is employed, b) how this model is employed and c) how a policy update is computed.

MODEL TYPE Many model-based methods in control and RL rely on deterministic models, i.e. $s_{t+1} = f(s_t, a_t)$. Additionally, most state-of-the-art methods involve probabilistic models to capture stochasticity in the system and uncertainty about the precise nature of the model. Other degrees of freedom in the modeling are given by the model structure, model prediction and model training. Many model architectures, ranging from local linear models [61], Gaussian Processes (GPs) [35], or Neural Networks (NN) [54], have been explored in the literature. These models can predict single steps ahead, e.g. s_{t+1} or capture full future trajectories $s_{t+1:t+N}$ at once. Finally, many training schemes and objectives have been derived to fit the model parameters optimally to the available data. Options are one- or multi-step predictions to minimize some sort of prediction error, e.g., least squares or maximum likelihood. The Gaussian Process regression formalism as

[61] Gu, Lillicrap, Sutskever, and Levine, "Continuous deep q-learning with model-based acceleration," 2016

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

[54] Gal, McAllister, and Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," 2016

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

deployed in the proposed PILCO-PID framework is a fully Bayesian scheme to account for uncertainties due to complex system dynamics or missing data. It is most commonly trained for one-step-ahead predictions [76] however extensions of these models to long-term predictions and latent-space models are presented later in this work, as well as in [162].

MODEL UTILIZATION A second perspective on MBRL originates from the way the model is employed to predict future system behavior. In particular, differences are based on how much the model is predicting into the future and how probabilistic models and their predictive distributions are computed. In particular non-linear, probabilistic models cause highly complex predictive distributions. Only for very simplified systems, e.g., linear system models and Gaussian noise, long term predictive distributions maintain an analytically tractable form. Both the model structure and policy structure might cause more complex distributions, which in turn requires approximation. Typically methods are based on linearization [57], moment matching [22], sampling [107] or numerical integration [160].

POLICY UPDATE The third dimension of differences in model-based RL methods is along with the utilization of the model and the model-predictions in the policy update. In particular, some models are used as a black-box simulator, whereas others are employed as a differentiable dynamics model. With the black box simulator, interactions with the model are possible very similar to the interaction with the real environment. This is, trajectory sequences can be generated for policies. Thus, any standard RL method can be deployed on this model-based rollout data, whichh is sometimes called *hallucinated* data. Also, combinations of real and hallucinated data can be incorporated in different parts of the MBRL method to achieve a trade-off between model and real system. The Dyna algorithm is one of the earliest examples of utilizing a learned world model to generate data, which is then used in another model-free RL method [145]. Other examples can be found in [61], which utilizes data from local linear models to accelerate model-free RL, [46] where hallucinated data is used for value updates, or [78] where model-based trajectories are employed in trust-region policy gradient methods. Alternatively, models and model-predictions can potentially be made differentiable and thus enable the computation of gradients, i.e. $\delta s_{t+1}/\delta a_t$ or $\delta s_{t+1}/\delta s_t$. Instead of *estimating* the policy gradient with standard RL frameworks, these gradients can facilitate the direct *computation* of the policy gradient. Since gradient information is propagated through the full rollout horizon, this method is called Backpropagation Through Time (BPTT). Somewhat orthogonal, instead of manually designing how to utilize the model within

[162] Wang, Hertzmann, and Fleet, "Gaussian process dynamical models," 2005

[57] Girard and Rasmussen, "Multiple-step ahead prediction for non linear dynamic systems–a gaussian process treatment with propagation of the uncertainty,"

[22] Candela, Girard, Larsen, and Rasmussen, "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting," 2003

[107] Nagabandi, Kahn, Fearing, and Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," 2018

[160] Vinogradska, Bischoff, Achterhold, Koller, and Peters, "Numerical quadrature for probabilistic policy search," 2018

[145] Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," 1990

[61] Gu, Lillicrap, Sutskever, and Levine, "Continuous deep q-learning with model-based acceleration," 2016

[46] Feinberg, Wan, Stoica, Jordan, Gonzalez, and Levine, "Model-based value estimation for efficient model-free reinforcement learning," 2018

[78] Kurutach, Clavera, Duan, Tamar, and Abbeel, "Model-ensemble trust-region policy optimization," 2018

[113] Pascanu, Li, Vinyals, Heess, Buesing, Racanière, Reichert, Weber, Wierstra, and Battaglia, "Learning model-based planning from scratch," 2017

[126] Racanière, Weber, Reichert, Buesing, Guez, Jimenez Rezende, Puigdomènech Badia, Vinyals, Heess, and Li, "Imagination-augmented agents for deep reinforcement learning," 2017

[80] Langlois, Zhang, Zhang, Abbeel, and Ba, "Benchmarking model-based reinforcement learning," 2019

an algorithm, the agent itself can learn how to use model predictions within the policy, as demonstrated in [113] and [126].

An attempt to benchmark current state-of-the-art MBRL methods has been published by [80]. A full survey on current MBRL methods, which reviews model learning techniques and model-based planing can be found in [99].

[99] Moerland, Broekens, and Jonker, "Model-based reinforcement learning: A survey," 2020

3.3.1 PILCO

In this work, we will focus on the PILCO framework [35]. For PILCO, the model is a probabilistic, Gaussian Process model (cf. 2.4), which allows for a Bayesian treatment of model uncertainty to capture the one-step-ahead predictive distribution [76]. A moment matching approximation allows to analytically compute the Gaussian marginal distribution for each predicted state, such as to match the moments of the true, non-Gaussian, predictive distribution. Thus, a multi-step predictive distribution is available [58]. Due to the nature of the Gaussian Process and the moment matching approximation, BPTT can be computed in closed form. This allows for efficient policy updates using gradient-based optimization.

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[58] Girard, Rasmussen, Quinonero-Candela, Murray-Smith, Winther, and Larsen, "Multiple-step ahead prediction for non linear dynamic systems—a Gaussian process treatment with propagation of the uncertainty," 2003

For the PILCO-PID framework, we consider discrete time dynamic systems of the form

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \epsilon_t \,, \tag{19}$$

with continuously valued state $\mathbf{x}_t \in \mathbb{R}^D$ and input $\mathbf{u}_t \in \mathbb{R}^F$. The system dynamics f is not known a priori. We assume a fully measurable state, which is corrupted by zero-mean independent and identically distributed (i.i.d.) Gaussian noise, i.e. $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \Sigma_\epsilon)$.

The specific RL formulation in PILCO aims at minimizing the expected cost-to-go given by

$$J = \sum_{t=0}^{T} \mathbb{E}[c(\mathbf{x}_t, \mathbf{u}_t; t)], \quad \mathbf{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0) \,, \tag{20}$$

where an immediate, possibly time dependent cost $c(\mathbf{x}_t, \mathbf{u}_t; t)$ penalizes undesired system behavior. As a policy search method, the best out of a range of policies $\mathbf{u}_t = \pi(\mathbf{x}_t; \theta)$ parametrized by θ. Particularly in *model-based* policy search frameworks, a model $\hat{f}$ of the system dynamics f is utilized to predict the system behavior and to optimize the policy.

PILCO, as a specific model-based policy search framework emphasizes data-efficiency and consistent handling of uncertainty when constructing the system dynamics model $\hat{f}$. To incorporate all available data from policy rollouts (experiments) on the actual system, a Gaussian Process (GP) [168] is utilized as a non-parametric, probabilistic

[168] Williams and Rasmussen, *Gaussian processes for machine learning*, 2005

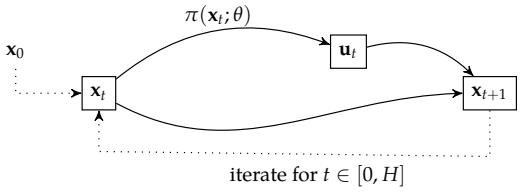


Figure 9: Computations for one time step in the long-term prediction. Black: original PILCO state propagation steps. Red: Augmented state propagation to accommodate PID policy optimization.

Algorithm 2: PILCO

Experiment: Execute random policy;
  record $\{\mathbf{x}_t, \mathbf{u}_t\}_{t=1...T}$
Train initial GP dynamics model $\mathbf{x}_{t+1} = \hat{f}(\mathbf{x}_t, \mathbf{u}_t)$
repeat
  repeat
    Simulation: Predict $J(\theta)$ given $\hat{f}$, $\pi(\mathbf{x}_t; \theta)$
    Analytically compute gradient $dJ(\theta)/d\theta$
    Gradient-based policy update (e.g. CG, BFGS)
  until convergence: $\theta^* = \arg\min J(\theta)$
  Experiment: Execute $\pi(\mathbf{x}_t; \theta^*)$; record $\{\mathbf{x}_t, \mathbf{u}_t\}_{t=1...T}$
  Update dynamics model $\hat{f}$ using all recorded data
until task learned
return $\pi^*(\mathbf{x}_t; \theta^*)$

model.

The PILCO framework is outlined in Alg. 2. In the inner loop, a simulated rollout is conducted based on the dynamics model $\hat{f}$ and the current policy $\pi(\mathbf{x}_t; \theta)$. The system's state $\mathbf{x}_t$ is propagated over a finite prediction horizon H starting at the system's initial state $\mathbf{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ as visualized for one time step in Fig. 9. The posterior distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ is approximated in each time step by a Gaussian distribution using moment matching [22]. Therefore, we obtain a Gaussian marginal distribution of the long-term predictions $p(\mathbf{x}_0), \ldots, p(\mathbf{x}_H)$. The expected long-term cost (20) of this rollout as well as its gradients with respect to the policy parameters θ can be computed analytically. Policy optimization can then be conducted based on this prediction method using standard gradient-based optimization techniques.

[22] Candela, Girard, Larsen, and Rasmussen, "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting," 2003

Starting with an initial random policy[1], the algorithm optimizes the cost-to-go by repeatedly executing the policy on the system, thereby gathering new data and building the dynamics model, subsequently improving the policy iteratively, until the task has been learned.

[1] A random policy might be a parametrized policy with randomly chosen parameters (cf. [32]) or a random input signal exciting the system to generate initial dynamics data.

3.4 PID Control

Proportional, Integral and Derivative (PID) control structures are still the main control tool being used in industrial applications, in partic-

[25] Cominos and Munro, "PID controllers: Recent tuning methods and design to specification," 2002

[66] Jiang and Gao, "An application of nonlinear PID control to a class of truck ABS problems," 2001

[137] Siciliano, Sciavicco, Villani, and Oriolo, *Robotics: modelling, planning and control*, 2010

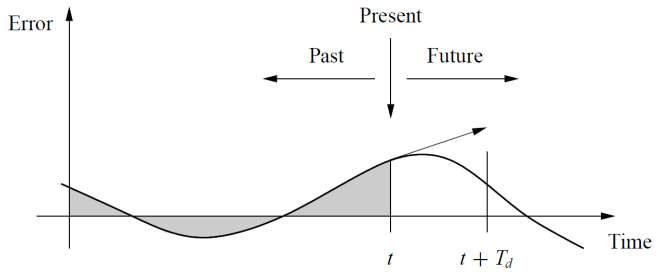


Figure 10: Visualization of the contributions to a PID control signal. In each time-step t, the PID control signal is comprising a linear combination of the accumulated past error, the current error and the predicted future error. Figure from [9]

ular in the process industry [25], but also in automotive applications [66] and in low-level control in robotics [137]. The large share of PID controlled applications is mainly due to the past record of success, the wide availability, and the simplicity in use of this technique. Even in multivariable systems, PID controllers can often be employed [68].

[68] Johnson and Moradi, *PID control - New Identification and Design Methods*, 2005

3.4.1 Definition of the PID Control Law

One of the goals in control is to make a system keep a desired set-point or follow a desired reference trajectory. This is, the output of the system y should match a reference r as closely as possible. The task of a controller is to select suitable inputs u to the system such as to minimize the control error $e = r - y$. One simple feedback control mechanism is given the On-Off controller [2] described by

$$u = \begin{cases} u_{max} & \text{if } e > 0 \\ u_{min} & \text{if } e < 0 \\ 0 & \text{otherwise}\,. \end{cases} \tag{21}$$

[2] Also known as 2 step or Bang-Bang control.

This controller utilizes the maximal/minimal control signal u_{max} and u_{min} to compensate for any non-zero error. The large, discontinuous switch in action can lead to overreactions and oscillatory behavior. Instead, *proportional control* reacts by choosing a control input according to the size of the error as given by

$$u = \begin{cases} u_{max} & \text{if } e \geq e_{max} \\ K_p e & \text{if } e_{min} < e \geq e_{max} \\ u_{min} & \text{if } e \leq e_{min}\,. \end{cases} \tag{22}$$

In a error-band in between e_{min} and e_{max}, the controller behaves linearly, according to the controller gain K_p.

Since the proportional control input contribution might not be sufficient to reach a desired reference, integral control can be adopted to

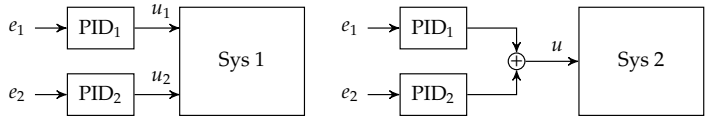


Figure 11: Possible PID structures, exemplified with two controllers. Left: Individual PID controllers acting on different system inputs. Right: Combination of PID controllers acting on the same system input.

accumulate larger control contributions if the error is not diminishing.

$$u = K_i \int_0^t e(\tau) d\tau \tag{23}$$

In integral control input is proportional to the accumulated error, weighted with the integral gain K_i. This control can be shown to result in zero steady-state error, in cases where a non-oscillating stead-state is reached.

Finally, derivative control is based on a prediction of future error. In the simplest instance, a linear extrapolation based on the current error derivative is taken into account.

$$u(t) = K_d \frac{de(t)}{dt} \tag{24}$$

Combining these three control contributions, the proportional-integral-derivative (PID) controller is given by

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau) d\tau + K_\mathrm{d} \frac{de(t)}{dt} \tag{25}$$

A visualization of the individual error contributions in the PID control law is shown in Fig. 10.

The PID controller is agnostic to the system dynamics and depends only on the system's error. Each controller is parametrized by its proportional, integral and derivative gain ($\theta_\mathrm{PID} = (K_\mathrm{p}, K_\mathrm{i}, K_\mathrm{d})$). A general PID control structure $C(s)$ for MIMO processes (19) can be described in transfer function notation by a $D \times F$ transfer function matrix

$$C(s) = \begin{bmatrix} c_{11}(s) & \cdots & c_{1D}(s) \\ \vdots & \ddots & \vdots \\ c_{F1}(s) & \cdots & c_{FD}(s) \end{bmatrix}, \tag{26}$$

where s denotes the complex Laplace variable and $c_{ij}(s)$ are of PID type. The multivariate error is given by $\mathbf{e}_t = \mathbf{x}_{\mathrm{des},t} - \mathbf{x_t} \in \mathbb{R}^D$ such that the multivariate input becomes $\mathbf{u}(s) = C(s)\mathbf{e}(s)$. Examples for possible PID structures are shown in Fig. 11. A comprehensive overview on PID control is given in [8].

[8] Åström, Hägglund, and Astrom, *Advanced PID control*, 2006

3.4.2 Tuning of PID Control Parameters

In practice, calibration of PID controllers is still often achieved by tedious manual tuning or by heuristic tuning rules [111]. The following review strives to categorize research and methods in tuning PID controllers according to the respective algorithmic ideas and assumptions. For the purpose of this work and to identify the connections and distinctions with respect to the proposed PILCO-PID framework, three main categories of PID tuning methods will be discussed: feature-based, model-based and optimization based methods.

[111] O'Dwyer, *Handbook of PI and PID controller tuning rules*, 2009

Feature-Based PID Tuning

Feature-based methods are amongst the first commonly adopted PID tuning concepts [174] and helped to provide systematic ways of deriving PID parameters for a novel plant. In feature-based methods, characteristic properties ("features") of the closed loop system are directly measured on the system by executing specific excitation signals. Given a set of rules ("gain functions"), the measured features are then directly translated into PID parameters.

[174] Ziegler and Nichols, "Optimum settings for automatic controllers," 1942

One of the earliest examples for feature-based PID tuning are the heuristic Ziegler-Nichols (ZN) rules [174]. In this case, a limit ("ultimate") proportional gain K_u is measured together with the oscillation period T_u. These measures are obtained by increasing the proportional gain K_p in closed-loop control until the system starts to oscillate. Given K_u and T_u, the classic ZN tuning rules would set the PID parameters as

$$K_p = 0.6K_u, \quad K_i = 1.2K_u/T_u, \quad K_d = 0.075K_uT_u\,. \tag{27}$$

Similar tuning rules are available, which strive to achieve different shapes of the closed-loop response. For example, ZN rules for P, PI, PD, or different amounts of overshoot are published. Other tuning rules have been derived based on different features and with different premisses regarding closed-loop performance and characteristics. For example, the Approximate M-constraint Integral Gain Optimization (AMIGO) rules [7] require the identification of a velocity gain K_v. They are designed to maximize the integral gain K_i such as to achieve faster decay of steady-state errors.

[7] Åström and Hägglund, "Revisiting the Ziegler–Nichols step response method for PID control," 2004

Feature-based tuning rules are typically derived for SISO systems, although some extensions to MIMO systems exist [115]. Conceptually, the features can be understood as relatively simple to obtain parameters of a simplified, low-order model of the plant. In case of the time-domain ZN rules, a first-order plus dead-time model can be di-

[115] Perez and Herrero, "Extending the AMIGO PID tuning method to MIMO systems," 2012

rectly given from the measured static gain K, dominant time constant T, and dominant dead time L.

$$G_p(s) = \frac{K}{1+sT} e^{(-sL)} \tag{28}$$

The tuning rules can be interpreted as the result of optimizing closed-loop characteristics, such as disturbance rejection, steady-state error, or overshoot, assuming the true system to be well approximated by the low-order, identified model. This perspective directly leads to the second class of model-based PID tuning techniques.

Model-Based PID Tuning

In situations where the true plant behavior can be accurately described by low-order system dynamics models, control theory provides a large tool-chest to derive controllers with desired properties. Many of these methods directly carry over to the case of PID controllers. Methods such as pole placement can be directly employed to shape the close-loop behavior if sufficiently simple plant models (e.g. first or second order transfer function models) are available. Extensions for pole placements of the dominant pole in higher-order systems exist as well. Methods like Internal Model Control (IMC) utilize the plant model directly to compute the (higher-order) controller. For some simpler plant models (first-order, first-order plus dead-time), these control designs can be translated back into respective PID controllers [129].

[129] Rivera, Morari, and Skogestad, "Internal model control: PID controller design," 1986

Optimization-Based PID Tuning

The class of optimization-based techniques is concerned with optimizing the (in)finite horizon closed-loop response of a plant controlled by a given PID controller. An objective is set-up to characterize desired behavior and optimization schemes are deployed to alter the PID parameters accordingly. Optimization objectives vary widely from Integrated Absolute Error (IAE) [62] or Integrated Squared Error (ISE) for finite-horizon measurements, to H_∞ constraints on the resulting sensitivity or complementary sensitivity functions [55] for model-based, infinite-horizon performance.

[62] Hang, Åström, and Ho, "Refinements of the Ziegler–Nichols tuning formula," 1991

[55] Garpinger and Hägglund, "A software tool for robust PID design," 2008

Without model knowledge, tuning methods need to resort to optimization schemes to optimize the empirically measured performance as given by the objective/cost-function. Derivative free methods based on Evolutionary Strategies (ES) or Bayesian Optimization (BO) can directly optimize the PID gains to achieve optimal performance. Examples based on multi-crossover genetic algorithms [62] or particle swarm optimization [53] have been presented.

[53] Gaing, "A particle swarm optimization approach for optimum design of PID controller in AVR system," 2004

Extensions to MIMO Systems

PID literature typically distinguishes between tuning methods for *multi-loop* PID control and *multivariable* PID control systems. The former have a diagonal transfer function matrix $C(s)$ whereas the latter allows PID controllers on all elements of $C(s)$, i.e. all combinations of errors and inputs can be controlled, thus allowing additional cross-couplings.

More advanced tuning concepts are most frequently developed for Single-Input-Single-Output (SISO) systems [8] [16]. For Multi-Input-Multi-Output (MIMO) systems, popular tuning methods, such as biggest log-modulus and the dominant pole placement tuning method [68], strive to tune each control loop individually, followed by a collective de-tuning to stabilize the multi-loop system. These tuning methods, however, rely on linear process models and require stable processes. For general PID control structures where multiple controllers act on each input, controller design is usually conducted by decoupling the process, subsequently allowing the design of individual SISO PIDs. Examples are online adjusted precompensators, which decouple the process transfer function matrix [172].

[8] Åström, Hägglund, and Astrom, *Advanced PID control*, 2006

[16] Berner, Hägglund, and Åström, "Asymmetric relay autotuning - Practical features for industrial use," 2016

[68] Johnson and Moradi, *PID control - New Identification and Design Methods*, 2005

[172] Yamamoto and Shah, "Design and experimental evaluation of a multivariable self-tuning PID controller," 2004

Contributions by PILCO-PID

With this framework, we address the general class of *multivariable* PID control systems with no restrictions on the elements of $C(s)$. In contrast to the previously presented tuning frameworks, PILCO-PID does not rely on a simplified, e.g., low-order, approximation of the system dynamics. PILCO's Bayesian, non-parametric dynamics model, allows to flexibly represent the true non-linear system dynamics. At the same time, PILCO-PID accounts for uncertainty in the system model in a principled way. Strong connections exist to the third class of optimization-based methods. As discussed in the context of control and RL in Sec. 2.3, the RL framework shares close ties with the finite horizon optimal control setting. Again, in contrast to the aforementioned methods, PILCO-PID is able to exploit analytic gradients and back-propagation through time to much more efficiently find locally optimal solutions, in comparison to the derivative-free optimization methods presented above.

4 Learning Multivariate PID Control

One of the challenges in model-based RL and in particular in the PILCO framework is to efficiently compute the trajectory distribution $p(\tau \mid \theta)$ (cf. (4)). This problem arises from the probabilistic nature of the system model and policy as well as the non-linear system behavior, which in general causes complex probability distributions [30]. This problem further involves the computation of analytic gradients $\mathrm{d}\tau/\mathrm{d}\theta$ to allow for efficient, gradient-based policy optimization.

[30] Deisenroth and Mohamed, "Expectation propagation in Gaussian process dynamical systems," 2012

The following derivation of the PILCO-PID framework is based on two high-level insights. First, the state of the underlying system can be augmented by *artificial* state information, to represent the required PID input information, such as error or error derivatives. Second, the change in these artificial states can be described with a linear dynamics. Thus, closed form solutions for the propagation of Gaussian state uncertainty into future states is available and closed-form gradients can be given.

This section firstly introduces the augmented system state in Sec. 4.1. The PID controller is reformulated as a linear, state-feedback controller given the augmented state in Sec. 4.2. Utilizing the augmented state, extensions such as multi-task training or time-varying goals are derived in Sec. 4.2.3 and Sec. 4.2.4 respectively.

4.1 System State Augmentation

This section presents a sequence of operations to augment the current system state $\mathbf{x}_t$ into an fully augmented state $\tilde{\mathbf{z}}_t$. These augmentations are designed such as to facilitate any multivariable PID controller (26) in the form of a parametrized static state feedback law. In particular a static state feedback policy is given by

$$\mathbf{u}_t = \mathbf{K}_{\mathrm{PID}}(\theta)\tilde{\mathbf{z}}_t\,, \tag{29}$$

where $\mathbf{K}_{\mathrm{PID}}(\theta)$ is a matrix with parameters θ, such that the product of matrix and augmented state vector represent arbitrary PID control structure as in (26).

To deploy a PID controller on a computer system, typically, the continuous time elements of Eq. (25) need to be discretized. Therefore,

the error derivative is approximated by a finite difference and the integrated error is approximated by a sum of errors.

$$\dot{e}_t \approx \frac{e_t - e_{t-1}}{\Delta T}\,, \tag{30}$$

$$\int_0^{t\cdot\Delta T} e(\tau)d\tau \approx \Delta T \sum_{\tau=0}^{t} e_\tau\,, \tag{31}$$

where ΔT represents the system's sampling time. More advanced approximation schemes exist to mitigate, for example, strong noise contributions in the numerical derivative computation. Details about practical implementation of discrete-time PID controllers on computer systems can be found in [6].

[6] Åström and Hägglund, *PID controllers: theory, design, and tuning*, 1995

The discrete time PID controller is consequently given by

$$u_t = K_p e_t + K_i \Delta T \sum_{\tau=0}^{t} e_\tau + K_d \frac{e_t - e_{t-1}}{\Delta T}\,. \tag{32}$$

Most notably, the resulting, discrete-time PID controller is not stateless but requires information about the previous error and the so-far accumulated error. The physical system state $\mathbf{x}_t$ is thus augmented by the necessary error information to an augmented state $\mathbf{z}_t$ defined by

$$\mathbf{z}_t := (\mathbf{x}_t, \mathbf{e}_{t-1}, \Delta T \sum_{\tau=0}^{t-1} \mathbf{e}_\tau)\,. \tag{33}$$

For simplicity, we denote vectors as tuples $(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(n)})$, where $\mathbf{v}^{(i)}$ may be vectors themselves. In this case, $\mathbf{z}_t^{(1)}$ indicates the physical system state at time step t, whereas $\mathbf{z}_t^{(2)}$ would indicate the added control error, as memorized from the previous time step. Based on the augmented system state $\mathbf{z}_t$ further augmentations will be detailed in the following sections to compute all required inputs for the PID controller (32).

4.1.1 Desired Goal State

The PID controller is most commonly used to stabilize around a desired set-point or track a desired reference trajectory. In both cases, the current error is computed given some current desired state $\mathbf{x}_{\text{des},t}$. Instead of a deterministic set-point, a Gaussian goal state distribution with non-zero variance is utilized. The desired set-point or target trajectory state is given by $\mathbf{x}_{\text{des},t} \sim \mathcal{N}(\mu_{\text{des},t}, \Sigma_{\text{des},t})$. Drawing the desired state from a Gaussian distribution yields better generalization to unseen targets as discussed in [35] This external reference is inde-

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

pendent of the current state $\mathbf{z}_t$. Thus, the joint distribution of the first, augmented state is given by

$$\begin{bmatrix} \mathbf{z}_t \\ \mathbf{x}_{\text{des},t} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_z \\ \mu_{\text{des},t} \end{bmatrix}, \begin{bmatrix} \Sigma_z & \mathbf{0} \\ \mathbf{0} & \Sigma_{\text{des},t} \end{bmatrix} \right) . \tag{34}$$

4.1.2 Error States

The current error is a linear function of $\mathbf{z}_t$ and $\mathbf{x}_{\text{des},t}$. Similarly, the approximated error derivative and integrated error can be expressed as linear operations on the so far computed state quantities.

$$\mathbf{e}_t = \mathbf{x}_{\text{des},t} - \mathbf{z}_t^{(1)} \tag{35}$$

$$\frac{\mathbf{e}_t - \mathbf{e}_{t-1}}{\Delta T} = \frac{1}{\Delta T}\mathbf{e}_t - \frac{1}{\Delta T}\mathbf{z}_t^{(2)} \tag{36}$$

$$\Delta T \sum_{\tau=0}^{t} \mathbf{e}_\tau = \mathbf{z}_t^{(3)} + \Delta T \mathbf{e}_t \, . \tag{37}$$

As discussed earlier, the finite difference approximation of the derivative error is prone to measurement noise. Yet, this framework can readily be extended to incorporate a low-pass filtered error derivative, which we omit for notational simplicity. In this case, additional historic error states would be added to the state $\mathbf{z}_t$ to provide the input for a low-pass Finite Impulse Response (FIR) filter.

Combining the previously introduced augmentations, we arrive at the fully augmented state $\tilde{\mathbf{z}}_t$ as given by

$$\mathbf{z}_t \xrightarrow{(34)} [\mathbf{z}_t, \, \mathbf{x}_{\text{des},t}] \tag{38}$$

$$\xrightarrow{(35)} [\mathbf{z}_t, \, \mathbf{x}_{des,t}, \, \mathbf{e}_t] \tag{39}$$

$$\xrightarrow{(36),(37)} [\mathbf{z}_t, \, \mathbf{x}_{des,t}, \, \mathbf{e}_t, \, \Delta T \sum_{\tau=0}^{t} \mathbf{e}(\tau), \, \frac{\mathbf{e}_t - \mathbf{e}_{t-1}}{\Delta T}] = \tilde{\mathbf{z}}_t \, . \tag{40}$$

Since all augmentations (34), (36), (35), (37) are linear transformations, the resulting augmented state distribution remains Gaussian for a Gaussian distributed state $\mathbf{z}_t$. Details about linear operations on Gaussian random variables are summarized in Appendix A.

Notice that state, desired state and error state do not necessarily need to have the same dimensionality. The desired state is reused in the cost function evaluation and can be utilized to penalize states that are not needed for the control policy/error computation. The error state needs to be a subset of the desired state which is itself a subset of the system state.

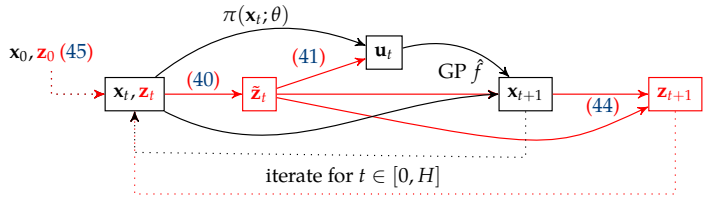


Figure 12: Computations for one time step in the long-term prediction. Black: original PILCO state propagation steps. Red: Augmented state propagation to accommodate PID policy optimization.

4.2 PID as Static State Feedback

Based on the final augmented state $\tilde{\mathbf{z}}_t$, the PID control policy for multivariate controllers can be expressed as a static state feedback policy:

$$\begin{aligned}\mathbf{u}_t &= \mathbf{K}_{\text{PID}}(\tilde{\mathbf{z}}_t^{(3)}, \tilde{\mathbf{z}}_t^{(4)}, \tilde{\mathbf{z}}_t^{(5)}) \\ &= \mathbf{K}_{\text{PID}} \left(\mathbf{e}_t, \Delta T \sum_{\tau=0}^{t} \mathbf{e}_t, \frac{\mathbf{e}_t - \mathbf{e}_{t-1}}{\Delta T} \right) ,\end{aligned} \tag{41}$$

where $\tilde{\mathbf{z}}_t^{(i)}$ indicates the i-th term of (40). The specific structure of the multivariate PID control law is defined by the parameters in $\mathbf{A}_{\text{PID}}$. For example, PID structures as shown in Fig. 11 would be represented by

$$\mathbf{K}_{\text{left}} = \begin{bmatrix} K_{\text{p},1} & 0 & K_{\text{i},1} & 0 & K_{\text{d},1} & 0 \\ 0 & K_{\text{p},2} & 0 & K_{\text{i},2} & 0 & K_{\text{d},2} \end{bmatrix} , \tag{42}$$

$$\mathbf{K}_{\text{right}} = \begin{bmatrix} K_{\text{p},1} & K_{\text{p},2} & K_{\text{i},1} & K_{\text{i},2} & K_{\text{d},1} & K_{\text{d},2} \end{bmatrix} . \tag{43}$$

4.2.1 State Propagation

A visualization of the state augmentation integrated into the one-step-ahead prediction is shown in red in Fig. 12 in comparison with the standard PILCO setting (in black). Given a Gaussian distributed initial state $\mathbf{x}_0$, the resulting predicted states will remain Gaussian for the presented augmentations.

Given the Gaussian distributed state and control input as derived in Sec. 4.1 and Sec. 4.2, the next system state is computed using the GP dynamics model $\hat{f}$. PILCO approximates the predictive distribution $p(\mathbf{x}_{t+1})$ by a Gaussian distribution using exact moment matching. From the dynamics model output $\mathbf{x}_{t+1}$ and the current error stored in $\tilde{\mathbf{z}}_t$, the next state is obtained as

$$\mathbf{z}_{t+1} = (\mathbf{x}_{t+1}, \tilde{\mathbf{z}}_t^{(3)}, \tilde{\mathbf{z}}_t^{(4)}) = (\mathbf{x}_{t+1}, \mathbf{e}_t, \Delta T \sum_{\tau=0}^{t} \mathbf{e}_\tau) . \tag{44}$$

Iterating (33) to (44), the long-term prediction can be computed over the prediction horizon H as shown in Fig. 12. For the initial state, we define

$$\mathbf{z}_0 := (\mathbf{x}_0,\, \mathbf{x}_{\text{des},0} - \mathbf{x}_0,\, \mathbf{0})\,. \tag{45}$$

4.2.2 Cost Function Derivatives

Given the presented augmentation and propagation steps, the expected cost gradient can be computed analytically such that the policy can be efficiently optimized using gradient-based methods. We summarize the high-level policy gradient derivation steps to point out the modifications to standard PILCO that are necessary to allow PID policy optimization. The expected cost[1] derivative is obtained as

$$\frac{dJ(\theta)}{d\theta} = \sum_{t=1}^{H} \frac{d}{d\theta} \underbrace{\mathbb{E}_{\mathbf{z}_t}[c(\mathbf{z}_t)]}_{=:\mathcal{E}_t} = \sum_{t=1}^{T} \frac{d\mathcal{E}_t}{dp(\mathbf{z}_t)} \frac{dp(\mathbf{z}_t)}{d\theta}\,. \tag{46}$$

[1] We only address cost on the state $\mathbf{z}_t$ for simplicity. For practical implementations, cost on $\mathbf{u}$ can be included by adding past inputs into the system state as shown in Sec. 4.3.

To simplify the notation, we write $dp(\mathbf{z}_t)$ to denote the sufficient statistics derivatives $d\mu_t$ and $d\Sigma_t$ of a Gaussian random variable $p(\mathbf{z}_t) \sim \mathcal{N}(\mu_t, \Sigma_t)$ (analogous to the treatment in [33]). The gradient of the immediate loss with respect to the state distribution, $d\mathcal{E}_t/dp(\mathbf{z}_t)$, is readily available for most standard cost functions like quadratic or saturated exponential terms and Gaussian input distributions (cf. [35]). The gradient for each predicted state in the long-term rollout is obtained by applying the chain rule to (44) resulting in

[33] Deisenroth, Fox, and Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," 2015

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

$$\frac{dp(\mathbf{z}_{t+1})}{d\theta} = \frac{\delta p(\mathbf{z}_{t+1})}{\delta p(\tilde{\mathbf{z}}_t)} \frac{dp(\tilde{\mathbf{z}}_t)}{d\theta} + \frac{\delta p(\mathbf{z}_{t+1})}{\delta p(\mathbf{x}_{t+1})} \frac{dp(\mathbf{x}_{t+1})}{d\theta}\,. \tag{47}$$

The derivatives highlighted in blue are computed for the linear transformation in (44) according to the general rules for linear transformations on Gaussian random variables as summarized in the appendix. Based on the dynamics model prediction as detailed in Sec. 4.2.1, the gradient of the dynamics model output $\mathbf{x}_{t+1}$ is given by

$$\frac{dp(\mathbf{x}_{t+1})}{d\theta} = \frac{\delta p(\mathbf{x}_{t+1})}{\delta p(\tilde{\mathbf{z}}_t)} \frac{dp(\tilde{\mathbf{z}}_t)}{d\theta} + \frac{\delta p(\mathbf{x}_{t+1})}{\delta p(\mathbf{u}_t)} \frac{dp(\mathbf{u}_t)}{d\theta}\,. \tag{48}$$

The derivatives shown in red can be computed analytically for the specific dynamics model [22]. Applying the chain rule for the policy output $p(\mathbf{u}_t)$ obtained by (41) yields

[22] Candela, Girard, Larsen, and Rasmussen, "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting," 2003

$$\frac{dp(\mathbf{u}_t)}{d\theta} = \frac{\delta p(\mathbf{u}_t)}{\delta p(\tilde{\mathbf{z}}_t)} \frac{dp(\tilde{\mathbf{z}}_t)}{d\theta} + \frac{\delta p(\mathbf{u}_t)}{\delta \theta}\,, \tag{49}$$

The derivatives marked in blue are introduced by the linear control law (41) and can be computed as summarized in the appendix. The gradient of the augmented state is given by

$$\frac{dp(\tilde{\mathbf{z}}_t)}{d\theta} = \frac{dp(\tilde{\mathbf{z}}_t)}{dp(\mathbf{z}_t)} \frac{dp(\mathbf{z}_t)}{d\theta} \tag{50}$$

Again, the part marked in blue is computed for the linear transformation (40). Starting from the initial state where $dp(\mathbf{z}_0)/d\theta = 0$, we obtain the gradients for all states with respect to the policy parameters $dp(\mathbf{z}_t)/d\theta$ by iteratively applying (47) to (50) for all time steps t.

4.2.3 Multiple Initial-State Goal State Combinations

In [32] PILCO has been extended to multiple-target reinforcement learning which means to search for policies that are not only able to stabilize the system around one predefined target but policies that can stabilize the system around multiple targets. These policies are meant to generalize to arbitrary targets and to allow for trajectory tracking. Therefore, the policy is defined as a function of the state *and* the target $\mathbf{u} = \pi(\mathbf{x}, \tau; \theta)$.

[32] Deisenroth and Fox, "Multiple-target reinforcement learning with a single policy," 2011

What they proposed is to use the joint probability distribution of state $\mathbf{x}_t = \mathcal{N}(\mu_t, \Sigma_t)$ and the derivation from a target $\tau = \mathcal{N}(\chi, \Phi_t)$ given by

$$p(\mathbf{x}_t, \tau_t - \mathbf{x}_t) = \mathcal{N}\left(\begin{bmatrix} \mu_t \\ \chi_t - \mu_t \end{bmatrix}, \begin{bmatrix} \Sigma_t & -\Sigma_t \\ -\Sigma_t & \Sigma_t + \Phi \end{bmatrix} \right) \tag{51}$$

$$=: \mathcal{N}(\mathbf{x}_t^{x,\tau} | \mu_t^{x,\tau}, \Sigma_t^{x,\tau}) \tag{52}$$

Given multiple training targets $\tau = (\tau_1, \ldots, \tau_N)$, the expected return J^π assuming a uniform prior $p(\tau)$ over all targets is approximated by

$$\mathbb{E}[J^\pi(\theta)] \approx \frac{1}{|\tau|} \sum_\tau \sum_{t=1}^{T} \mathbb{E}[c(\mathbf{x}_t)|\tau]. \tag{53}$$

Within our framework this is a specific case of a policy using only the current state and error from the fully augmented state distribution $\tilde{\mathbf{x}}^{(3)}(t)$.

Given the fully augmented state $\tilde{\mathbf{x}}^{(3)}(t)$ as presented in Sec. 4.1, policies can be defined on arbitrary partitions of this state. This allows for combinations like PID error feedback policies combined with state feedback. Concepts like velocity compensation plus error feedback as used in hydraulic force control can therefore easily incorporated in this framework (cf. [2]).

[2] Alleyne and Liu, "A simplified approach to force control for electro-hydraulic systems," 2000

Instead of encoding an implicit goal state within the cost function, we define a desired goal trajectory given by $x_{des} = (x_{des_1}, \ldots, x_{des_T})$.

As demonstrated in [32], we can not only optimize the control policy with respect to one combination of initial state x_0 and goal state x_{des} but with respect to multiple combinations by optimizing over the summarized cost given by

$$J(\theta) = \sum_{k=1}^{K} \sum_{t=1}^{T} \mathbb{E}\{c(\mathbf{x})\} \tag{54}$$

where K is the number of initial state and goal state combinations. The cost function is therefore formulated as a function of an current augmented state $\tilde{\mathbf{x}}^{(1)}$ where the measured state is augmented by the currently desired state of the goal trajectory $\tilde{\mathbf{x}}^{(1)}(t) = (\mathbf{x}(t)^T, \mathbf{x}_{des}(t)^T)^T$.

[32] Deisenroth and Fox, "Multiple-target reinforcement learning with a single policy," 2011

4.2.4 Time Varying Goal States

Additionally, in our framework, the target is not limited to one fixed setpoint but a time varying target trajectory might be incorporated. Each target trajectory is defined as $\tau_i = (x_{des,1}, \ldots, x_{des,T})$. The intuition is to enable exploration of wider regions of the target space by only a small number of initial state/target combinations through the use of time varying target positions and therefore better generalization to new target positions.

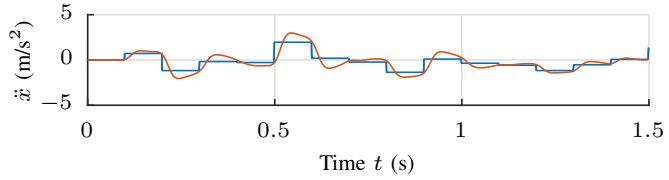
Notice the two possibilities to enforce tracking behavior in this framework. A control policy can be trained on multiple constant combinations of initial state and goal states ($|\tau| > 1$) (e.g. demonstrated in [32]). At the same time a policy can be trained on one (or possibly multiple) time varying goal trajectories to further emphasis the tracking property of the desired control policy.

4.3 Experimental Evaluation

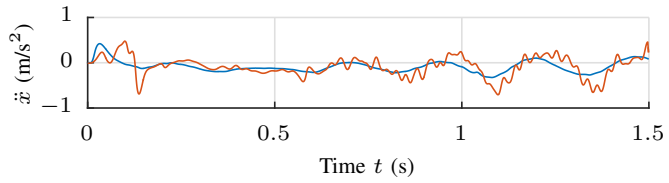
To demonstrate the capabilities of the presented framework to automatically tune coupled PID controllers without prior system knowledge and in a data-efficient fashion, we consider the problem of balancing an inverted pendulum on the Apollo robot as shown in Fig. 7. The inverted pendulum is a well-known benchmark in the control and reinforcement learning communities [146, 4]. Demonstrations of the iterative learning process and the resulting optimized policy can be found in the supplementary video material.

[146] Sutton and Barto, *Reinforcement learning: An introduction*, 1998

[4] Anderson, "Learning to control an inverted pendulum using neural networks," 1989



(a) Apollo robot simulation in SL.



(b) Real-world Apollo robot.

Figure 13: Comparison of the commanded (——) and actual (——) acceleration on the Apollo robot. Significant differences are apparent between the commanded acceleration and the actual acceleration, both in simulation (a) and, even stronger, on the real system (b). Effects are caused by joint-level friction, gear-drives and vibrations.

4.3.1 PID Learning for Robot Balancing

Experimental Setup

We employ an imperfect inverse dynamics model of Apollo's seven Degree-of-Freedom (DoF) robotic arm to compute the joint torques necessary to track the desired end effector acceleration u_t [128]. Technical details concerning the hardware platform can be found in [90], where reinforcement learning on this platform has been addressed using Bayesian Optimization techniques. The state of the system $\mathbf{x}_t$ comprises the end effector position x, velocity $\dot{x}$, pendulum angle ϕ, and angular velocity $\dot{\phi}$.

During the rollouts, the commanded acceleration is limited to $u_{\max} = 3\,\mathrm{m/s^2}$ for safety reasons. Test policies are executed for 20 s or interrupted once safety limits ($x_{\max} = 0.3\,\mathrm{m}$, $\theta_{\max} = 30^\circ$) are violated. The control signal is computed at 100 Hz and low-pass filtered by a second order Butterworth filter having a cut-off frequency of 20 Hz. Joint encoder readings and the robot's kinematic model are used to calculate the end-effector position at 1 kHz, whilst the pendulum position is tracked by a VICON vision system at 200 Hz.

The policy is optimized on a prediction horizon of $T = 10\,s$ based on a saturated loss function [31] given by

$$c(\mathbf{e}_t, \bar{\mathbf{u}}_t) = 1 - \exp(-(\mathbf{e}_t^T Q \mathbf{e}_t + \bar{\mathbf{u}}_t^T R \bar{\mathbf{u}}_t)/2)\,. \tag{55}$$

For balancing the pendulum in the upright position the desired trajectory is given by $\mathbf{x}_{\mathrm{des},t} = \mathbf{0}$. Weights are set to $Q = \mathrm{diag}(1/0.2^2, 1/0.02^2)$ for end effector and pendulum position error, and to $R = 1/0.4^2$ for the control input. The selected cost function saturates quickly for $x > w$ if x is weighted by $1/w^2$. End effector position and input are

[128] Righetti, Kalakrishnan, Pastor, Binney, Kelly, Voorhies, Sukhatme, and Schaal, "An autonomous manipulation system based on force control and optimization," 2014

[90] Marco, Hennig, Bohg, Schaal, and Trimpe, "Automatic LQR Tuning Based on Gaussian Process Global Optimization," 2016

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

therefore only penalized lightly, which permits higher gains while the pendulum angle is stabilized as it is penalized much stronger.

PID CONTROL STRUCTURE SETUP We employ a PID controller on the position error $e_x = x_{\text{des},t} - x$ and a PD controller acting on the pendulum angular error $e_\theta = \theta_{\text{des},t} - \theta$. The resulting control structure is shown in the right plot of Fig. 11. The PID/PD control structure with integral control on the end effector position serves to correct for any static bias in the angle measurement (e.g., from imperfect calibration) as is explained in [154, p. 67]. This structure has successfully been used for other balancing problems [90, 154].

[154] Trimpe and D'Andrea, "The Balancing Cube: A Dynamic Sculpture As Test Bed for Distributed Estimation and Control," 2012

[90] Marco, Hennig, Bohg, Schaal, and Trimpe, "Automatic LQR Tuning Based on Gaussian Process Global Optimization," 2016

The specific structure is chosen based on prior knowledge about the problem at hand as detailed in [90]. The integrator's contribution is required to counteract any pendulum angle measurement bias introduced by imperfect sensor calibration. The policy parametrization is therefore given by $\theta = (K_{\text{p},x}, K_{\text{i},x}, K_{\text{d},x}, K_{\text{p},\theta}, K_{\text{d},\theta})$. Assuming no prior knowledge, we initialize the policy to zero. Both controllers are coupled by the system dynamics and can therefore not be tuned independently, which makes the inverted pendulum a well suited benchmark for multivariate PID controller tuning.

MODIFICATIONS FOR DYNAMICS MODEL LEARNING When first training GP dynamics models $\mathbf{x}_{t+1} = \hat{f}(\mathbf{x}_t, \mathbf{u}_t)$ in the standard way (Sec. 3.3.1), this did not lead to acceptable models for long-term predictions and thus successful controller learning. The problems are caused by imperfections in the inverse dynamics model, joint friction and stiction, as well as sensor and actuator delay. These factors add unobserved states and therefore additional dynamics to the system, corrupting the measured data. This is visible in Fig. 13, where the desired acceleration and the numerically computed actual acceleration are visualized for a policy rollout on the robot. Several adaptations to the standard GP dynamics model learning framework were required to obtain a good prediction model, which we explain next.

GAUSSIAN PROCESS SETUP In contrast to the model in (19), we train the GP models to predict the difference between the current and the next state $\Delta\mathbf{x}_t = \mathbf{x}_{t+1} - \mathbf{x}_t$. We compute a sparse GP using Snelson's approximation [141] having a covariance parametrized by 400 inducing pseudo-input points. For both GP models, hyperparameters $\theta_{\text{GP},i} = (l_1, l_2, \sigma_\text{f}, \sigma_\text{n})$ including lengthscales for each dimension, as well as signal and noise variance have to be chosen. We chose the maximum likelihood estimate (MLE) on the basis of the previously gathered system data to compute the hyperparameters. These hyperparameters are kept constant during the iterative policy learning.

[141] Snelson and Ghahramani, "Sparse Gaussian processes using pseudo-inputs," 2005

NARX DYNAMICS MODEL In simulation only the joint level tracking dynamics is visible but especially small or continuous changes are tracked reasonably well. In contrast, on the actual system, the effective end effector acceleration deviates much stronger from the commanded acceleration. On the joint level, PID controllers are employed to track the computed joint torques.

In Fig.13 (a), the commanded acceleration (——) and the resulting acceleration at the end-effector (——) are given for a random policy rollout. The tracking dynamics of the low level joint controller is obviously causing deterioration from the perfect tracking that was assumed for the linear simulation (cf. Fig.13 (b)). The true end-effector acceleration (——) is now causing the state transitions visible in the position and velocity states. However, the dynamics model is learned having the *commanded* acceleration (——) as input.

Despite the imperfect accelerating tracking, the presented framework is capable of inferring a dynamics model from data and optimizing the PID control parameters to find a stabilizing policy. Fig.15 (c) is visualizing the predicted system behaviour and one actual rollout on the system for the final PILCO iteration.

This problem is caused by imperfections in the inverse dynamics model, joint friction and stiction as well as sensor and actuator delay, adding unobserved states and therefore additional dynamics to the system, disturbing the measured data. Those low-level controllers are already tuned for the robotic arm and therefore not part of the presented PID tuning. A specific NARX model structure together with data-processing and fixed GP hyperparameters proved to be essential to overcome the resulting artifacts in the recorded data.

Instead of modeling the system's full, four dimensional state and its dynamics, two independent GPs are trained to model the dynamics of the measured state parts; the end effector and pendulum position. The missing information about the system's velocities and potential latent states is recovered by employing a Nonlinear AutoRegressive eXogenous model (NARX) [17] of the form

$$\mathbf{x}_{t+1} = \hat{f}(\mathbf{x}_t, \ldots, \mathbf{x}_{t-n}, \mathbf{u}_t, \ldots, \mathbf{u}_{t-m}) \,. \tag{56}$$

[17] Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*, 2013

Information on latent states is implicitly encoded in the measured historic states and inputs. Different lengths of history might be required for individual parts of the system's state depending on the dynamics' time scales. We optimize the number of historic states individually for end effector position, pendulum position and control input. The NSGA II optimizer [29] is employed to compute the pareto front of the model's prediction error (using the same dataset as previously used for hyperparameter tuning) and the size of the NARX history. For our problem, we ended up with a new state of dimensionality

[29] Deb, Pratap, Agarwal, and Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," 2002

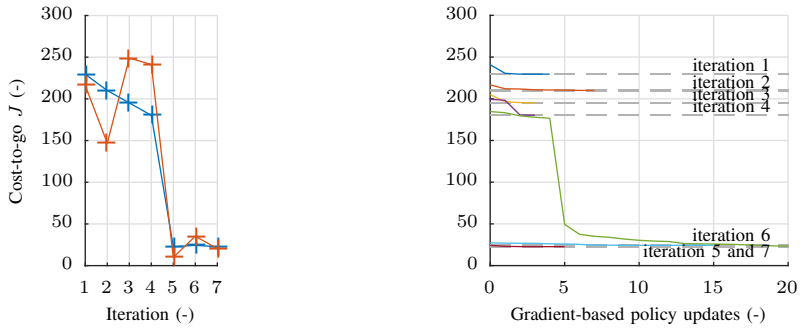


Figure 14: Expected cost-to-go optimization results. Left: Iterative improvement in predicted loss (——) compared to the cost observed in a single robot experiment (——). Right: Optimization results for each iteration.

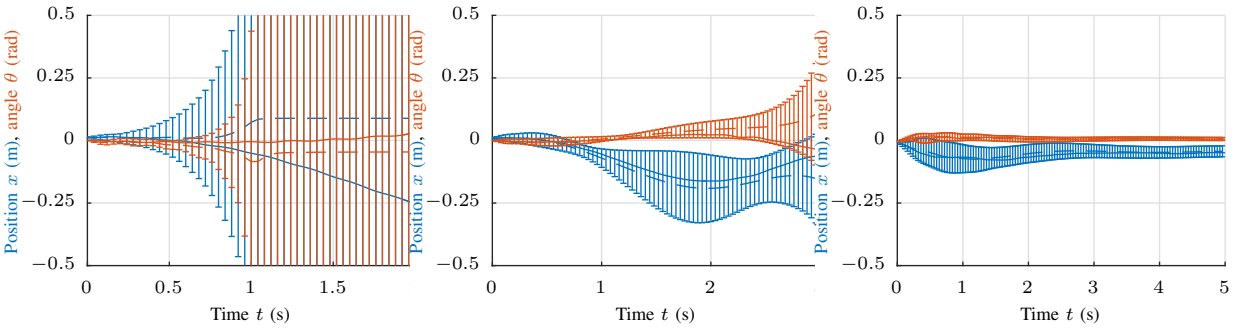
14 given by $\mathbf{x}_{\text{NARX}} := (x_t, \ldots, x_{t-3}, \phi_t, \ldots, \phi_{t-2}, u_t, \ldots, u_{t-6})$. The hidden dynamics between commanded input and executed acceleration (cf. Fig. 13) requires a longer history in the input state to capture all relevant effects.

DATA PREPROCESSING The recorded data is downsampled to 25 Hz and non-causally low-pass filtered with a 2nd order Butterworth filter and cut-off frequency 12.5 Hz. The downsampled control input is obtained by averaging the input signal on each sampling interval.

Learning Results

The iterative learning experiment is visualized in Fig. 14. To gather initial data about the system, four random rollouts are conducted, applying a white noise input $u_{\text{rnd},t} \sim \mathcal{N}(0, 1\,\text{m/s}^2)$ to the system. The first iteration is based on the dynamics model learned from these random rollouts. The left figure visualizes the optimized predicted loss for each iteration in comparison to the actual loss obtained from evaluating (55) on one sample rollout of the actual system. The drop of the predicted loss in iteration 5 shows that by that time, sufficient data about the system dynamics has been gathered to find a stabilizing policy parametrization. In the right figure, the predicted loss optimization is visualized for each iteration individually, as a function of the number of linesearches conducted by the BFGS optimizer.

In Fig. 15, we visualize the dynamics learning progress, showing predicted system behavior and actual rollout for the dynamics models and policies as obtained at the first, intermediate and final stage of the iterative learning process. In each iteration, the predictive distribution is computed for the currently available dynamics model and the currently optimized policy for starting at the initial state of the real system. Given only few data points from the initial random policy rollouts, the model prediction becomes quickly uncertain as shown in Fig. 15 (a). Fig. 15 (b) shows increased model accuracy but unstable controls. The final dynamics model is accurately predicting the stabilization of the system by the optimized policy. The obtained dynamics model is



(a) Iteration 1 (b) Iteration 3 (c) Iteration 7

Figure 15: Predicted system behavior (dashed lines, error-bars indicate 95 % confidence intervals) and experimental results (solid lines) visualized for the first (a), an intermediate (b) and the final iteration (c) of the policy learning process. The end effector position (——) and the angular position (——) are shown.

precisely incorporating disturbances caused by friction and low-level tracking dynamics by additional uncertainty in the state distribution, even in the steady state as visible in Fig. 15 (c).

In this example, the total interaction time with the physical system is only 106 seconds, demonstrating fast and data-efficient learning. This model-based method outperforms a model-free, Bayesian optimization method (cf. [90]), with respect to the number of rollouts on the actual system. The policy optimization itself is carried out offline, and the predicted system behavior can be utilized to set appropriate safety boundaries to test new controllers without damaging the system.

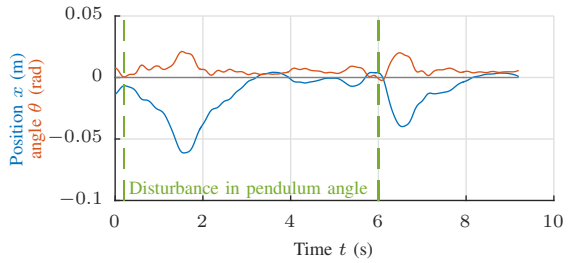


Figure 16: Disturbance rejection of the optimized PID policy. End effector position (——) and pendulum angle (——) display the closed loop response of the optimized PID policy to manually introduced disturbances (- - -).

To demonstrate the robustness of the learned policy, the system is manually deflected. Disturbances in pendulum angle (cf. Fig. 16) and end effector position are dispelled fast and without overshoot. By commanding a non-zero desired trajectory, the learned PID controller can be utilized for tracking tasks as demonstrated in the supplementary video for a sinusoidal end-effector trajectory.

4.3.2 PID Learning for RC Race Car

As a second application example, the proposed PILCO-PID method is deployed to tune the vehicle controller of an autonomous model race car. The model race car, as shown in Fig. 17, is supposed to drive around a predefined race track at the physical limits of the tires. Two coupled PID controllers are commanding throttle and steering to track the desired target trajectory and speed as accurately as possible. A

[90] Marco, Hennig, Bohg, Schaal, and Trimpe, "Automatic LQR Tuning Based on Gaussian Process Global Optimization," 2016

visualization of the test track is shown in Fig. 18. The target velocity is chosen to reach the friction limits of the tires in the fast hairpin corners.

Driving a race car at the friction limits is a difficult control problem for several reasons. Unlike in normal operation, where linear models are good approximations of the system dynamics, close to the optimal friction, the tire-road interaction exposes strong non-linear behavior. Crossing the point of optimal friction, the system becomes inherently unstable. Finally, a strong coupling exist between the lateral and longitudinal forces produces by a tire. This coupling results in a trade-off between lateral (i.e. cornering/steering capabilities) and longitudinal (i.e. acceleration/velocity tracking) control. The race-car is thus an ideal testbed for the proposed PILCO-PID method, since non-linear, unstable dynamics and couplings between the controllers can be jointly addressed.

The evaluation of the PILCO-PID method on the model race car has been conducted in a master thesis project and has been published in [81]. The testbed itself and the proposed, coupled and multivariate PID control structure have been designed and build in a previous master thesis project as documented in [171].

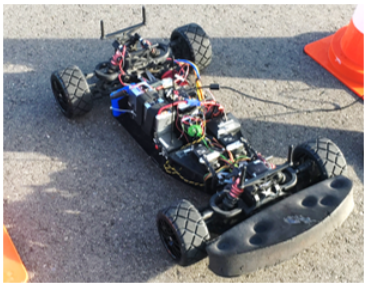


Figure 17: The efficiency of PILCO-PID to tune multivariate PID control structures is demonstrated on a scale 1:6 remote controlled race car.

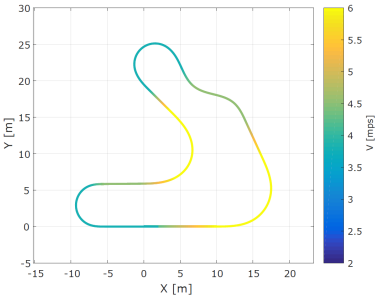


Figure 18: The race track (top-down view) and target vehicle velocity (color coding) are selected to reach the tire's friction limit.

[81] Lefarov, "Model-Based policy Search for Learning Multivariate PID Gain Scheduling Control," 2018

[171] Wischnewski, "Control of highly automated and autonomous vehicles in critical driving situations," 2017

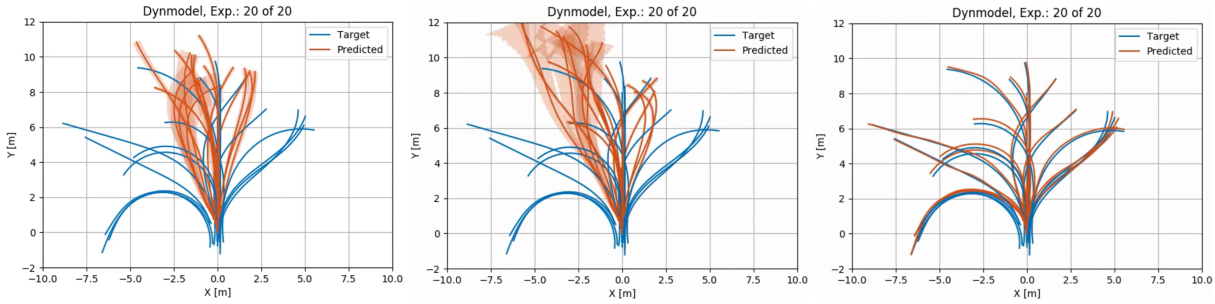
Experimental Setup

The autonomous race car is based on a commercial remote controlled (RC) race car in scale 1:6. The car is equipped with an on-board micro-controller, inertial measure system and actuators to control the vehicle's throttle and steering. Details about the system hardware and control system can be found in [171].

Similar to the experiments on the Apollo robot in Sec. 4.3.1, a GP-NARX model is used to model the dynamics of the race car. In particular, the GP models the position and orientation of the car by taking into account the previous positions, orientation, steering, and throttle inputs.

For optimization, the car is placed at the origin (vehicle position $x = y = 0$). The target trajectories are sub-trajectories of length three seconds, sampled at random from the full race track. Each target trajectory is started with a deviation of $\Delta x, \Delta y \sim \mathcal{N}(0, 0.2^2)$ from the origin to foster tracking of the desired target in the presence of initial tracking errors. As detailed in Sec. 4.2.3 and Sec. 4.2.4, multiple, time-varying goal states are utilized to optimize the policy for multiple target trajectories simulatenously. A visualization of the resulting optimization problem is provided in Fig. 19. Details about the model training, cost function and experimental setup can be found in [81].

[81] Lefarov, "Model-Based policy Search for Learning Multivariate PID Gain Scheduling Control," 2018



(a) Iteration 5 (b) Iteration 15 (c) Iteration 30

Figure 19: The policy learning progress for the RC race car is visualized. A batch of desired target trajectories (——) is depicted together with the predicted race car behavior (mean ——, standard deviation ▭) for different iterations in the policy optimization process. Whilst the initial policy (a) is mostly not steering at all, the trained policies learned to accurately track the desired targets (c).

Learning Results

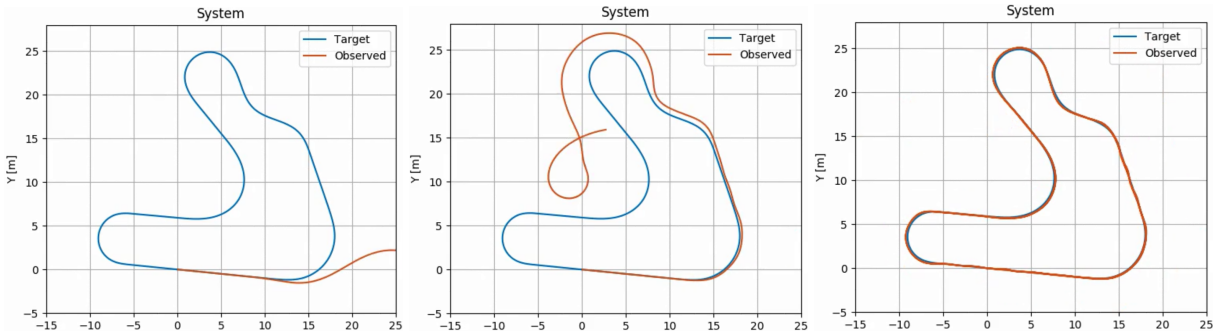
For the inner learning loop of PILCO-PID (cf. Alg. 2), the current GP dynamics model is employed to predict the closed-loop system behavior and to find the optimal policy parameters. In Fig. 19, the predicted system behavior and the desired target trajectories are visualized for (a) beginning, (b) intermediate, and (c) final iterations in the policy optimization.

For learning, the controller is initialized with zero PID gains (i.e. $\theta = 0$). Therefore, the GP model correctly predicts straight line motion with nearly no steering or acceleration/deceleration commands. In the absence of control interventions, the uncertainty about the closed-loop system behavior grows over time. In Fig. 19, the mean (——) and standard deviation (▭) of the predicted vehicle position is visualized together with the target trajectories (——).

In later iterations (cf. Fig. 19 (b)), the updated policy parameters result in an improved, closed-loop system behavior. The final, learned PID controller (cf. Fig. 19 (c)) is able to track the desired target trajectories and simultaneously compensate uncertainty in the system model. As depicted, the optimized controller compensates for uncertainty (e.g. disturbances) in the vehicle position and reaches the desired trajectory with little uncertainty.

So far, the predicted closed-loop behavior is subject to the accuracy of the learned GP model. For comparison, the resulting policy parameters in different stages of the optimization process are evaluated in a high-fidelity vehicle simulation and on the test vehicle. For validation, the target is to drive multiple laps around the test-track whilst minimizing lateral and longitudinal tracking errors.

The target track and the resulting vehicle trajectories are visualized in Fig. 20. As predicted by the GP model (cf. Fig. 19 (a)), the initial policy with close to zero PID gains, can barely control the vehicle and misses the first corner (cf. Fig. 20 (a)). Subsequent policy iterations in the policy optimization drastically improve the tracking performance. In iteration 15 (cf. Fig. 20 (b)) long and low-speed corners can be tracked by the learned policy. Finally, in iteration 30 (cf. Fig. 20 (c)),



(a) Iteration 5 (b) Iteration 15 (c) Iteration 30

Figure 20: The race car's target trajectory (——) is visualized together with the actual, closed-loop, race car trajectory (——) for different iterations in the policy learning process. Whilst initial controllers are not able to control the race car in corners (cf. Fig. (a) and (b)), the learned policy from later iterations can accurately track the desired target (cf. Fig (c)).

the final policy is able to drive multiple labs at target velocity with minimal tracking deviations.

5

Summary

This first part of the work introduced PILCO-PID, a novel algorithm to automatically tune PID controllers, which are the number one control structures utilized in industrial applications. We derive the PILCO-PID method as an extension of the Probabilistic Inference for Learning COntrol (PILCO) [35] framework. Specifically, this framework represents arbitrary, coupled, and multivariate PID controllers as linear state feedback controllers in a new augmented system state. This way, the PILCO-PID method can inherit the principled treatment of uncertainty and handling of non-linear models directly from PILCO. The proposed framework for multivariate PID tuning is flexible concerning the possible PID control structures and the nature of the process dynamics. In particular, it can cope with general non-linear MIMO processes and multivariate PID structures.

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

In this part, we demonstrate the efficiency of PILCO-PID to tune coupled PID controllers on real-world systems in two example applications. The first example demonstrates the efficiency of the proposed PILCO-PID method to calibrate PID controllers for an inverted pendulum balancing task on a complex humanoid upper-body robotic system. Without prior knowledge about the robot's kinematics or dynamics, the proposed method can learn robust, stabilizing controllers within few rollouts on the real robot. With the tuning of PID controllers for the control of an autonomous model race car, a second example sheds light on learning PID control for a strongly coupled and non-linear system. The learning agent is facing severe non-linear system dynamics and strong coupling between the lateral and longitudinal vehicle dynamics to operate the car at the physical limits of the tire-road friction.

The presented framework can readily be extended to cascaded PID structures and tracking controllers by considering multiple different [32] and time-varying goal states. In the example of learning to control a race car, time-varying goal states and multiple goal trajectories facilitated efficient learning of a global policy. The extension of the PILCO-PID concept to *gain-scheduled PID* controllers is feasible. For gain-scheduled PID control, the PID gains are no longer static but

[32] Deisenroth and Fox, "Multiple-target reinforcement learning with a single policy," 2011

become a function of the system state. In continuous-time system, the gain-scheduled PID controller would thus be formulated as

$$u(t) = K_p(x(t))e(t) + K_i(x(t)) \int_{\tau=0}^{t} e(\tau)d\tau + K_d(x(t))\dot{e}(t) \tag{57}$$

$$e(t) = x_{\text{des}}(t) - x(t)\,. \tag{58}$$

Initial work on extending PILCO-PID in this direction has been presented in [81]. Different from the PID controller, the treatment of uncertainty becomes more challenging for gain-scheduled PID controllers. In particular, the joint distribution of system state $x(t)$ and control signal $u(t)$ is no longer Gaussian, and approximations are required to keep an analytically tractable, Gaussian distribution. In particular, the non-linear dependency of $u(t)$ on $x(t)$ via the non-linear control-gain mappings $K_p(x(t))$, $K_i(x(t))$, and $K_d(x(t))$ complicates the computation of the cross-covariance between control $u(t)$ and state $x(t)$. Possible approximate solutions have been proposed in [81].

[81] Lefarov, "Model-Based policy Search for Learning Multivariate PID Gain Scheduling Control," 2018

Appropriately dealing with hidden and low-level dynamics, problems found in almost all real-world applications were significant hurdles in the experimental application. In particular, we found that the learning of dynamics models geared towards long-term predictions is critical to successful finite horizon policy optimization but largely unaddressed by current approaches. In the following part of the work, we will present principled ways to set up and learn models tailored towards the use in real-world, model-based RL systems.

Part II

Learning Models for Model-Based Reinforcement Learning

6 Introduction

As introduced in the first part of this work, model-based reinforcement learning methods are a promising direction to automatically derive control policies in industrial applications. In the past, model-based RL was already involved in the solution of several important RL problems such as learning directly from high-dimensional input space [82], generalizing between local solutions [83], safe exploration [14] or data-efficient learning [31]. Despite these advances, many state-of-the-art model-based RL methods require careful set-up, tuning and supervision when applied to real-world systems. Significant shortcomings of these methods are rooted in the respective model assumptions and the employed model-learning and prediction algorithms. In this part of the work, we introduce two novel modeling frameworks to improve model-based RL's robustness in real-world scenarios. The presented modeling frameworks follow two main directions to achieve this goal.

[82] Levine, Finn, Darrell, and Abbeel, "End-to-end training of deep visuomotor policies," 2016

[83] Levine and Koltun, "Guided Policy Search," 2013

[14] Berkenkamp, Schoellig, and Krause, "Safe controller optimization for quadrotors with Gaussian processes," 2016

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

SUITABLE MODEL ASSUMPTIONS First and foremost, strong model assumptions are often required to simplify the mathematical problem but do not well match the characteristics of real-world systems. Typical model-based RL methods incorporate assumptions about the dynamical system's structure, noise processes, available data or observability. For example, independently and identically distributed (i.i.d.) noise-free data of a system's input and output signals is most commonly expected. In contrast, data from the closed-loop operation of a dynamical system is typically highly correlated. Sensor imperfections typically corrupt the measured system signals. In particular, noise is not only limited to the model output (e.g., the predicted next system state) but needs to be incorporated in the model input as well. Similarly, the available sensors are typically not sufficient to measure the system's full state, such that the standard MDP description, i.e., the assumption of a measured, Markovian state, is most commonly violated.

MODELS TAILORED FOR RL The second line of work to improve the robustness and data-efficiency of model-based RL frameworks strives to adapt general model-learning frameworks to the subsequent task of model-based RL. By taking the model-based policy search requirements into account, we can derive methods, which use the available data more efficiently. As a result, these methods require fewer rollouts or facilitate more stable learning. For example, model-based

RL requires good *long-term* predictions from complete closed-loop rollouts. In contrast, typical model learning approaches optimize one-step-ahead predictions. The resulting models tend to diverge when predicting long-term behavior due to the accumulation of small errors [17]. Furthermore, when using a model to infer a policy, the model does not necessarily need to perform well for arbitrary inputs, but rather for typical inputs in the policy space.

[17] Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*, 2013

6.1 Contributions

This part of the work proposes two novel model-learning techniques, Multi-Step Gaussian Processes (MSGPs) and Probabilistic Recurrent State-Space Models (PR-SSMs). Both methods are tailored to efficiently learn long-term predictive models for real-world problems, where latent states and noisy data are typically encountered.

With MSGP an efficient inference scheme is presented for latent autoregressive GP dynamics models. These models are particularly suited for model learning in policy search applications like PILCO, however their effectiveness is demonstrated as well for the pure model-learning use-case.

PR-SSM on the other hand presents an efficient inference scheme for Gaussian Process State-Space models. With the proposed doubly stochastic variational inference method, important limitations of previous model learning techniques can be overcome. This method allows to efficiently optimize the true latent state posterior distribution, maintain the temporal correlations and thus learn non-linear Gaussian process state-space models. At the same time the combination of gradient-based information and sampling facilitates efficient and scalable learning.

The main portion of the work presented in this part of the thesis has been previously published at the following venues.

> A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, M. Toussaint, and S. Trimpe. "Optimizing Long-term Predictions for Model-based Policy Search." In: *Conference on Robot Learning (CORL)*. 2017, pp. 227–238

> A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. "Probabilistic Recurrent State-Space Models." In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1280–1289

6.2 Outline

This part initially discusses the system identification problem, in particular problems encountered on real-world, physical systems in Sec. 6.3. Problems, related work and connections to the MSGP and PR-SSM methods are outlined for problems such as noisy outputs, latent states and noisy inputs in Sec. 6.3.1, Sec. 6.3.2, and Sec. 6.3.3 respectively.

The novel Multi-Step Gaussian Process (MSGP) model and inference scheme for improved long-term predictive model learning is presented in the Sec. 7. Based on the main ideas underlying MSGP as detailed in Sec. 7.1.1, the computation and optimization of the long-term predictive distribution is derived in Sec. 7.1.2 and Sec. 7.1.3 resepctively. The close link and thus reusability of the proposed MSGP method in policy search, i.e., reinforcement learning, applications is discussed in Sec. 7.1.4. Experimental results are presented in Sec. 7.2 for both, the model learning performance (cf. Sec. 7.2.1) and the improvements achieved in policy search applications (cf. Sec. 7.2.2). The summary in Sec. 7.3 discusses the main advantages and drawbacks of MSGP.

The second novel contribution in the model learning part of this work is the Probabilistic Recurrent State-Space-Model (PR-SSM) as presented in Sec. 8. The model structure is outlined in Sec. 8.1 followed by an efficient and tractable inference scheme in Sec. 8.2. Extensions are presented to scale PR-SSM to large scale datasets in Sec. 8.3 Experimental evaluations demonstrate the PR-SSM performance in Sec. 8.4 on both benchmark data (cf. 8.4.2) and large-scale data (cf. 8.4.3). This part concludes with a discussion of the proposed model-learning methods in Sec. 8.5.

6.3 System Identification

System identification is the task of identifying mathematical models to describe the behavior or patterns of input and output signals from dynamical systems [89]. These models are key to model-based control design [10, 21] and model-based reinforcement learning (RL) [31, 41]. In RL we usually assume no access to an analytical model of the real system. Therefore, we need to infer the model from the available input/output data. From an interaction with the system of length H, we obtain a trajectory of input/output data $\tau = ((\boldsymbol{u}_0, \boldsymbol{y}_0), \dots, (\boldsymbol{u}_H, \boldsymbol{y}_H))$ and possibly the corresponding feedback policy π_θ with parameters θ. Learning good models from experimental data is difficult for several reasons, which will be discussed in the following sections.

[89] Ljung, "Perspectives on system identification," 2010

[10] Aström and Murray, *Feedback systems: an introduction for scientists and engineers*, 2010

[21] Camacho and Alba, *Model predictive control*, 2013

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

[41] Doerr, Nguyen-Tuong, Marco, Schaal, and Trimpe, "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers," 2017

6.3.1 Noisy outputs

For system identification, we are limited to a finite amount of data, which is corrupted by noise due to stochasticity in the system and in the sensors. The resulting uncertainty about the true system behavior given our data is generally classified into aleatoric and epistemic uncertainty [37].

[37] Der Kiureghian and Ditlevsen, "Aleatory or epistemic? Does it matter?" 2009

Aleatoric uncertainty is caused by stochastic processes in the underlying system, which we cannot influence. Therefore, the results of our experiments might be altered slightly each time. In case of dynamical systems this noise might be due to process noise, which alters the state transitions. A second source of noise might be observation noise, caused by sensor imperfections, which alters our observations. Epistemic uncertainty on the other hand is due to our lack of data or insufficient prior knowledge. Given a finite amount of measured data, there necessarily remains some inherent uncertainty about the true, underlying system behavior.

In system identification we try to incorporate and model these uncertainties. Accurately modeling uncertainties is particularly important to prevent control optimization from exploiting erroneous parts in the model [106]. Assuming a fully measurable system state and neglecting the observation noise, the one-step system dynamics could be modeled as

[106] Murray-Smith, Sbarbaro, Rasmussen, and Girard, "Adaptive, cautious, predictive control with Gaussian process priors," 2003

$$\boldsymbol{x}_{t+1} = \hat{\boldsymbol{f}}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \epsilon_t. \tag{59}$$

Bayesian nonparametric Gaussian Process (GP) models [168] are a popular choice for dynamics models as they allow one to incorporate uncertainty about model and predictions [117] in a fully Bayesian way.

[168] Williams and Rasmussen, *Gaussian processes for machine learning*, 2005

[117] Peterka, "Bayesian system identification," 1981

Based on GP dynamics models, several methods have been proposed to propagate model uncertainty over multiple prediction steps using assumed density filtering (moment matching) [124] [58] or variational approximations [27]. However, the GP models are usually intrinsically optimized for one-step-ahead predictions.

[124] Quinonero-Candela, Girard, and Rasmussen, *Prediction at an uncertain input for Gaussian processes and relevance vector machines-application to multiple-step ahead time-series forecasting*, 2002

by a GP $\hat{\boldsymbol{f}}$ as

In contrast, both novel model learning frameworks, MSGP and PR-SSM operate on long-term prediction objectives. MSGP involves the moment matching approximation to facilitate analytically tractable and therefore fast computations of the long-term predictive distribution and its gradients. PR-SSM in contrast involves a mixture of variational inference and sampling to allow for an even better approximation of the true long-term predictive distribution.

[58] Girard, Rasmussen, Quinonero-Candela, Murray-Smith, Winther, and Larsen, "Multiple-step ahead prediction for non linear dynamic systems—a Gaussian process treatment with propagation of the uncertainty," 2003

[27] Damianou and Lawrence, "Deep Gaussian Processes.," 2013

The aleatoric uncertainty can further be divided into uncertainty in the underlying process and uncertainty when observing this underlying process. The process uncertainty is typically modeled by process

noise as in (59). To account for sensor noise, an observation model can be added, resulting in the system model

$$\boldsymbol{x}_{t+1} = \hat{\boldsymbol{f}}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \epsilon_t. \tag{60}$$

$$\boldsymbol{y}_t = \boldsymbol{x}_t + \gamma_t\,, \tag{61}$$

with noise process γ. In contrast to typical Bayesian treatments [76] of dynamical systems with models similar to (59), both MSGP and PR-SSM optimize the additional observation noise contribution as in (61).

6.3.2 Latent states

Modeling the behavior of systems with only partially observable states has been an active field of research for many years and several schools of thought have emerged. Representations range from SSMs [157] over Predictive State Representations (PSRs) [87, 140, 130] to autoregressive models [103, 58, 85, 17], as well as hybrid versions combining these approaches [91, 93, 39]. In most real-world scenarios, only a part of the relevant system state can be directly observed. To still capture the relevant information, which is required to predict the future system behavior, two classes of methods have been established:

History-Based Models

The first class utilizes historical input and output measurements up to a horizon l_y and l_u respectively, to predict the next observation. Including previous measurements can resolve the issue of having only partial state measurements in many real systems. This leads to Nonlinear AutoRegressive models with eXogenous inputs (NARX),

$$\boldsymbol{y}_{t+1} = \hat{\boldsymbol{f}}(\boldsymbol{y}_t, \ldots, \boldsymbol{y}_{t-l_y}, \boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t-l_u})\,. \tag{62}$$

NARX models based on GPs (GP-NARX) have been proposed for example by [104] and [76].

Autoregressive (history-based) methods avoid the complex inference of a latent state and instead directly learn a mapping from a history of h past inputs and observations to the next observation, i.e. $\boldsymbol{y}_{t+1} = f(\boldsymbol{y}_{t:t-h}, \boldsymbol{u}_{t:t-h})$. These models face the issue of learning from noise corrupted input data. Recent work addresses this problem by either actively accounting for input noise [94] or reverting to a hybrid, autoregressive formulation in a latent, but noise free state [93, 39]. Such models can be made deep and trained in a recurrent manner as presented in [91]. In theory, a horizon h identical to the true latent state dimensionality (dimension of $\boldsymbol{x}_t$) is sufficient to model all relevant dependencies of the system under consideration [88]. In practice,

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[157] Van Overschee and De Moor, *Subspace identification for linear systems: Theory - Implementation - Applications*, 2012

[87] Littman and Sutton, "Predictive representations of state," 2002

[140] Singh, Littman, Jong, Pardoe, and Stone, "Learning predictive state representations," 2003

[130] Rudary and Singh, "A nonlinear predictive state representation," 2004

[103] Murray-Smith and Girard, "Gaussian Process priors with ARMA noise models," 2001

[58] Girard, Rasmussen, Quinonero-Candela, Murray-Smith, Winther, and Larsen, "Multiple-step ahead prediction for non linear dynamic systems—a Gaussian process treatment with propagation of the uncertainty," 2003

[85] Likar and Kocijan, "Predictive control of a gas–liquid separation plant based on a Gaussian process model," 2007

[17] Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*, 2013

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[93] Mattos, Damianou, Barreto, and Lawrence, "Latent Autoregressive Gaussian Processes Models for Robust System Identification," 2016

[39] Doerr, Daniel, Nguyen-Tuong, Marco, Schaal, Toussaint, and Trimpe, "Optimizing Long-term Predictions for Model-based Policy Search," 2017

[104] Murray-Smith, Johansen, and Shorten, "On transient dynamics, off-equilibrium behaviour and identification in blended multiple model structures," 1999

[94] McHutchon and Rasmussen, "Gaussian process training with input noise," 2011

[88] Ljung, "System identification," 1998

however, autoregressive models typically need a much larger history horizon to cope with noisy observations and arbitrary sampling frequencies.

State-Space Models

SSMs form the second class of methods, where an explicit representation of a Markovian system state is inferred.

State-space models (SSMs) are one popular class of representations for model learning [17], which describe a system with input $\boldsymbol{u}_t$, output $\boldsymbol{y}_t$, and a latent Markovian state $\boldsymbol{x}_t$. The transition model f, observation model g, process, and measurement noise $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\gamma}_t$ form a discrete-time SSM

$$\begin{aligned} \boldsymbol{x}_{t+1} &= f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}_t \,, \\ \boldsymbol{y}_t &= g(\boldsymbol{x}_t) + \boldsymbol{\gamma}_t \,. \end{aligned} \tag{63}$$

[17] Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*, 2013

In this work, we consider the control of an unknown dynamical system given as a State-Space Model (SSM) with unknown transition function $\boldsymbol{f}$ and observation function $\boldsymbol{g}$ as

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}_t^{(x)} \,, \quad \boldsymbol{y}_t = \boldsymbol{g}(\boldsymbol{x}_t) + \boldsymbol{\epsilon}_t^{(y)} \,, \tag{64}$$

with Gaussian process noise $\boldsymbol{\epsilon}_t^{(\mathrm{x})} \sim \mathcal{N}(0, \Sigma_{\mathrm{x}})$ and Gaussian sensor noise $\boldsymbol{\epsilon}_t^{(\mathrm{y})} \sim \mathcal{N}(0, \Sigma_{\mathrm{y}})$. The system has time discrete, continuously valued inputs $\boldsymbol{u}_t \in \mathbb{R}^{D_{\mathrm{u}}}$ and continuously valued observations $\boldsymbol{y}_t \in \mathbb{R}^{D_{\mathrm{y}}}$. Its internal, latent state $\boldsymbol{x}_t \in \mathbb{R}^{D_{\mathrm{x}}}$ can usually not be fully measured. Inputs to the system are either feedforward control signals or generated by a deterministic policy $\boldsymbol{u}_t = \pi(\boldsymbol{y}_t; \theta_\pi)$ parametrized by θ_π.

For a Bayesian treatment, GP-SSMs have been proposed by [49], where both the transition model and the observation model can have GP priors. In [51], data preprocessing and Bayesian modeling is combined for SSM. Inference in latent space proves to be a challenging task which is the major drawback of the existing SSM methods. Usually, an expectation-maximization (EM)-like method is employed to alternate between latent-state inference (E-Step) and model learning (M-step). While the E-step is usually based on smoothing methods, either using sampling [50], variational approximations [49] or moment matching approximations [155], the M-step usually consists in optimizing a variational lower bound on the trajectory likelihood. Learning the latent state representation proves to be a hard problem as it leads to a number of open parameters, which increases linearly in the dataset size. In recent work [91, 44], the fully parametrized latent state has been replaced by a recurrent recognition model, which is essentially a deep network that infers the latent state from future and past observations and therefore restricts the unlimited number of open latent space parameters to a limited number of deep network parameters.

[49] Frigola, Chen, and Rasmussen, "Variational Gaussian process state-space models," 2014

[51] Frigola and Rasmussen, "Integrated Pre-Processing for Bayesian Nonlinear System Identification with Gaussian Processes," 2013

[50] Frigola, Lindsten, Schön, and Rasmussen, "Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC," 2013

[155] Turner, Deisenroth, and Rasmussen, "State-space inference and learning with Gaussian processes," 2010

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[44] Eleftheriadis, Nicholson, Deisenroth, and Hensman, "Identification of Gaussian Process State Space Models," 2017

By employing an autoregressive model, our method bypasses the challenging inference in latent space. At the same time, our method allows for analytically tractable optimization of the long-term trajectory likelihood similar to the inference in the SSM. In our experiments, we compare the performance of the proposed method to GP-NARX (and its PILCO variant).

In the deep learning community, powerful recurrent models have been published for sequential data [64]. However, these models are inherently deterministic. In order to utilize their long-term predictions for policy search or trajectory optimization, additional precautions are necessary to avoid regions of insufficient or inconclusive system knowledge. Additionally, training these models requires large amounts of data, which makes them less suitable for learning on physical systems with no prior knowledge. Recent work tries to alleviate these issues by introducing deep recurrent GPs [91] to leverage the data-efficiency of GP regression and to obtain uncertainty estimates. Similarly, [97] utilize variational auto-encoders to obtain predictive distributions for long-term system behavior. Whilst being conceptually interesting, extensions of these methods for data-efficient learning from scratch are yet to be proposed.

In contrast, SSMs are based on a compact, Markovian state representation. Furthermore, they allow for the direct application of many existing control algorithms, which rely on the explicit representation of the latent state. Exact solutions for state inference and model learning for linear Gaussian SSMs are given by the well known Kalman filter/smoother [70] and subspace identification [157]. In the case of non-linear latent state transition dynamics, both deterministic and probabilistic variants are active fields of research.

Deterministic variants such as Long Short-Term Memory (LSTM) models have been shown to be powerful representations for tasks such as natural language processing [158] or text understanding [144]. However, for the purpose of system identification and control, probabilistic predictions are often preferred to make model errors explicit [31]. A variety of stochastic deep recurrent models has been presented based on Stochastic Gradient Variational Bayes (SGVB) [13, 77, 165, 24, 5, 71, 48, 56]. These approaches, however, require large data sets to train very flexible models. The PR-SSM inference is inspired by the learning procedure in these deep recurrent models while employing GPs as a principled way of model regularization. Both procedures share the explicit unrolling of transition and observation model. Errors between the predicted and the observed system output are propagated back over time. Therefore, the transition dynamics has to be inferred, but the latent state (distribution) is given implicitly. This way, the challenging initialization and optimization of latent state variables is prevented. In contrast to deep recurrent models, the PR-SSM loss

[64] Hochreiter and Schmidhuber, "LSTM can solve hard long time lag problems," 1997

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[97] Mishra, Abbeel, and Mordatch, "Prediction and Control with Temporal Segment Models," 2017

[70] Kalman, "A new approach to linear filtering and prediction problems," 1960

[157] Van Overschee and De Moor, *Subspace identification for linear systems: Theory - Implementation - Applications*, 2012

[158] Venugopalan, Xu, Donahue, Rohrbach, Mooney, and Saenko, "Translating videos to natural language using deep recurrent neural networks," 2014

[144] Sutskever, Martens, and Hinton, "Generating text with recurrent neural networks," 2011

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

[13] Bayer and Osendorfer, "Learning stochastic recurrent networks," 2014

[77] Krishnan, Shalit, and Sontag, "Deep Kalman filters," 2015

[165] Watter, Springenberg, Boedecker, and Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," 2015

[24] Chung, Kastner, Dinh, Goel, Courville, and Bengio, "A recurrent latent variable model for sequential data," 2015

[5] Archer, Park, Buesing, Cunningham, and Paninski, "Black box variational inference for state space models," 2015

[71] Karl, Soelch, Bayer, and Smagt, "Deep variational bayes filters: Unsupervised learning of state space models from raw data," 2016

[48] Fraccaro, Sønderby, Paquet, and Winther, "Sequential neural models with stochastic layers," 2016

[56] Gemici, Hung, Santoro, Wayne, Mohamed, Rezende, Amos, and Lillicrap, "Generative Temporal Models with Memory," 2017

and model regularization is automatically obtained from the GP assumption. Furthermore, PR-SSMs obtain predictive distributions and the proposed initial state recognition model facilitates learning on shorter sub-trajectories and unstable systems, which is not possible in deep recurrent models.

GP-SSMs are a popular class of probabilistic SSMs [163, 75, 155, 50, 49, 44]. The use of GPs allows for a fully Bayesian treatment of the modeling problem resulting in an automatic complexity trade-off, which regularizes the learning problem. Filtering and smoothing in GP-SSMs has already been covered extensively: deterministic (e.g. linearization) as well as stochastic (e.g. particles) methods are presented in [75, 36]. These methods, however, assume an established system model, which is generally not available without prior knowledge. In this work, the latent state smoothing distribution is given implicitly and optimized jointly during model learning.

Approaches to probabilistic GP-SSMs mainly differ in their approximations to the model's joint distribution (e.g. when solving for the smoothing distribution or for the observation likelihood). One class of approaches aims to solve for the true distribution, which requires sample-based methods, e.g. Particle Markov Chain Monte Carlo (PMCMC), as in [50, 49]. These methods are close to exact and thus are able to represent temporal correlations. However, they are computationally inefficient and intractable for higher latent state dimensions or larger datasets. A second class of approaches is based on variational inference and mean field approximations in the latent state [91, 47]. These methods, however, operate on latent autoregressive models, which can be initialized by the observed output time series, such that the learned latent representation acts as a smoothed version of the observations. In Markovian latent spaces, no such prior information is available and therefore initialization is non-trivial. Model optimization based on mean field approximations empirically leads to highly suboptimal local solutions. Bridging the gap between both classes, recent methods strive to recover (temporal) latent state structure. In [44], a linear, time-varying latent state structure is enforced as a tractable compromise between the true non-linear dependencies and no dependencies as in mean field variational inference. However, to facilitate learning, a more complex recognition model over the linear time-varying dynamics is required. In contrast, the proposed PR-SSM can efficiently incorporate the true dynamics by combining sampling- and gradient-based learning.

[163] Wang, Fleet, and Hertzmann, "Gaussian process dynamical models for human motion," 2008

[75] Ko and Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models," 2009

[155] Turner, Deisenroth, and Rasmussen, "State-space inference and learning with Gaussian processes," 2010

[50] Frigola, Lindsten, Schön, and Rasmussen, "Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC," 2013

[49] Frigola, Chen, and Rasmussen, "Variational Gaussian process state-space models," 2014

[44] Eleftheriadis, Nicholson, Deisenroth, and Hensman, "Identification of Gaussian Process State Space Models," 2017

[75] Ko and Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models," 2009

[36] Deisenroth, Turner, Huber, Hanebeck, and Rasmussen, "Robust filtering and smoothing with Gaussian processes," 2012

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[47] Föll, Haasdonk, Hanselmann, and Ulmer, "Deep Recurrent Gaussian Process with Variational Sparse Spectrum Approximation," 2017

[44] Eleftheriadis, Nicholson, Deisenroth, and Hensman, "Identification of Gaussian Process State Space Models," 2017

6.3.3 Noisy Inputs

The standard GP framework assumes noise free input measurements and outputs corrupted by constant-variance Gaussian noise. In time-series modeling, where each observation is corrupted by noise, the noise free input assumption as well as the i.i.d. assumption are violated. In models as in (59) or (61), the model input is $[\boldsymbol{x}_t, \boldsymbol{u}_t]$. Whilst $\boldsymbol{u}_t$ is typically a known control signal, for $\boldsymbol{x}_t$ at best noisy observations are available on real systems.

Neglecting the input noise results in model bias, especially in situations of low signal to noise ratios. An example for model-bias is shown in the top plot of Fig. 22, which illustrates the shortcomings of optimizing a model on noisy measurements using one-step ahead predictions (cf. GP-NARX results). When performing full rollouts of a trajectory on such a learned one-step dynamics, the model-bias leads to implausible predictions. In contrast, methods that explicitly deal with noisy inputs (NIGP) and optimize long-term predictions (REVARB, MSGP) can improve prediction performance significantly.

One line of research addressing noisy input measurements is treating them as deterministic inputs and inflates the corresponding output variance, which leads to state-dependent, heteroscedastic noise models [59, 72]. In non-linear time series modeling, model input and output share the same noise distribution, which can be exploited by more tailored methods as for example Noise Input Gaussian Processes (NIGP) [94].

In MSGP and PR-SSM, the model is based on estimated true latent states $\hat{\boldsymbol{x}}_t$, which are noise free. Therefore, inputs for each time-step can be treated as noise-free variables and no special treatments of input noise is required. In case of MSGP, the noisy observations $\boldsymbol{y}_t$ only serve as rough initialization to guide the learning.

[59] Goldberg, Williams, and Bishop, "Regression with input-dependent noise: A Gaussian process treatment," 1997

[72] Kersting, Plagemann, Pfaff, and Burgard, "Most likely heteroscedastic Gaussian process regression," 2007

[94] McHutchon and Rasmussen, "Gaussian process training with input noise," 2011

7 Multi-Step Gaussian Process Models

This chapter presents a novel model learning technique for Model-Based Policy Search (MBPS). In particular, the main contribution is the development of a new model optimization objective for learning latent autoregressive GP dynamics models by maximizing the likelihood of complete rollouts under a given policy. The proposed method is tailored, but not limited, to model learning for policy search. The effectiveness of our approach in both scenarios, model learning and policy search, is confirmed through a number of benchmark evaluations, as well as a comparison on a real robotic system.

7.1 Multi-Step Gaussian Processes for RL

In order to address the issues identified in Sec. 6.3, we propose an improved model learning technique aiming at increasing the performance of MBPS methods.

7.1.1 Main Idea

Our contribution is based on three core insights:

(i) Optimize for long-term predictions of closed loop behavior

Models employed for finite horizon policy optimization should capture the closed-loop, long-term behavior of the system given a feedback policy. When optimizing models for one-step-ahead predictions, the model learning process is oftentimes derailed by effects such as system noise, causing small errors to accumulate. If, however, the model is trained to predict full trajectories and accumulated errors are backpropagated into the predicted system states during model optimization, the resulting model becomes better at capturing the relevant long-term system dynamics.

(ii) Restrict model learning to feasible policy control outputs

When learning models with the purpose of policy search, we frequently have access to the actual policy that generated the data. Thus, by replacing the inputs to the model $\boldsymbol{u}_t$ by the inputs generated by the specific policy $\boldsymbol{u}_t = \pi(\boldsymbol{y}_t, \theta_\pi)$, we are able to optimize the model for predicting long-term system development based on how the pol-

icy would act for the predicted system behavior. In contrast to general system identification methods, which usually aim at learning a model over the entire space of arbitrary control input sequences, our method can learn a model tailored to the smaller control input manifold spanned by the class of considered feedback policies. Given a stabilizing policy, the incorporation of the feedback signal into the long-term prediction can further compensate for model errors and therefore stabilize model learning.

(iii) Learn model specifically for the approximations in the later policy search step

Model-learning for long-term predictions and finite-horizon policy search are strongly related problems. If approximations are necessary in the model evaluation for the subsequent policy search step, they should already be taken into account in the model learning step. Instead of optimizing for an arbitrary prediction error measure, we directly optimize a quantity such as long-term predictive distributions, which is required for policy search.

These three ideas taken together enable efficient model-based learning of policies on real systems with latent states and noisy observations from scratch, which proved to be a major limitation for other state-of-the-art frameworks. In the following sections, we elaborate on how we implement these ideas in a Bayesian model learning framework, which we refer to as Multi-Step Gaussian Processes (MSGP).

7.1.2 Trajectory Likelihood Optimization

MSGP is built around a generative model for the distribution of observations $\boldsymbol{y}_{0:T}$ conditioned on the system's initial state $\boldsymbol{x}_0$ and *either* the sequence of actions $\boldsymbol{u}_{0:T}$ *or* the applied control policy, given by θ_π. Therefore, MSGP can learn from data observed in open-loop as well as closed-loop experiments. Ultimately, for a good long-term prediction model, all trajectories observed on the real system should be likely under these predictive distributions. The MSGP model is learned by directly minimizing the negative log-likelihood. For a set D_{ff} of trajectories based on a feedforward (ff) control signals $\boldsymbol{u}_{0:T_i}$, and a different set D_{fb} of trajectories originating from feedback (fb) policies $\theta_{\pi,j}$, the model loss is composed from

$$L_{\text{ff}}(\theta) = -\sum_{\tau_i \in D_{\text{ff}}} \log p(\boldsymbol{y}_{0:T_i} | \boldsymbol{u}_{0:T_i}, x_{0,i}) \,, \tag{65}$$

and

$$L_{\text{fb}}(\theta) = -\sum_{\tau_j \in D_{\text{ff}}} \log p(\boldsymbol{y}_{0:T_j} | \theta_{\pi,j}, x_{0,j}) \,. \tag{66}$$

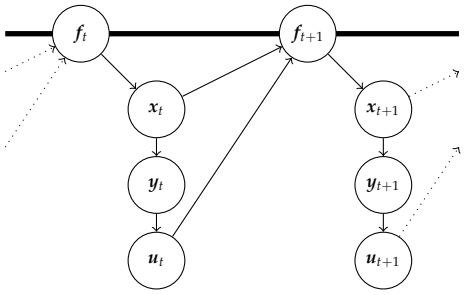


Figure 21: General system description. The solid black line indicates the latent states being jointly Gaussian under a Gaussian Process prior. This model can be extended to a latent autoregressive model by including historic states and inputs into the transition model.

The generative MSGP model is parametrized by θ (made precise below). MSGP approximates (65) and (66) such that temporal correlations between the current timestep and all previous timesteps are maintained. Model errors are therefore accumulated over time and backpropagated during model optimization to obtain good long-term predictions. The optimal model parameters θ^*, taking into account all open-loop and closed-loop trajectories, is obtained as

$$\theta^* = \arg\min_{\theta} L_{\text{ff}}(\theta) + L_{\text{fb}}(\theta)\,. \tag{67}$$

The MSGP model is a specific version of the SSM in (64). The observation function g is given, without loss of generality, by the identity function (cf. [50]). The latent state transition model f is an autoregressive GP model. Thereby, we sidestep the challenging inference in a truly latent state space with unknown dimensionality and represent the missing information by taking into account historic state information. As in (64), the model includes Gaussian process and observation noise. A graphical model of the generative model is shown in Fig. 21.

[50] Frigola, Lindsten, Schön, and Rasmussen, "Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC," 2013

In contrast to standard GP dynamics models, which are trained on one-step-ahead predictions given measured training data, the latent system state is not directly measurable. Following work by [155], the latent state GP is therefore parametrized by m inducing inputs $\bar{\boldsymbol{X}} = [\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_m]^T$ and targets $\bar{\boldsymbol{y}} = [\bar{\boldsymbol{y}}_1, \ldots, \bar{\boldsymbol{y}}_m]^T$. Inducing points are artificial GP inputs and targets which can be optimized to capture the system behavior. This approach is commonly employed for learning sparse GP representations [142, 125]. The GP inputs are of the autoregressive form in (62). The GP hyperparameters θ_{hyp}, inducing inputs $\bar{\boldsymbol{X}}$ and targets $\bar{\boldsymbol{y}}$, as well as process and observation noise variances are parameters of the generative model, which are jointly optimized. The parameter vector is therefore given by $\theta = (\theta_{\text{hyp}}, \bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_m, \bar{\boldsymbol{y}}_1, \ldots, \bar{\boldsymbol{y}}_m, \Sigma_{\text{x}}, \Sigma_{\text{y}})$.

[155] Turner, Deisenroth, and Rasmussen, "State-space inference and learning with Gaussian processes," 2010

[142] Snelson and Ghahramani, "Sparse Gaussian processes using pseudo-inputs," 2006

[125] Quinonero-Candela, Rasmussen, and Williams, "Approximation methods for Gaussian process regression," 2007

Standard GP models are usually trained by maximizing the data log likelihood [168] given by

$$\log p(\boldsymbol{y}|\boldsymbol{X}) = -\frac{1}{2}\boldsymbol{y}^T(K + \sigma_{\text{n}}^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K + \sigma_{\text{n}}^2 I| - \frac{n}{2}\log 2\pi\,, \tag{68}$$

[168] Williams and Rasmussen, *Gaussian processes for machine learning*, 2005

where $\boldsymbol{y}$ and $\boldsymbol{X}$ are training data points obtained from measured system data with input dimensionality n. These standard models only account for Gaussian output noise given by σ_n. The GP covariance matrix K is obtained as $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \forall \boldsymbol{x}_i, \boldsymbol{x}_j \in \bar{\boldsymbol{X}}$ for a given kernel function k. This objective automatically trades data-fit (first term) for model complexity (second term). In contrast, the long-term trajectory likelihood objectives in (65) and (66)) are not automatically penalizing the model complexity of the latent dynamics model. Instead, all inducing inputs and outputs can be modified independently. To avoid overfitting of the latent state model, we add the model complexity penalty given by

$$L_{\text{comp}}(\theta) = -\frac{1}{2} \log |K + \sigma_{\text{x}}^2 I| \,, \tag{69}$$

to the long-term likelihood term in (65) and (66). The full MSGP optimization objective is therefore given by

$$\theta^* = \arg\min_{\theta} L_{\text{ff}}(\theta) + L_{\text{fb}}(\theta) + L_{\text{comp}}(\theta) \tag{70}$$

7.1.3 Marginal Observation Distribution

In the following, we summarize the necessary computations to obtain the conditional observation distributions from (65) and (66). The approximations are inspired by the long-term prediction method utilized in the policy search framework PILCO [31]. For clarity, we will drop the notion of explicitly conditioning on the initial state $\boldsymbol{x}_0$, the input sequence $\boldsymbol{u}_{0:T}$, the policy parameters θ_π, and the model parameters θ. We also drop the time indices and denote the full trajectories of states, GP-predictions, outputs, and inputs over the prediction horizon by $\boldsymbol{x}$, $\boldsymbol{f}$, $\boldsymbol{y}$ and $\boldsymbol{u}$, respectively. Due to the Markovian state $\boldsymbol{x}_t$, the joint distribution of $p(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{y}, \boldsymbol{u})$ factorizes such that the marginal observation distribution is given by

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

$$\begin{aligned} p(\boldsymbol{y}) = \int p(\boldsymbol{x}_0) \prod_{t=1}^{T} & p(\boldsymbol{y}_t|\boldsymbol{x}_t) p(\boldsymbol{x}_t|\boldsymbol{f}_t) p(\boldsymbol{f}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) \\ & p(\boldsymbol{u}_{t-1}|\boldsymbol{y}_{t-1}; \theta_\pi) \mathrm{d}\boldsymbol{u} \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{f} \,. \end{aligned} \tag{71}$$

We assume a Gaussian distribution for the initial state

$$p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0, \Sigma_0) \,. \tag{72}$$

Starting from $p(\boldsymbol{x}_0)$, we can repeatedly apply the following steps to obtain the full marginal observation distribution from (71).

Observation Model

Based on the Gaussian approximation of the latent state $p(\boldsymbol{x}_t)$ in every prediction step the conditional observation distribution is obtained as

$$p(\boldsymbol{y}_t|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{x}_t, \Sigma_\mathrm{y}) \,. \tag{73}$$

Policy

The policy $\boldsymbol{u}_t = \pi(\boldsymbol{y}_t; \theta_\pi)$ is evaluated on the current observation $\boldsymbol{y}_t$. To obtain the input to the dynamics model, we compute the joint distribution of the current state and the policy output $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ as given by

$$p(\boldsymbol{x}_t, \boldsymbol{u}_t) = \int p(\boldsymbol{u}_t|\boldsymbol{y}_t, \theta_\pi) p(\boldsymbol{y}_t|\boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{y}_t \,. \tag{74}$$

This integral can be computed exactly for Gaussian observation models (cf. (73)) and linear policies. For non-linear policies (e.g. RBF networks or neural networks), we revert to a moment matching approximation of the joint probability distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$.

Dynamics Model

As detailed before, the latent state dynamics is modeled by an autoregressive GP based on m inducing inputs. Each latent state dimension is modeled independently by a single output GP. We employ the popular Squared Exponential (SE) kernel [168] given by

[168] Williams and Rasmussen, *Gaussian processes for machine learning*, 2005

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_\mathrm{f}^2 \exp(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \Lambda (\boldsymbol{x}_i - \boldsymbol{x}_j)) \,, \tag{75}$$

with lengthscales $\Lambda = \mathrm{diag}(1/l_1^2, \ldots 1/l_D^2)$ and signal variance σ_f^2. The predictive distribution for the next latent state f_{t+1} is then given by

$$p(f_{t+1}|\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}) \sim \mathcal{N}(\mu_{f_{t+1}}, \sigma_{f_{t+1}}^2) \,, \tag{76}$$

where the mean and the variance are given by

$$\mu_{f_{t+1}} = k(\bar{\boldsymbol{x}}_t)^T K^{-1} \bar{\boldsymbol{y}} \,, \quad \sigma_{f_{t+1}}^2 = k(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{x}}_t) - k(\bar{\boldsymbol{x}}_t)^T K^{-1} k(\bar{\boldsymbol{x}}_t) \,, \tag{77}$$

with $k(\bar{\boldsymbol{x}}_t) = [k(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{x}}_1), \ldots, k(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{x}}_m)]^T$.

Propagation of Uncertainty

The input to the dynamics model is jointly Gaussian. In general, the marginal distribution for propagating a Gaussian distributed input $p(\bar{\boldsymbol{x}}_t)$ through an arbitrary non-linear function $f(x)$, as given by

$$p(f_{t+1}) = \int p(f_{t+1}|\bar{\boldsymbol{x}}_t) p(\bar{\boldsymbol{x}}_t) d\bar{\boldsymbol{x}}_t \,, \tag{78}$$

is non-Gaussian. In order to obtain analytic gradients for the propagation step, we utilize Moment Matching (MM). First and second moment of the predictive distribution can be calculated in closed form if a Gaussian-like kernel is employed as derived in [124]. The approximated predictive distribution for a Gaussian distributed input $\bar{\boldsymbol{x}}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\bar{x}}, \boldsymbol{\Sigma}_{\bar{x}})$ is given by

[124] Quinonero-Candela, Girard, and Rasmussen, *Prediction at an uncertain input for Gaussian processes and relevance vector machines-application to multiple-step ahead time-series forecasting*, 2002

$$p(f_{t+1}) = \mathcal{N}(\mu_{f_{t+1,\mathrm{MM}}}, \sigma_{f_{t+1,\mathrm{MM}}}), \tag{79}$$

The expressions for mean and variance, and references to detailed derivations of the moment matching formalism can be found in the supplementary material. By applying the steps from (73) to (79) repeatedly over the prediction horizon, we obtain the full predictive distribution (65) and (66). Every prediction step is therefore dependent on all previous timesteps such that model errors are accumulated and consequently backpropagated during model optimization to improve the quality of long-term predictions. Due to the choice of a GP latent space dynamics model and the moment matching approximation for the predictive trajectory distribution, gradients for the loss with respect to the model parameters θ can be obtained analytically. We can then optimize the model parameters using standard gradient descent methods.

7.1.4 Relation between Model Learning and Policy Search

Both, MBPS and model learning can be expressed as

$$\theta_* = \arg\min_{\theta} J(p(\boldsymbol{y}_{0:T} \mid \boldsymbol{x}_0, \theta)). \tag{80}$$

In MSGP, the loss function J used for model learning is the negative log-likelihood of the observed trajectories and the model's open parameters θ are given by the GP parameters. In contrast, for policy search, the loss J could be a quadratic penalty for deviations from a desired trajectory and the model's parameters θ are given by the policy parameters. In both problems, an optimization based on the predictive distribution $p(\boldsymbol{y}_{0:T})$ as given by (71) has to be carried out. Since (71) is generally not analytically tractable, approximations are made in model learning and policy search. Ideally, the approximations made in the subsequent policy search step should be incorporated in the model learning part already. MSGP is therefore tailored for the Probabilistic Inference for Learning COntrol (PILCO) method [31], since MSGP employs the same approximations to the predictive distribution (65) and (66) as utilized in PILCO. Since we changed the GP training procedure but not the GP-model structure itself, we end up with a standard GP, which can be directly employed within the PILCO framework.

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

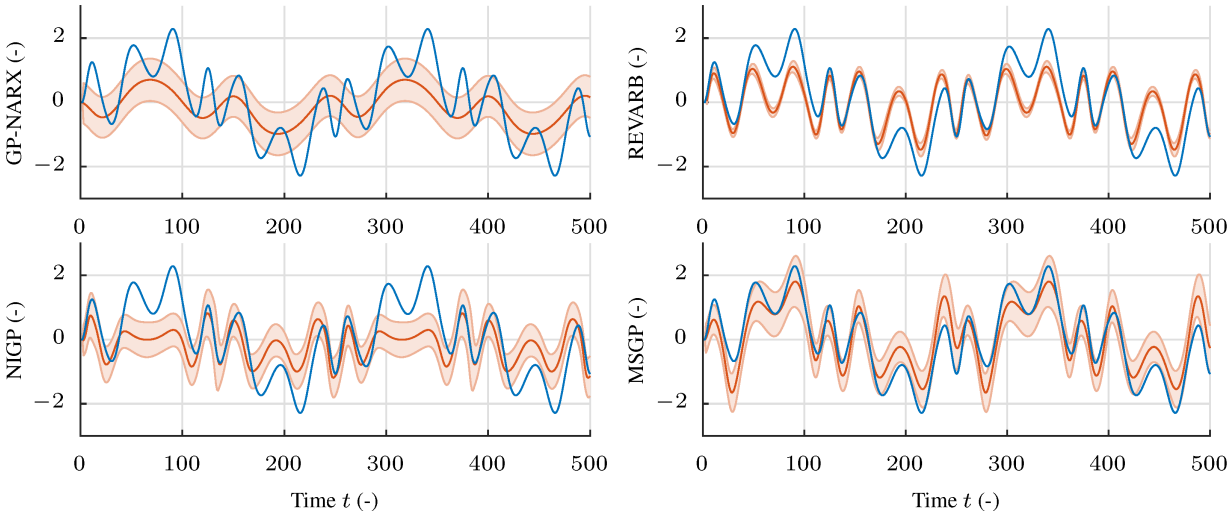


Figure 22: Comparison of the free simulation on the synthetic system 4 test dataset. The predictive distribution of GP-NARX, NIGP, REVARB and the proposed MSGP are shown (mean ——, +/- 2 standard deviation ▭) together with the true, latent state of the system (——).

PILCO is a method which propagates a parametrized policy through a probabilistic model in order to compute the gradients of the policy parameters with respect to the cost function. To learn about the system, PILCO iterates between deriving an optimal policy for the learned system model and running the resulting policy on the real system to acquire more data. This method has been successfully applied to real-world systems like low-cost manipulators [35] and throttle valve control [18], and extensions are available for multiple task learning [35] or training of multivariate PID structures [41].

7.2 Experimental Evaluation

In the previous sections, MSGP, a novel model learning technique has been proposed, which is particularly tailored to the needs of model-based RL. Thus, the upcoming experimental evaluation of MSGP is twofold: (i) Evaluation of the model learning capabilities in comparison to existing system identification methods on a set of well-known benchmarks. (ii) Comparison of the MSGP policy search performance to a state-of-the-art method.

7.2.1 Model Learning Benchmark

For evaluating the system identification performance, the proposed MSGP method is compared to state-of-the-art model learning methods from the literature: (i) GP-NARX [76], (ii) PILCO's GP-NARX [31], (iii) NIGP [94], and (iv) REVARB [91]. The benchmark is conducted on five synthetic and five real-world datasets. To compare the quality of the learned system dynamics models, we compute the free simulation

[35] Deisenroth, Rasmussen, and Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," 2011

[18] Bischoff, Nguyen-Tuong, Koller, Markert, and Knoll, "Learning Throttle Valve Control Using Policy Search," 2013

[41] Doerr, Nguyen-Tuong, Marco, Schaal, and Trimpe, "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers," 2017

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

[94] McHutchon and Rasmussen, "Gaussian process training with input noise," 2011

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

Task	GP-NARX	GP-NARX PILCO	NIGP	REVARB 1	REVARB 2	MSGP
Synthetic system 1	0.2265	0.3511	0.2453	0.2129	0.1717	0.3805
Synthetic system 2	0.3535	0.3467	0.4100	0.3395	0.3726	0.3424
Synthetic system 3	0.1572	0.1487	0.1950	0.3657	0.1695	0.1341
Synthetic system 4	0.5711	0.6016	0.7814	0.6978	0.4554	0.3287
Synthetic system 5	0.0164	0.0150	0.0101	0.1193	0.0035	0.0143
Actuator	0.6376	1.3767	0.6483	0.4328	0.5522	0.7124
Balancing	0.0773	0.0881	0.0776	0.1360	0.0732	0.0599
Drives	0.6888	0.7024	0.7525	0.7463	0.5620	0.4217
Furnace	1.1996	1.2012	1.1949	1.3434	1.9569	1.2013
Dryer	0.2856	0.2956	0.2802	0.8764	0.1286	0.1523

Table 1: Comparison of model learning methods on synthetic (first 5 rows) and real-world (last 5 rows) benchmark examples. The RMSE result is given for the free simulation on the noise free test dataset. The best model (——) and the second best model (- - -) are indicated for each dataset. The background color indicates best ( ) to worst ( ) model performance. Best viewed in color.

for each model. Each benchmark dataset comprises a system's input-output trajectory $\tau = (u_0, y_0), \ldots, (u_T, y_T)$ up to a rollout horizon T. Some initial input and output data is required for the NARX models' history to predict into the future. The models thus start to predict from $\tilde{x}_0 = [y_{l_y}, \ldots, y_0, u_{l_u}, \ldots, u_0]^T$ given their specific requirements for input and output history l_u and l_y respectively. Details about the benchmark datasets and the setup of the individual benchmark methods can be found in Appendix B.

Synthetic Benchmark Datasets

The resulting RMSEs are stated for each method and dataset in Tab. 1 The results show that learning methods based on long-term predictions (REVARB, MSGP) tend to outperform the one-step-ahead models (GP-NARX, GP-NARX-PILCO, NIGP). Overall, the results show that the performance of MSGP is comparable to the one of REVARB methods, however, none of the methods outperforms the others in all cases. The presented results are based on the moment matching approximation to obtain long-term predictive distributions. While moment matching is essential to retrieve an estimate of the prediction's uncertainty, it also introduces additional errors. Propagating only the mean prediction results in better RMSE results for some benchmarks as reported in [93]. A visualization of the free running prediction results on the synthetic system 4 dataset is shown Fig. 22. The model bias introduced by the input noise is clearly visible for the standard GP-NARX model (top plot); the true dynamics is not captured by this model. In contrast, NIGP, REVARB and MSGP capture the dynamics much better. However, REVARB is clearly underestimating the un-

[93] Mattos, Damianou, Barreto, and Lawrence, "Latent Autoregressive Gaussian Processes Models for Robust System Identification," 2016

certainty about the underlying system and both NIGP and REVARB cannot properly fit the high and low peaks, i.e. the learned length-scales are too short to generalize well on the brink of the state space covered by training data.

Real-World Benchmark Datasets

The results in Tab. 1 show that (like for the synthetic benchmarks) the methods based on long-term predictions (MSGP and REVARB) generally outperform the methods based on one-step-ahead predictions. While there is still no clear overall winner, MSGP seems to perform slightly better for the real-world data sets than for the synthetic ones.

7.2.2 Policy Search

So far, we evaluated the model learning performance as measured by the root mean squared error of a long-term prediction on test datasets. However, it is unclear how to quantify model quality with respect to the purpose of policy search. For the purpose of learning feedback policies, good models are not necessarily required to obtain the best predictions for an arbitrary feedforward signal. However, this is usually the objective in model learning methods.

To demonstrate the policy learning properties of the proposed MSGP method combined with PILCO, we investigate the problem of learning to balance an inverted pendulum, which is a well-known benchmark scenario in control and reinforcement learning [4, 90].

Experimental Setup

We employ a humanoid upper-body robot platform as shown in Fig. 23. The system outputs considered for balancing are the pendulum and end-effector positions, and the input is given by the end-effector acceleration. Commanded accelerations are internally translated into joint torques by an existing inverse dynamics model. The policy is composed from a PID feedback controller on the end-effector and a PD controller on the pendulum angle, similar to the experimental setup in [41].

Despite the simplicity of the task of balancing an inverted pendulum, the robotic arm (cf. Fig. 23) introduces all real-world challenges like noise, delays, and latent states as discussed in Sec. 6.3. These problems have been tackled before [41] using standard PILCO, which required additional prior system knowledge, manual tuning and a GP-NARX model to deal with partial observability. Given the MSGP framework, we can learn in this scenario from scratch without the need for manual tuning and injection of prior knowledge.

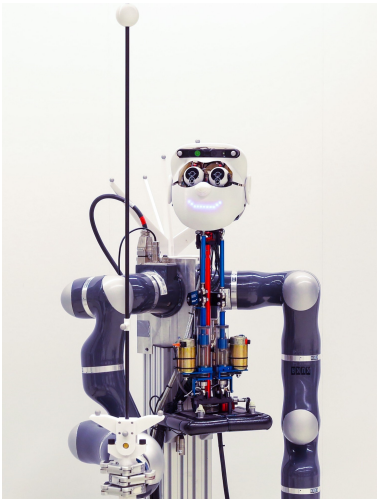
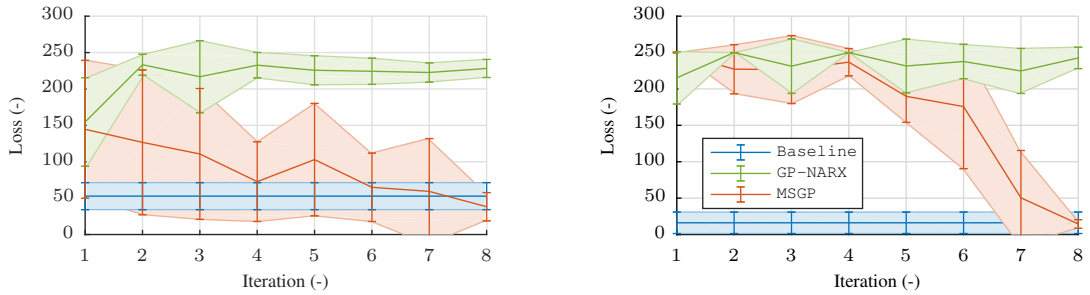


Figure 23: A humanoid upper-body robot learning to balance an inverted pendulum. For this task, improved data-efficiency and therefore faster policy learning is demonstrated based on the presented long-term optimization model-learning procedure.

[4] Anderson, "Learning to control an inverted pendulum using neural networks," 1989

[90] Marco, Hennig, Bohg, Schaal, and Trimpe, "Automatic LQR Tuning Based on Gaussian Process Global Optimization," 2016

[41] Doerr, Nguyen-Tuong, Marco, Schaal, and Trimpe, "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers," 2017



(a) Predicted rollout loss.

(b) Actual rollout loss.

Figure 24: Iterative learning to balance the inverted pendulum without prior knowledge based on the GP-NARX model as used in standard PILCO ( ) and the proposed MSGP model ( ). As baseline, the performance of an optimal policy ( ) based on a fixed GP-NARX model is shown. In contrast to the iterative learning procedures, this baseline model was trained on a much larger dataset that incorporates random, instable and stable rollouts. MSGP improves the long-term model predictions such that policy search with no prior domain knowledge is feasible. The graphs represent the outcome of five independent learning runs, depicted as the average (——) with +/- 1 standard deviations ( ).

For this task, most model learning methods discussed in Sec 7.2.1 are not directly applicable. In order to efficiently optimize the expected policy cost, analytic policy gradients must be available for a long-term prediction. Since MSGP training results in a standard GP-NARX model, we can directly employ the existing PILCO framework. Deriving and implementing analytic policy gradients for the more involved deep and recurrent GP models like REVARB [91] is challenging and not straight-forward to implement in existing model-based RL frameworks. Additionally, the REVARB inference is implemented for single trajectories only, whereas the iterative model learning procedure has to learn from multiple experimental rollouts. Thus, policy search is directly applicable only to GP-NARX and MSGP.

Experimental Results

We compare policy search results based on the proposed MSGP model to standard PILCO using a GP-NARX model. Both methods are initiated with data from four rollouts based on smooth random signals. The distribution of policy costs over learning iterations (mean +/- one standard deviation) as obtained from five independent learning experiments is visualized in Fig. 24. The left plot shows the predicted cost, whereas the right plot shows the actual cost experienced for the rollout of the policy on the real system. We employ the standard saturated cost function in PILCO, penalizing both system outputs and inputs.

PILCO has been demonstrated to successfully learn pendulum swing-up on a lab test-bench [31]. On our robotic setup, due to the challenges discussed in Sec. 6.3, neither a standard GP nor a GP-NARX dynamics model are sufficient to learn a good predictive model or a stabilizing policy (cf. GP-NARX results in Fig. 24). In contrast, the MSGP model optimization is able to learn a successful model and policy from scratch without requiring prior system knowledge. Stable balancing is achieved after six to eight policy iterations, which corresponds to roughly 30 seconds of interaction with the real system.

To demonstrate the general ability of the GP-NARX model to represent the system's latent dynamics, we report in Fig. 24 (blue) also

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

the baseline policy obtained offline from a much larger, separately recorded dataset. For this, policy search was performed on a GP-NARX model that was trained on a larger dataset including multiple random, unstable, and stable rollouts. The baseline model is clearly sufficient to find a stabilizing policy. This indicates that progress in the iterative PILCO procedure is not limited by the expressivity of the GP-NARX model, but by the model optimization procedure itself. MSGP-based PILCO is able to robustly learn a stabilizing policy iteratively and with a much smaller dataset than the baseline.

7.3 Summary

In this chapter, we introduced MSGP, a lightweight model learning technique to improve long-term predictions, which is applicable to model-based policy search frameworks in a straightforward way. The methodology can be interpreted as optimizing a recurrent, latent GP-NARX dynamics model by maximizing the likelihood of observed system trajectories. In contrast to classic system identification methods, we explicitly optimize the model with respect to its applicability in policy search. Therefore, long-term predictions take into account the dependency on the employed policy and the errors introduced by the approximations made in the policy search step.

Benchmark results are given for artificial and real-world system identification tasks, comparing MSGP to several state-of-the-art nonlinear system identification methods, including recurrent (deep) GPs. While MSGP is competitive to state-of-the-art methods for system identification, it clearly outperforms existing approaches on the robot policy search task. Using MSGP, iterative learning of a challenging robotic task from scratch is possible, in contrast to the default PILCO implementation.

One computational challenge in MSGP is the calculation of the full predictive distribution for all training trajectories over the full trajectory length in each function and gradient evaluation for every model learning iteration. The model optimization is therefore expensive on big datasets. Runtimes on the presented datasets range from minutes (artificial/real-world datasets) up to two hours for the last iterations on the robotic application. Potential speed improvements can be obtained by more efficient autodifferentiation (e.g. using Tensorflow), parallelization across trajectories and stochastic model update steps utilizing minibatches of subtrajectories. The optimization problem could possibly be relaxed by adding free latent state variables as additional optimization parameters. The system dynamics then becomes an additional optimization constraint.

By switching from the one-step-ahead prediction methods to long-term optimization methods, we would expect to improve the robustness against varying sampling frequencies, as well as delays and disturbances due to non-measured latent states. Moreover, tailoring model learning to the class of policies employed in the subsequent policy search step should improve the learning performance. These are promising directions for future research.

8

Probabilistic Recurrent State-Space Model

In the previous sections, the novel MSGP method has been introduced for learning long-term predictive models, in particular for policy search applications and real-world systems. As demonstrated (cf. Sec.7.2), MSGP improves on state-of-the-art model-learning techniques in RL. The impressive performance on real-systems can be attributed to the ability of accounting for noisy data and unobserved dynamics. MSGP facilitates this by means of optimizing the long-term, latent states of a GP-NARX model (cf. Sec.7.1.1).

As discussed in the summary section 7.3, with MSGP, several drawbacks remained open for future work. In this section, a second model learning framework, Probabilistic Recurrent State-Space-Models (PR-SSM) is introduced to address the shortcomings of MSGP. Most fundamentally, the efficiency of MSGP was facilitated by Gaussian approximations in each time-step. This assumption, however, implies strong limitations of the MSGP framework. In particular, skewed or multi-modal distributions that might easily arise from propagating uncertainty in non-linear dynamics models (cf. [57]), can not be represented in the MSGP per design.

[57] Girard and Rasmussen, "Multiple-step ahead prediction for non linear dynamic systems–a gaussian process treatment with propagation of the uncertainty,"

With PR-SSM, one of the goals is to accurately represent complex, i.e., non-Gaussian latent state distributions. Computing the full trajectory distribution in non-linear state-space models is however not analytically tractable. The main challenge is thus in finding an appropriate trade-off between the accuracy of the posterior approximation and the computational tractability.

The PR-SSM model is defined in Sec. 8.1, followed by the novel inference scheme in Sec. 8.2. Extensions to facilitate large-scale data are discussed in Sec. 8.3. Experimental results are detailed in Sec. 8.4. This section concludes with a summary of the presented PR-SSM method and an outlook on possible future topics.

8.1 PR-SSM Model Definition

The PR-SSM is built upon a GP prior on the transition function $f(\cdot)$ and a parametric observation model $g(\cdot)$. This is a common model structure, which can be assumed without loss of generality over (63), since any observation model can be absorbed into a sufficiently large latent state [52]. Eliminating the non-parametric observation model, however, mitigates the problem of *'severe non-identifiability'* between

[52] Frigola-Alcade, "Bayesian time series learning with Gaussian processes," 2015

transition model $f(\cdot)$ and observation model $g(\cdot)$ [49]. Independent GP priors are employed for each latent state dimension d given individual inducing points $\boldsymbol{\zeta}_d$ and $\boldsymbol{z}_d$.

[49] Frigola, Chen, and Rasmussen, "Variational Gaussian process state-space models," 2014

In the following derivations, the system's latent state, input and output at time t are denoted by $\boldsymbol{x}_t \in \mathbb{R}^{D_x}$, $\boldsymbol{u}_t \in \mathbb{R}^{D_u}$, and $\boldsymbol{y}_t \in \mathbb{R}^{D_y}$, respectively. The shorthand $\hat{\boldsymbol{x}}_t = (\boldsymbol{x}_t, \boldsymbol{u}_t)$ denotes the transition model's input at time t. The output of the transition model is denoted by $\boldsymbol{f}_{t+1} = f(\hat{\boldsymbol{x}}_t)$. A time series of observations from time a to time b (including) is abbreviated by $\boldsymbol{y}_{a:b}$ (analogously for the other model variables).

The joint distribution of all PR-SSM random variables is given by

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z}) = \left[\prod_{t=1}^{T} p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\right] p(\boldsymbol{x}_1) p(\boldsymbol{z}) \left[\prod_{t=2}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{f}_t) p(\boldsymbol{f}_t \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z})\right], \tag{81}$$

where $p(\boldsymbol{f}_t \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}) = \prod_{d=1}^{D_x} p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d)$ and $\boldsymbol{z} \equiv [\boldsymbol{z}_1, \dots \boldsymbol{z}_{D_x}]$. A graphical model of the resulting PR-SSM is shown in Fig. 25.

The individual contributions to (81) are given by the observation model and the transition model, which are now described in detail. The observation model is governed by

$$p(\boldsymbol{y}_t \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{y}_t \mid g(\boldsymbol{x}_t), \operatorname{diag}(\sigma_{y,1}^2, \dots, \sigma_{y,D_y}^2)), \tag{82}$$

In particular, in our experiments, we employed a parametric observation model

$$g(\boldsymbol{x}_t) = \boldsymbol{C}\boldsymbol{x}_t\,. \tag{83}$$

The matrix $\boldsymbol{C}$ is chosen to select the D_y first entries of $\boldsymbol{x}_t$ by defining $\boldsymbol{C} := [\boldsymbol{I}, \boldsymbol{0}] \in \mathbb{R}^{D_y \times D_x}$ with $\boldsymbol{I}$ being the identity matrix. This model is suitable for observation spaces that are low-dimensional compared to the latent state dimensionality, i.e. $D_y < D_x$, which is often the case for physical systems with a restricted number of sensors. The first D_y latent state dimensions can therefore be interpreted as noise free sensor measurements. For high-dimensional observation spaces (e.g. images), a more involved, given observation model (e.g. a pre-trained neural network) may be seamlessly incorporated into the presented framework as long as $g(\cdot)$ is differentiable.

Process noise is modeled as

$$p(\boldsymbol{x}_t \mid \boldsymbol{f}_t) = \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{f}_t, \operatorname{diag}(\sigma_{x,1}^2, \dots, \sigma_{x,D_x}^2))\,. \tag{84}$$

The transition dynamics is described independently for each latent state dimension d by $p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d) p(\boldsymbol{z}_d)$. This probability is given by the sparse GP prior (17) and predictive distribution (16), where

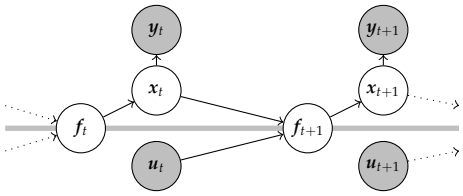


Figure 25: Graphical model of the PR-SSM. Gray nodes are observed variables in contrast to latent variables in white nodes. Thick lines indicate variables, which are jointly Gaussian under a GP prior.

$x^* = \hat{\boldsymbol{x}}_t$ and $f^* = f_{t,d}$. The initial system state distribution $p(\boldsymbol{x}_1)$ is unknown and has to be estimated.

8.2 PR-SSM Inference

Computing the log likelihood or a posterior based on (81) is generally intractable due to the nonlinear GP dynamics model in the latent state. However, the log marginal likelihood $\log p(\boldsymbol{y}_{1:T})$ (evidence) can be bounded from below by the Evidence Lower BOound (ELBO) [19]. This ELBO is derived via Jensen's inequality by introducing a computationally simpler, variational distribution $q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})$ to approximate the model's true posterior distribution $p(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z} \mid \boldsymbol{y}_{1:T})$ (cf. eq. (81)). In contrast to previous work [49, 91, 44] , the proposed approximation explicitly incorporates the true temporal correlations in the latent state, whilst being scalable to large datasets. Previous work based on sequential Monte Carlo methods [50, 149] already allowed for temporal correlations but required computationally challenging resampling in each timestep. The inference scheme is inspired by doubly stochastic variational inference for deep GPs as presented in [131].

8.2.1 Variational Sparse GP

PR-SSM employs a variational sparse GP [152] based on a variational distribution $q(\boldsymbol{z})$ on the GP's inducing outputs as previously used in [49, 44]. Eliminating the inducing outputs, however, results in dependencies between inducing outputs and data which, in turn, leads to a complexity of $\mathcal{O}(NP^2)$, where N is the number of data points and P the number of inducing points [152]. Unfortunately, this complexity is still prohibitive for large datasets. Therefore, we resort to an explicit representation of the variational distribution over inducing outputs as previously proposed in [63]. This explicit representation enables scalability by utilizing stochastic gradient-based optimization since individual GP predictions become independent given the explicit inducing points. Following a mean-field variational approximation the inducing output distribution is given as $q(\boldsymbol{z}) = \prod_{d=1}^{D_x} \mathcal{N}(\boldsymbol{z}_d \mid \mu_d, \Sigma_d)$

[19] Blei, Kucukelbir, and McAuliffe, "Variational inference: A review for statisticians," 2017

[49] Frigola, Chen, and Rasmussen, "Variational Gaussian process state-space models," 2014

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[44] Eleftheriadis, Nicholson, Deisenroth, and Hensman, "Identification of Gaussian Process State Space Models," 2017

[50] Frigola, Lindsten, Schön, and Rasmussen, "Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC," 2013

[149] Svensson and Schön, "A flexible state–space model for learning nonlinear dynamical systems," 2017

[131] Salimbeni and Deisenroth, "Doubly Stochastic Variational Inference for Deep Gaussian Processes," 2017

[152] Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," 2009

[63] Hensman, Fusi, and Lawrence, "Gaussian processes for big data," 2013

for each latent state dimension d with diagonal variance Σ_d. Marginalizing out the inducing outputs, the GP predictive distribution is obtained as Gaussian with mean and variance given by

$$\begin{aligned} \mu &= m_{\hat{x}_t} + \boldsymbol{\alpha}(\hat{\boldsymbol{x}}_t)(\mu_d - m_{\zeta_d}) \,, \\ \sigma^2 &= k_{\hat{x}_t,\hat{x}_t} - \boldsymbol{\alpha}(\hat{\boldsymbol{x}}_t)(K_{\zeta_d,\zeta_d} - \Sigma_d)\boldsymbol{\alpha}(\hat{\boldsymbol{x}}_t)^T \,, \\ \boldsymbol{\alpha}(\hat{\boldsymbol{x}}_t) &:= k_{\hat{x}_t,\zeta_d} K_{\zeta_d,\zeta_d}^{-1} \,. \end{aligned} \tag{85}$$

8.2.2 Variational Approximation

In previous work [91], a factorized variational distribution is considered based on a mean-field approximation for the latent states $\boldsymbol{x}_{1:T}$. Their variational distribution is given by

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

$$q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z}) = \left[\prod_{d=1}^{D_x} q(\boldsymbol{z}_d) \left[\prod_{t=2}^{T} p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d)\right]\right] \left[\prod_{t=1}^{T} q(\boldsymbol{x}_t)\right] .$$

This choice, however, leads to several caveats: (i) The number of model parameters grows linearly with the length of the time series since each latent state is parametrized by its individual distribution $q(\boldsymbol{x}_t)$ for every time step. (ii) Initializing the latent state is non-trivial since the true, underlying observation mapping is generally unknown and non-bijective. (iii) The model design does not represent correlations between time steps. Instead, these correlations are only introduced by enforcing pairwise couplings during the optimization process. The first two problems have been addressed in [91, 44] by introducing a *recognition* model, e.g. a Bi-RNN[1], which acts as a smoother which can be learned through backpropagation and which allows to obtain the latent states given the input/output sequence.

[44] Eleftheriadis, Nicholson, Deisenroth, and Hensman, "Identification of Gaussian Process State Space Models," 2017

[1] A bi-directional RNN operates on a sequence from left to right and vice versa to obtain predictions based on past and future inputs.

The issue of representing correlations between time steps, however, is currently an open problem which we aim to address with our proposed model structure. Properly representing these correlations is a crucial step in making the optimization problem tractable in order to learn GP-SSMs for complex systems.

For PR-SSM, the variational distribution is given by

$$q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z}) = \prod_{t=2}^{T} \left[p(\boldsymbol{x}_t \mid \boldsymbol{f}_t) \prod_{d=1}^{D_x} [p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d) q(\boldsymbol{z}_d)] \right] q(\boldsymbol{x}_1) \,, \tag{86}$$

with

$$q(\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1 \mid \boldsymbol{\mu}_{x_1}, \boldsymbol{\Sigma}_{x_1}) \,, q(\boldsymbol{z}_d) = \mathcal{N}(\boldsymbol{z}_d \mid \boldsymbol{\mu}_d, \boldsymbol{\Sigma}_d) \,.$$

In contrast to previous work, the proposed variational distribution does not factorize over the latent state but takes into account the true transition model, based on the sparse GP approximation from (81). In previous work, stronger approximations have been required to achieve an analytically tractable ELBO. This work, however, deals with the more complex distribution by combining sampling and gradient-based methods.

In [49], the variational distribution over inducing outputs has been optimally eliminated. This leads to a smoothing problem in a second system requiring computationally expensive, e.g. sample-based, smoothing methods. Instead, we approximate the distribution by a Gaussian, which is the optimal solution in case of sparse GP regression (cf. [152]).

[49] Frigola, Chen, and Rasmussen, "Variational Gaussian process state-space models," 2014

[152] Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," 2009

The PR-SSM model parameters include the variational parameters for the initial state and inducing outputs and hyperparameters, such as inducing inputs, noise parameters and GP kernel parameters:

$$\theta_{\text{PR-SSM}} = (\boldsymbol{\mu}_{x_1}, \boldsymbol{\Sigma}_{x_1}, \boldsymbol{\mu}_{1:D_x}, \boldsymbol{\Sigma}_{1:D_x}, \boldsymbol{\zeta}_{1:D_x}, \sigma^2_{\text{x},1:D_x}, \sigma^2_{\text{y},1:D_y}, \theta_{\text{GP},1:D_x}) \,. \quad (87)$$

Note that in the PR-SSM, the number of parameters grows only with the number of latent dimensions, but not with the length of the time series.

8.2.3 Variational Evidence Lower Bound

Following standard variational inference techniques [19], the ELBO is given by

[19] Blei, Kucukelbir, and McAuliffe, "Variational inference: A review for statisticians," 2017

$$\begin{aligned} \log p(\boldsymbol{y}_{1:T}) &\geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\log \frac{p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})}{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \right] \\ &=: \mathcal{L}_{\text{PR-SSM}} \,. \end{aligned} \quad (88)$$

Maximizing the ELBO is equivalent [19] to minimizing

$$\mathrm{KL}(q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z}) \parallel p(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z} \mid \boldsymbol{y}_{1:T})) \,. \quad (89)$$

Therefore, this is a way to optimize the approximated model parameter distribution with respect to the intractable, true model parameter posterior.

Based on (81) and (86) and using standard variational calculus, the ELBO (88) can be transformed into

$$\begin{aligned} \mathcal{L}_{\text{PR-SSM}} = &\sum_{t=1}^{T} \mathbb{E}_{q(\boldsymbol{x}_t)} [\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)] \\ &- \sum_{d=1}^{D_x} \mathrm{KL}(q(\boldsymbol{z}_d) \parallel p(\boldsymbol{z}_d; \boldsymbol{\zeta}_d)) \,, \end{aligned} \quad (90)$$

with $q(\boldsymbol{x}_t)$ defined in Sec. 8.2.4. The first part is the expected log-likelihood of the observed system outputs $\boldsymbol{y}$ based on the observation model and the variational latent state distribution $q(\boldsymbol{x}_t)$. This term captures the capability of the learned latent state model to explain the observed system behavior. The second term is a regularizer on the inducing output distribution that penalizes deviations from the GP prior. Due to this term, PR-SSM automatically trades off data fit against model complexity. A detailed derivation of the ELBO can be found in the supplementary material.

8.2.4 Stochastic Gradient ELBO Optimization

Training the proposed PR-SSM requires maximizing the ELBO in (90) with respect to the model parameters $\theta_{\text{PR-SSM}}$. While the second term, as KL between two Gaussian distributions, can be easily computed, the first term requires evaluation of an expectation with respect to the latent state distribution $q(\boldsymbol{x}_t)$, given by

$$q(\boldsymbol{x}_t) = \int \prod_{\tau=2}^{t} [p(\boldsymbol{x}_\tau \mid \boldsymbol{f}_\tau) p(\boldsymbol{f}_\tau \mid \hat{\boldsymbol{x}}_{\tau-1}, \boldsymbol{z})] \\ q(\boldsymbol{x}_1) q(\boldsymbol{z}) d\boldsymbol{x}_{1:t-1} d\boldsymbol{f}_{2:t} d\boldsymbol{z} \,. \tag{91}$$

Since the true non-linear, latent dynamics is maintained in the variational approximation (86), analytic evaluation of (91) is still intractable. Because of the latent state's Markovian structure, the marginal latent state distribution $q(\boldsymbol{x}_t)$ at time t is *conditionally* independent of past time steps, given the previous state distribution $q(\boldsymbol{x}_{t-1})$ and the explicit representation of GP inducing points. This enables a differentiable, sampling-based estimation of the expectation term. Samples $\tilde{\boldsymbol{x}}_t$ from (91) can be obtained by recursively drawing from the sparse GP posterior in (85) for $t = 1, \ldots, T$. Drawing samples from a Gaussian distribution can be made differentiable with respect to its parameters μ_d, σ_d^2 using the *re-parametrisation trick* [74]. The gradient can be propagated back through time due to this re-paramatrization and unrolling of the latent state. An unbiased estimator of the first term in the ELBO in (90) is given by

[74] Kingma and Welling, "Auto-encoding variational bayes," 2013

$$\mathbb{E}_{q(\boldsymbol{x}_t)}[\log(\boldsymbol{y}_t \mid \boldsymbol{x}_t)] \approx \frac{1}{N} \sum_{i=1}^{N} \log p(\boldsymbol{y}_t \mid \tilde{\boldsymbol{x}}_t^{(i)}) \,. \tag{92}$$

Based on the stochastic ELBO evaluation, analytic gradients of (90) can be derived to facilitate stochastic gradient-descent-based model optimization.

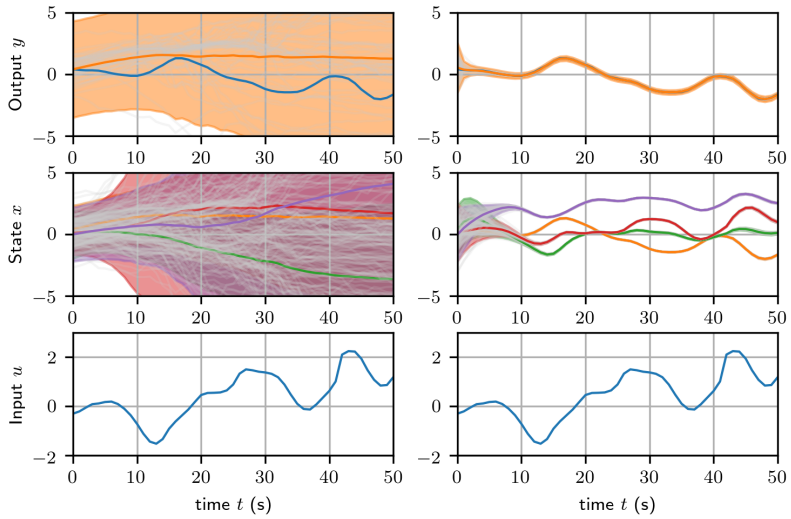


Figure 26: Predictions of the initial, untrained (left) and the final, trained PR-SSM (right) based on the full gradient ELBO optimization. The system input/output data (——) is visualized together with the model prediction (——) for a part of the *Furnace* dataset. Samples of the latent state distribution and output distribution are shown in gray. The colored, shaded areas (——) visualize mean +/- two std of the latent state x and observation y. The initial model exhibits a random walk behavior in the latent space. In the trained model, the decay of the initial state uncertainty can be observed in the first time steps. In this experiment, no recognition model has been used (cf. Sec. 8.3).

8.2.5 Model Predictions

After model optimization based on the ELBO (90), model predictions for a new input sequence $\boldsymbol{u}_{1:T}$ and initial latent state $\boldsymbol{x}_1$ can be obtained based on the approximate, variational posterior distribution in (86). In contrast to [91], no approximations such as moment matching are required for model predictions. Instead, the complex latent state distribution is approximated based on samples from (91). The predicted observation distribution can then be computed from the latent distribution according to the observation model in (82). Instead of a fixed, uninformative initial latent state, a learned recognition model (cf. Sec. 8.3 for details) can be utilized to find a more informative model initialization.

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

8.3 Extensions for Large Datasets

Optimizing the ELBO (90) based on the full gradient is prohibitive for large datasets and long trajectories. Instead, a stochastic optimization scheme based on mini-batches of sub-trajectories is introduced. Directly optimizing the initial latent state distribution $q(\boldsymbol{x}_1)$ for each sub-trajectory would lead to a full parametrization of the latent state which is undesirable as described in Sec. 8.2.2. Instead, we propose a parametric recognition model, which initializes the latent state $q(\boldsymbol{x}_1)$. In recent work on SSMs [91, 44], a recognition model is introduced

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[44] Eleftheriadis, Nicholson, Deisenroth, and Hensman, "Identification of Gaussian Process State Space Models," 2017

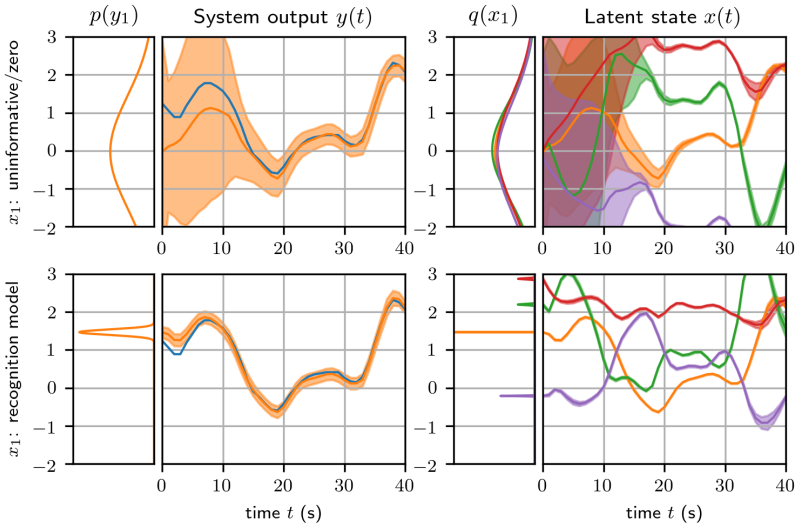


Figure 27: Comparison of the fully trained PR-SSM predictions with (lower row) and without (upper row) initial state recognition model on a test dataset. The initial transient based on the uncertainty from an uninformative initial state distribution $q(\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1 \mid \boldsymbol{0}, \boldsymbol{I})$ decays, as shown in upper plots. Below the outcome is shown when $q(\boldsymbol{x}_1)$ is initialized by the smoothing distribution $q(\boldsymbol{x}_1 \mid \boldsymbol{y}_{1:L}, \boldsymbol{u}_{1:L})$, given the first L steps of system input/output. Using the recognition model yields a significantly improved latent state initialization and therefore decreases the initial state uncertainty and the initial transient behavior.

to parametrize the smoothing distribution $p(\boldsymbol{x}_{1:T} \mid \boldsymbol{y}_{1:T}, \boldsymbol{u}_{1:T})$. We propose a similar approach, but only to model the initial latent state

$$q(\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1 \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \approx q(\boldsymbol{x}_1 \mid \boldsymbol{y}_{1:L}, \boldsymbol{u}_{1:L}), \tag{93}$$

$$\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1 = h(\boldsymbol{y}_{1:L}, \boldsymbol{u}_{1:L}; \theta_{\text{recog}}). \tag{94}$$

The initial latent state distribution is approximated by a Gaussian, where mean and variance are modeled by a recognition model h. The recognition model acts as a smoother, operating on the first L elements of the system input/output data to infer the first latent state. Instead of directly optimizing $q(\boldsymbol{x}_1)$ during training, errors are propagated back into the recognition model h, which is parametrized by θ_{recog}.

Additionally, the proposed recognition model can also be used to predict behavior on test data, where the initial latent state is not known.

8.4 Experimental Evaluation

In the following, we present insights into the PR-SSM optimization schemes, comparisons to state-of-the-art model learning methods and a large scale experiment.

8.4.1 PR-SSM Learning

For small datasets (i.e. short training trajectory lengths), the model can be trained based on the full gradient of the ELBO in (90). A comparison of the model predictions before and after training with the full ELBO gradient is shown in Fig. 26.

	One-step-ahead, autoregressive		Multi-step-ahead, latent space autoregressive			Markovian state-space models	
Task	GP-NARX	NIGP	REVARB 1	REVARB 2	MSGP	SS-GP-SSM	**PR-SSM**
Actuator	0.627 (0.005)	0.599 (0)	0.438 (0.049)	0.613 (0.190)	0.771 (0.098)	0.696 (0.034)	0.502 (0.031)
Ballbeam	0.284 (0.222)	0.087 (0)	0.139 (0.007)	0.209 (0.012)	0.124 (0.034)	411.6 (273.0)	0.073 (0.007)
Drives	0.701 (0.015)	0.373 (0)	0.828 (0.025)	0.868 (0.113)	0.451 (0.021)	0.718 (0.009)	0.492 (0.038)
Furnace	1.201 (0.000)	1.205 (0)	1.195 (0.002)	1.188 (0.001)	1.277 (0.127)	1.318 (0.027)	1.249 (0.029)
Dryer	0.310 (0.044)	0.268 (0)	0.851 (0.011)	0.355 (0.027)	0.146 (0.004)	0.152 (0.006)	0.140 (0.018)
Sarcos	0.169 (-)	n.a.	n.a.	n.a.	n.a.	n.a.	0.049 (-)

Table 2: Model learning benchmark on five real-world datasets. The RMSE result (mean (std. deviation)) from 5 independent runs is given for the free simulation on the test dataset. The best model (——) and the second best model (- - -) are indicated for each dataset. The background color indicates best ( ) to worst ( ) model performance. Best viewed in color. The proposed PR-SSM consistently outperforms the reference (SS-GP-SSM) in the class of Markovian state space models and robustly achieves performance comparable to the one of state-of-the-art latent, autoregressive models. Best viewed in color.

Empirically, three major shortcomings of the full gradient-based optimization schemes are observed:

(i) Computing the full gradient for long trajectories is expensive and prone to the well-known problems of exploding and vanishing gradients [114].

(ii) An uninformative initial state is prohibitive for unstable systems or systems with slowly decaying initial state transients.

(iii) Momentum-based optimizers (e.g. Adam) exhibit fragile optimization performance and are prone to overfitting.

The proposed method addresses these problems by employing the stochastic ELBO gradient based on minibatches of sub-trajectories and the initial state recognition model (cf. Sec. 8.3). Fig. 27 visualizes the initial state distribution $q(\boldsymbol{x}_1)$ and the corresponding predictive output distribution $p(\boldsymbol{y}_1)$ for the fully trained model based on the full gradient (top row), as well as for the model based on the stochastic gradient and recognition model (bottom row). The transient dynamics and the associated model uncertainty is clearly visible for the first 15 time steps until the initial transient decays and approaches the true system behavior. In contrast, the learned recognition model almost perfectly initializes the latent state, leading to much smaller deviations in the predicted observations and far less predictive uncertainty. Notice how the recognition model is most certain about the distribution of the first latent state dimension (orange), which is directly coupled to the observation through the parametric observation model (cf. (82)). The uncertainty for the remaining, latent states, in contrast, is slightly higher.

Comparing the full ELBO gradient-based model learning and the stochastic version with the recognition model, the stochastic model learning is far more robust and counteracts the overfitting tendencies

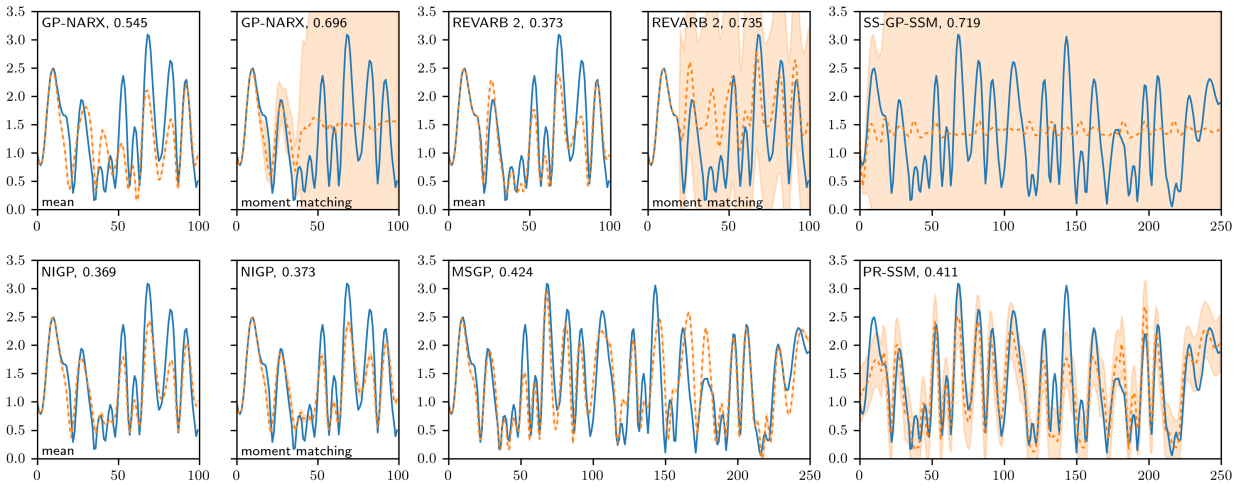


Figure 28: Free simulation results for the benchmark methods on the *Drives* test dataset. The true, observed system output (——) is compared to the individual model's predictive output distribution (mean +/- two std., ——). Results are presented for the one-step-ahead models GP-NARX and NIGP in the left column. REVARB and MSGP (shown in the middle column) are both based on multi-step optimized autoregressive GP models in latent space. In the right column, the SS-GP-SSMs, as a model based on a Markovian latent state, is compared to the proposed PR-SSM.

in the full gradient-based model learning. A comparison of the model learning progress for both methods is depicted in the supplementary material. Due to the improved optimization robustness and the applicability to larger datasets, the stochastic, recognition-model-based optimization scheme is employed for the model learning benchmark presented in the next section. Empirically, the cost of the proposed sampling scheme is much lower compared to methods employing SMC for sampling the full model posterior. In the experiments, 50 latent state samples were employed (details in the supplementary material).

8.4.2 Model Learning Benchmark

The performance of PR-SSM is assessed in comparison to state-of-the-art model learning methods on several real-world datasets as previously utilized by [91]. The suite of reference methods is composed of: One-step ahead autoregressive GP models: GP-FITC [142] and NIGP [94]. Multi-step-ahead autoregressive and recurrent GP models in latent space: REVARB based on 1, respectively 2, hidden layers [91] and MSGP [39]. GP-SSMs, based on a full Markovian state: SS-GP-SSM [149] and the proposed PR-SSM. Currently, no published and runnable code exists for the model learning frameworks presented in [155, 50, 49, 44].

To enable a fair comparison of the methods' performance and robustness, whitened data, a default configuration across tasks and a predefined amount of input/output data for initialization is employed. The presented results are therefore not directly comparable to previous work, where different data pre/postprocessing or method configurations are employed. A more thorough evaluation, which matches the

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[142] Snelson and Ghahramani, "Sparse Gaussian processes using pseudo-inputs," 2006

[94] McHutchon and Rasmussen, "Gaussian process training with input noise," 2011

[39] Doerr, Daniel, Nguyen-Tuong, Marco, Schaal, Toussaint, and Trimpe, "Optimizing Long-term Predictions for Model-based Policy Search," 2017

[149] Svensson and Schön, "A flexible state–space model for learning nonlinear dynamical systems," 2017

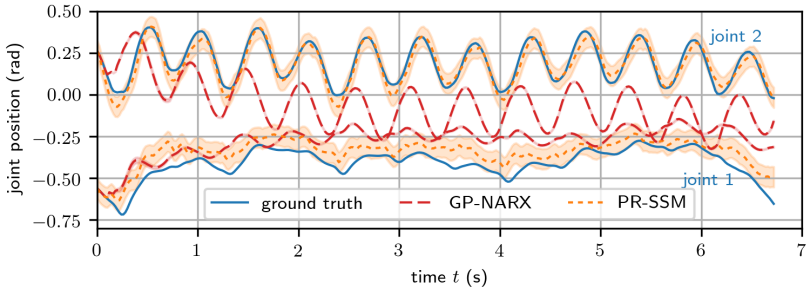


Figure 29: Results on the Sarcos large scale task: Predictions from the GP-NARX baseline (- - -) and the PR-SSM ( ) for two out of seven joint positions are shown together with the ground truth, measured joint positions (——). PR-SSM clearly improves over the GP-NARX predictions. Similar results are obtained for PR-SSM on the remaining 5 joints, where the GP-NARX model fails completely (cf. supplementary materials for details).

published results from previous work, as well as experimental details are given in the supplementary material.

The benchmark results are summarized in Tab. 2. A detailed visualization of the resulting model predictions on the *Drives* dataset is shown in Fig. 28. For the one-step-ahead models (GP-NARX, NIGP), two variants are used to obtain long-term predictive distributions: Propagating the mean (no uncertainty propagation) and approximating the true posterior by a Gaussian using exact moment matching [58]. The results show that PR-SSM consistently outperforms the SS-GP-SSM learning method. Similarly, performance is improved in comparison to baseline methods (GP-NARX and NIGP). In the ensemble of models based on long-term optimized autoregressive structure (REVARB, MSGP), no method is clearly superior. However, the performance of PR-SSM is consistently strong. The probabilistic methods results are competitive or improve over the performance of deterministic RNN/LSTM models, as shown in [91]. Note that PR-SSM demonstrates robust model learning performance across all datasets.

[58] Girard, Rasmussen, Quinonero-Candela, Murray-Smith, Winther, and Larsen, "Multiple-step ahead prediction for non linear dynamic systems—a Gaussian process treatment with propagation of the uncertainty," 2003

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

8.4.3 Large Scale Experiment

To evaluate the scalability, results are provided for the forward dynamics model of the SARCOS 7 degree of freedom robotic arm. The task is characterized by 60 experiments of length 337 ($\approx$ 20.000 datapoints), 7 input, and 7 output dimensions. PR-SSM is set up with a latent state dimensionality $D_x = 14$. From the set of reference methods, only GP-NARX can be adapted to run efficiently on this dataset without major effort (details are given in the supplementary material). A visualization of the model predictions is shown in Fig 29 and prediction RMSEs are listed in Tab. 2. The results show that PR-SSM is able to learn robustly and accurately for all system outputs from all experimental data. In contrast, the GP-NARX baseline achieves worse predictions and fails to predict the remaining five joints (not shown).

8.5 Summary

In this work, we presented Probabilistic Recurrent State-Space Models (PR-SSM) as a novel model structure and efficient inference scheme for learning probabilistic, Markovian state-space models. Based on GP priors and doubly stochastic variational inference, a novel model optimization criterion is derived, which is closely related to the one of powerful, but deterministic, RNNs or LSTMs. By maintaining the true latent state distribution and thereby enabling long-term gradients, efficient inference in latent space becomes feasible. Furthermore, a novel recognition model enables learning of unstable or slow dynamics as well as scalability to large datasets. Robustness, scalability and high performance in model learning is demonstrated on real-world datasets in comparison to state-of-the-art methods.

A limitation of PR-SSM is its dependency on an a-priori fixed latent state dimensionality. This shortcoming could potentially be resolved by a sparsity enforcing latent state prior, which would suppress unnecessary latent state dimensions.

Part III

Model Assumptions in Model-Free Reinforcement Learning

9 Introduction

Policy search methods are amongst the few successful RL [148] methods which are applicable to high-dimensional or continuous control problems, such as the ones typically encountered in robotics [118, 34]. The first two parts of this work focus on model-based policy search methods and efficient modeling for real-world systems, respectively. These methods promise great data-efficiency, principled treatment of uncertainty and, therefore, fast and efficient policy improvement. For one, this is due to the utilization of all available data from possibly different policies and tasks, which can be integrated into a single model of the actual system's dynamics. Other than that, the efficiency of model-based approaches is mostly influenced by prior model assumptions' strength. As an example, modeling a system's dynamics with GPs and the most commonly used squared exponential kernel, which is infinitely differentiable, implies a process with mean square derivatives of all orders. Thus, significant assumptions about the modeled dynamics' smoothness are already made by the employed model structure.

[148] Sutton, McAllester, Singh, and Mansour, "Policy gradient methods for reinforcement learning with function approximation," 2000

[118] Peters and Schaal, "Reinforcement learning of motor skills with policy gradients," 2008

[34] Deisenroth, Neumann, and Peters, "A survey on policy search for robotics," 2013

Ultimately, the question arises, which model assumptions are valid for a given modeling problem. Many real-world problems exist, where strong smoothness assumptions are typically invalid. For example, robotic applications with contact interactions typically experience discontinuous effects once contact is established or released [23]. Similarly, problems with discrete decisions, such as typically found in scheduling or game-like applications, can not be well learned with smooth dynamics models. Though more flexible models might be able to capture these effects in the limit, data-efficiency will be severely restricted.

[23] Chebotar, Hausman, Zhang, Sukhatme, Schaal, and Levine, "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning," 2017

As discussed in Sec. 2.2, other classes of RL and policy search methods exist. As in the (dynamics) model-based regime, these RL classes as well incorporate the available interaction data to solve a specific modeling problem. For Bayesian Optimization (BO) or Policy Gradient (PG) methods, this is an expected return model. Value- or Q-Functions methods model the expected value as a function of state and possibly action. Consequently, similar questions about model assumptions and thus the capability to generalize from all available data arise for these modeling problems.

The following part shifts the focus to the class of Policy Gradient (PG) methods. Instead of extending and refining the model assumptions in model-based RL methods to make them more applicable to

real-world applications, this part takes the opposite direction. A novel methodology called Deep-Deterministic Off-Policy Gradients (DD-OPG) is derived starting from a model-free stance of vanilla policy gradient methods. DD-OPG carefully reintroduces model-assumptions to foster data-efficiency and robustness in learning.

The main portion of work presented in this part of the thesis has been previously published as

> A. Doerr, M. Volpp, M. Toussaint, S. Trimpe, and C. Daniel. "Trajectory-based off-policy deep reinforcement learning." In: *International Conference on Machine Learning (ICML)*. 2019, pp. 1636–1645

9.1 Outline

This part initially discusses model-free policy gradient methods in Sec. 9.2. Advantages and drawbacks are contrasted with the novel contributions in the DD-OPG method. A discussion on related work from different view angles is given in Sec. 9.3.

The novel DD-OPG method's primary derivation can be found in Chap. 10. This chapter starts by introducing the general problem formulation and policy gradient framework in Sec. 10.1. A short presentation of the standard importance sampling estimators to incorporate off-policy data is given in Sec. 10.2. The surrogate model, necessary to efficiently incorporate deterministic policy data, as the core of the proposed model-free DD-OPG method is detailed in Sec. 10.3. In Sec. 10.4, the main policy optimization scheme is presented.

The experimental evaluation of the DD-OPG method is presented in 11. Individual studies are conducted to showcase the novel surrogate model (Sec. 11.1), a performance benchmark (Sec. 11.2), and an ablation study for different parts of the DD-OPG methods (Sec. 11.3). This work concludes with a summary and outlook into future work in Sec. 11.3.

9.2 Discussion on Policy Gradient Methods

Policy Gradient (PG) algorithms have achieved impressive results on highly complex tasks [133, 134]. However, standard algorithms are vastly data-inefficient and rely on millions of data points to achieve the aforementioned results. Typical applications are therefore limited to simulated problems where policy rollouts can be cheaply obtained.

Algorithms based on stochastic policy gradients, like REINFORCE [169] and G(PO)MDP [12], typically estimate the policy gradient based on a batch of trajectories, which are obtained by executing the current

[133] Schulman, Levine, Abbeel, Jordan, and Moritz, "Trust region policy optimization," 2015

[134] Schulman, Wolski, Dhariwal, Radford, and Klimov, "Proximal policy optimization algorithms," 2017

[169] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992

[12] Baxter and Bartlett, "Infinite-horizon policy-gradient estimation," 2001

policy on the system (i.e., based on on-policy samples). All previous experience is discarded in the next step, and new trajectories are sampled using the updated policy. This scheme also holds for more recent methods, like PPO [134] or POIS [95], where a surrogate objective is constructed, which can be optimized till convergence. Typically, Importance Sampling (IS) techniques are employed to evaluate a target policy based on rollouts obtained from behavioral policies (i.e., from off-policy samples). Albeit these off-policy evaluation schemes, in these algorithms, no data is shared between iterations. Prominent examples of off-policy offline algorithms typically employ actor-critic architectures [139]. The parametric critic model, typically a value function, is updated to summarize all knowledge gathered so far. In contrast, we proposed the model-free Deep Deterministic Off-Policy Gradient method (DD-OPG)[1], which incorporates previously gathered rollout data by sampling from a trajectory replay buffer. The replay buffer of all available data effectively enables backtracking to promising solutions. DD-OPG only requires minimal assumptions to construct the surrogate model.

[134] Schulman, Wolski, Dhariwal, Radford, and Klimov, "Proximal policy optimization algorithms," 2017

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

[139] Silver, Lever, Heess, Degris, Wierstra, and Riedmiller, "Deterministic policy gradient algorithms," 2014

[1] Code available at https://github.com/boschresearch/DD_OPG

Next to the inefficient use of available data, stochasticity in both the policy and the environment causes highly variable gradient estimates and, therefore, slow convergence. When executing the probabilistic policy on the system, noise is injected into the policy gradient in each time step, leading to a variance, which linearly increases with the length of the horizon [101]. Additive Gaussian noise is typically employed as a source of exploration. Additionally, PG methods built around the likelihood ratio trick intrinsically require probabilistic policies. Only then can policies be updated to increase the likelihood of actions, which have been advantageous in previous rollouts. Instead of independent noise, temporally-correlated noise [112], or exploration directly in parameter space can lead to a wider variety of behaviors [121]. Here, the behavioral policy is deterministic, thereby effectively reducing the gradient variance. Methods like DPG [139] and DDPG [86] learn a parametric value function model to translate changes in policy and, therefore, actions to changes in expected value. Similarly, our proposed model-free DD-OPG algorithm constructs a non-parametric critic based on importance sampling. This critic, called the surrogate model in the following, allows for updating a deterministic policy without the need for explicit parametric value models.

[101] Munos, "Policy gradient in continuous time," 2006

[112] Osband, Blundell, Pritzel, and Van Roy, "Deep exploration via bootstrapped DQN," 2016

[121] Plappert, Houthooft, Dhariwal, Sidor, Chen, Chen, Asfour, Abbeel, and Andrychowicz, "Parameter space noise for exploration," 2017

[86] Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, and Wierstra, "Continuous control with deep reinforcement learning," 2015

To summarize: We propose an importance sampling-based surrogate model of the return distribution, which enables off-policy, offline policy optimization. This surrogate facilitates deterministic policy gradients to reduce gradient variance and enables the incorporation of all available data from a replay buffer. Exploration in the policy parameter space is achieved by a prioritized resampling of the surrogate's support data, thus favoring promising regions in policy space.

Normalized IS, which we demonstrate to act similarly as a baseline in standard PG methods, additionally reduces the variance of the employed estimates. Although no additional parametric value function baseline (as utilized in TRPO/PPO for variance reduction) is required in our method, fast progress and, therefore, data-efficient learning is demonstrated on typical continuous control tasks.

9.3 Related Work

Policy search methods [118, 34] and policy gradient methods [169, 12]are well studied in the RL community and many connections to DD-OPG exist. In the following, we review related work from several perspectives.

Importance Sampling Perspective

Importance sampling has been employed to either reweight full trajectory distributions [136, 67, 173, 95] or to reweight individual state-action pairs [102, 45] . Except for [67], no global IS estimator is derived, but estimates are only based on the current iteration's data. In contrast, DD-OPG introduces a global surrogate model based on all available deterministic policy rollouts and computes local, stochastic approximations using prioritized replay. Instead of DD-OPG's action space lengthscale, alternative approaches consider truncation of the importance weights [166, 134, 45]. So far, the connection between both approaches has not yet been the subject of more in-depth analysis.

Variance Reduction Perspective

One primary concern when estimating the policy gradient from Monte Carlo samples of trajectory returns is the variance, or conversely, the required number of trajectory samples. Several techniques have been introduced to reduce the variance of the standard REINFORCE Monte Carlo estimator. Instead of full trajectory returns, each action can be judged based on the experienced reward to go, i.e., the accumulated reward as started from the specific action. Most commonly, value function estimates are deployed as baselines or control variates to reduce the estimator variance further. In DD-OPG, the normalization of the importance weighted estimator acts as a data-based baseline. This results in variance reduction at the cost of introducing bias.

[118] Peters and Schaal, "Reinforcement learning of motor skills with policy gradients," 2008

[34] Deisenroth, Neumann, and Peters, "A survey on policy search for robotics," 2013

[169] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992

[12] Baxter and Bartlett, "Infinite-horizon policy-gradient estimation," 2001

[136] Shelton, "Policy improvement for POMDPs using normalized importance sampling," 2001

[67] Jie and Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," 2010

[173] Zhao, Hachiya, Tangkaratt, Morimoto, and Sugiyama, "Efficient sample reuse in policy gradients with parameter-based exploration," 2013

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

[102] Munos, Stepleton, Harutyunyan, and Bellemare, "Safe and efficient off-policy reinforcement learning," 2016

[45] Espeholt, Soyer, Munos, Simonyan, Mnih, Ward, Doron, Firoiu, Harley, and Dunning, "IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures," 2018

[166] Wawrzynski and Pacut, "Truncated importance sampling for reinforcement learning with experience replay," 2007

[134] Schulman, Wolski, Dhariwal, Radford, and Klimov, "Proximal policy optimization algorithms," 2017

Objective Function Perspective

In general, the goal of an RL agent is to maximize its expected return J by optimizing the parameters θ of its behavioral policy. Given some local estimator for the policy gradient $dJ/d\theta$, we can deploy standard gradient ascent to achieve this objective [169]. Taking gradient steps typically moves farther away from available data and thus naturally increases the variance or bias of the gradient estimators. More advanced methods therefore incorporate trust regions [133] or lower bounds, which can be fully optimized till convergence [134, 95]. Effectively, this results in a trade-off between the size of the policy update and trust in the estimate. Secondly, when designing a policy gradient algorithm based on this surrogate model, we can shape the objective function to influence the exploration-exploitation trade-off. For example, optimizing upper confidence bound fosters exploration in promising but currently uncertain regions. The proposed DD-OPG optimizes a stochastic version based on the lower bound, derived in [95].

[169] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992

[133] Schulman, Levine, Abbeel, Jordan, and Moritz, "Trust region policy optimization," 2015

[134] Schulman, Wolski, Dhariwal, Radford, and Klimov, "Proximal policy optimization algorithms," 2017

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

Exploration Perspective

Deterministic policies as means of variance reduction have been previously discussed for example in [135, 121]. Instead of action noise for exploration, exploration is achieved by stochasticity in parameter space. The DD-OPG method relies on deterministic policies for variance reduction but introduces exploration employing stochastic gradients from the prioritized replay model.

[135] Sehnke, Osendorfer, Rückstieß, Graves, Peters, and Schmidhuber, "Policy gradients with parameter-based exploration for control," 2008

[121] Plappert, Houthooft, Dhariwal, Sidor, Chen, Chen, Asfour, Abbeel, and Andrychowicz, "Parameter space noise for exploration," 2017

Bayesian Optimization Perspective

Bayesian optimization methods are typically agnostic to the step-by-step agent-environment interactions. Instead, they model the expected return as a function of the policy parameters, given return estimates from rollouts with known policies. Typically in Bayesian optimization, a Gaussian Process prior is assumed to facilitate a Bayesian inference scheme. However, many typically employed kernel functions (e.g., the squared exponential kernel) can not appropriately account for abrupt changes and non-stationary behavior in the expected return, as is generally observed when moving from stable to unstable regions in parameter space.

In [170], the authors propose a kernel that expresses covariance in parameter space in terms of likelihood ratios of trajectory actions conditions on observed state trajectories. The resulting warping of the parameter space is very comparable to our proposed surrogate model.

[170] Wilson, Fern, and Tadepalli, "Using trajectory data to improve Bayesian optimization for reinforcement learning," 2014

The likelihood ratios in the importance sampling weights determine the influence of neighboring data-points on the estimate.

10

Deep-Deterministic Off-Policy Gradients

In this chapter, the novel Deep-Deterministic Off-Policy Gradient (DD-OPG) algorithm is developed, starting from the baseline Policy Gradient (PG) estimator as known from the REINFORCE [169] algorithm. First, the PG estimator is derived in Sec. 10.1. An extension to off-policy data based on importance sampling is discussed in Sec. 10.2. The novel DD-OPG contributions, learning from deterministic policies and objectives to optimize the resulting surrogate model, are derived in Sec. 10.3 and Sec. 10.4 respectively.

[169] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992

10.1 Preliminaries

This section depicts the general episodic RL problem in a discrete-time Markovian environment and summarizes the core building block of the proposed DD-OPG method, the standard return-based policy gradient estimators. DD-OPG closely follows this algorithmic structure (cf. Alg. 3), however with extensions to incorporate deterministic, off-policy rollouts as detailed in the following sections. The RL problem is characterized by a discrete-time Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma, p_0)$. An agent is interacting with an environment, whose states $s_t \in \mathcal{S}$ transitions according to the agent's actions $a_t \in \mathcal{A}$ and the environment's transition probabilities $p(s_{t+1} \mid s_t, a_t)$ into a successor state. Starting from a state s_0 drawn from the initial state distribution $p(s_0)$, agent tries to maximize its discounted reward, according to a reward function $r : S \times A \to \mathbb{R}$ and discount factor γ, accumulated over a horizon length H. In policy search, the agent acts according to a (stochastic) policy $\pi_\theta = \pi(a_t \mid s_t; \theta)$, parameterized by θ. The expected accumulated reward is given by

$$J(\theta) = \int p(\tau \,|\, \theta) R(\tau) d\tau \,, \tag{95}$$

where the trajectory $\tau \in \mathcal{T}$ is the sequence of state-action pairs $\tau = (s_0, a_0, \ldots, s_H, a_H)$, the (discounted) trajectory return is given by $R(\tau) = \sum_{t=0}^{H-1} \gamma^t r(s_{\tau,t}, a_{\tau,t})$, and due to the Markov property, the trajectory distribution in (95) is given by

$$p(\tau \,|\, \theta) = p(s_0) \prod_{t=0}^{H} p(s_{t+1} \,|\, s_t, a_t) \pi(a_t \,|\, s_t; \theta) \,. \tag{96}$$

The dynamics of the system $p(s_{t+1} \mid s_t, a_t)$ and the initial state distribution $p(s_0)$ are generally unknown to the learning agent.

Model-free policy gradient methods typically directly estimate the expected cost gradient based on the log-derivative trick. The gradient is given by

$$\nabla_\theta J(\theta) = \int p(\tau\,|\,\theta)\nabla_\theta \log p(\tau\,|\,\theta) R(\tau) d\tau\,. \tag{97}$$

Given on-policy samples $\tau_i \sim p(\tau|\theta)$, the following Monte Carlo (MC) estimators are obtained for the expected return

$$\hat{J}^{MC}(\theta) = \frac{1}{N}\sum_{i=1}^{N} R(\tau_i)\,, \tag{98}$$

and the policy gradient

$$\nabla_\theta \hat{J}^{MC}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi(a_t\,|\,s_t;\theta) R(\tau_i)\right]. \tag{99}$$

Since the unknown initial state and dynamics distributions are independent of the policy parameters θ (cf. (96)), the trajectory likelihood gradient $\nabla_\theta \log p(\tau \mid \theta)$ with respect to the policy parameters can be computed analytically for a given, differentiable policy $\nabla_\theta \log \pi(a_t\,|\,s_t;\theta)$.

10.2 Off-Policy Evaluation

The MC estimators require a substantial amount of on-policy rollouts $\tau_i \sim p(\tau|\theta^*)$ to reduce the gradient estimator's variance and typically many more rollouts than used in state-of-the-art implementations to approximate the true gradient [65] closely.

For off-policy data, Importance Sampling (IS) can be utilized to incorporate trajectories from a behavioural policy $\pi_{\theta'}$ in order to evaluate a new target policy $\pi_{\theta*}$ [173, 45, 102, 95]. In general, a Monte Carlo estimate of an expectation $\int p(x) f(x) dx$ (such as (95)) can be obtained by sampling from a tractable distribution $x_i \sim q(x)$ and re-weighting the sampled function evaluations $f(x_i)$ based on the likelihood-ratio $p(x_i)/q(x_i)$. The expected return can be rewritten as

$$J(\theta) = \int p(\tau \mid \theta') \frac{p(\tau \mid \theta)}{p(\tau \mid \theta')} R(\tau) d\tau\,, \tag{100}$$

[65] Ilyas, Engstrom, Santurkar, Tsipras, Janoos, Rudolph, and Madry, "Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?" 2018

[173] Zhao, Hachiya, Tangkaratt, Morimoto, and Sugiyama, "Efficient sample reuse in policy gradients with parameter-based exploration," 2013

[45] Espeholt, Soyer, Munos, Simonyan, Mnih, Ward, Doron, Firoiu, Harley, and Dunning, "IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures," 2018

[102] Munos, Stepleton, Harutyunyan, and Bellemare, "Safe and efficient off-policy reinforcement learning," 2016

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

such that the IS weighted Monte Carlo estimator is given by

$$\hat{J}^{IS}(\theta) = \frac{1}{N}\sum_{i=0}^{N}\frac{p(\tau_i \mid \theta)}{p(\tau_i \mid \theta')}R(\tau_i) \tag{101}$$

$$= \frac{1}{N}\sum_{i=0}^{N} w(\tau_i, \theta)R(\tau_i)\,, \tag{102}$$

where N trajectories are sampled from a policy $\pi_{\theta'}$ to infer the expected cost of policy π_θ. Although system dynamics and initial state distribution in (96) are unknown, the likelihood-ratio, i.e. the importance weights, can be computed since the unknown parts cancel out, such that

$$w(\tau, \theta) = \frac{p(\tau \mid \theta)}{p(\tau \mid \theta')} = \frac{\prod_{t=0}^{H}\pi(a_t \mid s_t; \theta)}{\prod_{t=0}^{H}\pi(a_t \mid s_t; \theta')}. \tag{103}$$

During learning, trajectories are collected from multiple different policies $\mathcal{D} = \{(\tau_i, \theta_i)\}_{i=1}^{N}$. To incorporate all data, the importance sampling distribution can be replaced by an empirical mixture distribution $q(\tau \mid \theta_1, \ldots, \theta_N) = 1/N \sum_i p(\tau \mid \theta_i)$ such that the available trajectories are i.i.d. draws from the empirical mixture distribution $\tau_i \sim q(\tau \mid \theta_1, \ldots, \theta_N)$ [67]. The resulting importance weights are given by

[67] Jie and Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," 2010

$$w(\tau, \theta) = \frac{\prod_{t=0}^{H}\pi(a_t \,|\, s_t; \theta)}{\frac{1}{N}\sum_j \prod_{t=0}^{H}\pi(a_t \,|\, s_t; \theta_j)}\,. \tag{104}$$

Computing the importance weights in (104), however, scales quadratically with the number of available trajectories due to the summation over the likelihoods of all trajectories given all available policies. Scaling this estimator to today's deep neural network policies with a large number of required rollouts is, thus, a significant challenge. Computing the surrogate model based on all data, as in [67], is only feasible for several hundred rollouts. Instead, the proposed DD-OPG method employs a trajectory replay buffer and a probabilistic selection scheme to compute a stochastic approximation of the full surrogate model. This idea is related to prioritized experience replay [132] but for full trajectories. It enables scaling to much larger datasets and at the same time helps to avoid local minima by stochastically optimizing the objective.

[132] Schaul, Quan, Antonoglou, and Silver, "Prioritized experience replay," 2015

Another technique typically employed for IS is weight normalization [95]. The *weighted importance sampling* estimator obtains a lower variance estimate at the cost of adding bias. It has been employed in [116] and is both theoretically and empirically better-behaved [96, 122, 136] compared to the pure IS estimator. The weighted importance sampling estimator is given by

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

[116] Peshkin and Shelton, "Learning from Scarce Experience," 2002

[96] Meuleau, Peshkin, Kaelbling, and Kim, "Off-policy policy search," 2000

[122] Precup, Sutton, and Singh, "Eligibility Traces for Off-Policy Policy Evaluation.," 2000

[136] Shelton, "Policy improvement for POMDPs using normalized importance sampling," 2001

$$\hat{J}^{\text{WIS}}(\theta) = \frac{1}{Z}\sum_{i=0}^{N} w(\tau_i, \theta)R(\tau_i)\,, \tag{105}$$

where importance weights $w(\tau_i, \theta)$ might be computed according to (103) or (104) and a normalizing constant $Z = \sum_{i=0}^{N} w(\tau_i, \theta)$ instead of the standard normalization $Z = N$, previously used in (102).

From the policy gradient perspective, by normalizing the importance weights, we obtain a gradient estimator, which includes a parameter dependent baseline.

Proposition 1. *The policy gradient estimator obtained from the self-normalized importance sampling expected cost estimator $\hat{J}^{WIS}$ is given by*

$$\nabla_\theta \hat{J}^{WIS}(\theta) = \frac{1}{Z} \sum_{i=1}^{N} \Bigg[w(\tau_i, \theta) \sum_{t=0}^{H} \Big[\nabla_\theta \log \pi(a_t^{(i)} \,|\, s_t^{(i)}; \theta) \Big] \Big[R(\tau_i) - \hat{J}^{WIS}(\theta) \Big] \Bigg]. \tag{106}$$

A proof of this proposition is shown in appendix D.1. This estimator is closely related to standard PG estimators with an added baseline term for variance reduction.

In standard, REINFORCE like, PG methods, two of the most common variance reduction techniques [60] are: i) incorporation of the reward-to-go for each policy action update instead of the entire Monte Carlo path return; and ii) subtraction of a state-dependent baseline term, such as to obtain an estimate of the advantage of the previously taken action. The intuition behind method i) is to reward actions only for rewards obtained after the action took effect, but not for those obtained earlier on. However, to compute the importance weights not for the full trajectory distribution but for each state-action pair individually, the computation of a matrix of size $\mathcal{O}(N^2H^2)$ would be required. Therefore, the model-free, importance sampling-based approaches are typically limited to the path return based estimators. Model-based methods (i.e., parametric models for the value function) are employed in the cost-to-go estimators. Variance reduction method ii) is automatically obtained by the normalized estimator as shown in proposition 1. However, in contrast to the bias-free value function control variates, at the cost of adding bias. Additionally, optimal baselines to further decrease the variance of the gradient estimator have been derived in [67] and could be incorporated into DD-OPG.

[60] Greensmith, Bartlett, and Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," 2004

[67] Jie and Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," 2010

10.3 Deterministic Policy Gradients

The policy gradient estimators in (99) and (106) rely on a policy distribution $\pi(a_t|s_t;\theta)$ in order to obtain a gradient signal on how to update the policy parameters to increase the likelihood of successful actions. In this situation, the typically Gaussian additive policy noise acts in two ways, causing exploration and serving as the basis for the estimation of the objective function.

Exploration is being driven directly through the noise in the action space, i.e., the policy covariance. While driving exploration through noisy actions will converge in the limit, the resulting explorative behavior exhibits no temporal correlations, which can make it inefficient. *Estimation* of the objective function is typically achieved by reweighting the action distribution according to the policy's likelihood. Standard policies are given as $\pi(a_t|s_t;\theta) = \mathcal{N}(a_t|\mu_\theta(s_t), \Sigma_\theta)$, where μ_θ is represented by some function approximator parameterized by θ, e.g. a neural network. The additive Gaussian noise covariance is typically a diagonal matrix, parameterized by θ as well. The proposed deterministic policy gradient method strives to separate the exploration and estimation part.

Parameter Space Exploration By utilizing deterministic rollout policies, the only noise introduced into the gradient estimate originates from the stochasticity of the environment, and we have to perform exploration in parameter space instead of action space exploration. However, as stated above, parameter-based exploration may, in many cases, be more efficient than exploration in action space since parameter-based exploration will lead to temporally correlated actions that can explore the state space faster. Typically, however, this effect is negated for neural network policies since the parameter space that has to be explored is prohibitively large. Thus, to navigate large parameter spaces efficiently, some approximate evaluation of the cost function (95) is needed.

Trajectory based objective estimate Whilst evaluation of the Monte Carlo based expected cost estimate is possible also for deterministic policies, the off-policy evaluation is no longer feasible since the likelihood ratio $p(\tau|\theta)/p(\tau|\theta')$ (cf. (103)) becomes zero for two distinct dirac policy action distributions if $\mu_\theta(s) \neq \mu_{\theta'}(s)$.

However, we can still compare trajectories under a stochastic evaluation distribution, similar to a kernel function where the standard deviation of the evaluation function relates to a kernel lengthscale in action space.

Thus, we introduce the evaluation policy

$$\tilde{p}(a_t|s_t;\theta) = \mathcal{N}(a_t|\mu_\theta(s_t), \Sigma)\,, \tag{107}$$

where $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_{D_u})$ is a diagonal covariance matrix as typically employed in deep RL methods with Gaussian action noise. The deterministic policy is given by $p(a_t|s_t;\theta) = \delta(a = \mu_\theta(s_t))$, where δ is the dirac delta. From the general IS expectation in (100) and our evaluation policy in (107), the surrogate model follows as

$$\hat{J}^{\mathrm{surr}}(\theta) = \frac{1}{Z}\sum_{i=1}^{N} \tilde{w}(\tau_i, \theta) R(\tau_i)\,, \tag{108}$$

with surrogate weights

$$\tilde{w}(\tau_i, \theta) = \frac{\prod_{t=0}^{H} \mathcal{N}(a_t^{(i)} | \mu_\theta(s_t^{(i)}), \Sigma)}{\frac{1}{N} \sum_{j=0}^{N} \prod_{t=0}^{H} \mathcal{N}(a_t^{(i)} | \mu_{\theta_j}(s_t^{(i)}), \Sigma)}, \tag{109}$$

where, depending on the choice of normalization constant Z, we obtain the analogue to the standard IS estimator ($Z = N$) or the analog to the weighted IS estimator ($Z = \sum_{i=1}^{N} \tilde{w}(\tau_i, \theta)$). Reintroducing the fixed Gaussian noise as an implicit loss to obtain gradients for the evaluation of deterministic policies is clearly a model assumption in the proposed method but can be justified from several perspectives.

The hyper-parameter Σ allows for control over the amount of information shared between neighboring policies. Similar to the cap of importance weights in PPO [134], this parameter allows to control bias and variance of the surrogate model. Analyzing the introduced bias and relation to the PPO weight cap is, however, ongoing research. In the limit of $\Sigma \to \mathbf{0}$, the proposed surrogate (108) approaches the MC estimator (98). Only in case of two different policy parameterizations $\theta_j \neq \theta_i$, but equivalent actions $\mu_{\theta_j}(s) = \mu_{\theta_i}(s)$ for the sampled states s, the surrogate model would output an average whereas the MC estimator would not mix up the obtained returns. For $\Sigma = \Sigma_\theta$, the surrogate model recovers the true IS estimate, given that all trajectories are generated using the same additive Gaussian noise. Finally, for $\Sigma \to \text{inf}$, the estimate is simply the average over all available path returns.

[134] Schulman, Wolski, Dhariwal, Radford, and Klimov, "Proximal policy optimization algorithms," 2017

Modeling the expected return distribution by choosing a lengthscale in action space can furthermore be motivated from a second perspective. Typically expected return distributions oftentimes comprise sharp transitions between stable and unstable regions, where policy parameters change only slightly but reward changes drastically. One global lengthscale is, therefore, typically not well suited to directly model the expected return. This is a common problem in Bayesian Optimization for reinforcement learning, where typical smooth kernel functions (e.g., squared exponential kernel) with globally fixed lengthscales are unable to model both stable and unstable regimes at the same time. However, in the proposed model, a lengthscale in action space is translated via the sampled state distribution and policy function μ into implicit assumptions in the actual policy parameter space. Doing so, instead of operating on arbitrary Euclidean distances in policy parameter space, a more meaningful distance in trajectory and action space is available. Typically, for a given system, the distance of trajectories and between actions is more graspable compared to arbitrary deep neural network policy parameters.

The expected return estimator (108) falls back to zero for policy evaluation far away from training data. To estimate the variance of the importance sampling estimator itself, typically, the Effective Sample

Size (ESS) is evaluated. Based on the variance of the importance weights, it analyses the effective number of available data points at a specific policy evaluation position. In [95], a lower bound on the expected return has been proposed such that with probability $1-\delta$ it holds that

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

$$\mathrm{E}_{\tau\sim p(\tau|\theta)}[R(\tau)] \geq \frac{1}{N}\sum_{i=1}^{N}\tilde{w}(\tau_i,\theta)R(\tau_i) - \|R\|_\infty \sqrt{\frac{(1-\delta)d_2(p(\tau|\theta)\|p(\tau|\theta'))}{\delta N}}, \tag{110}$$

where d_2 is the exponentiated 2-Rényi divergence. Due to the identity $\mathrm{ESS}(P||Q) = N/d_2(P||Q)$, this lower bound can be estimated in a sample-based way by employing the ESS estimator

$$\hat{\mathrm{ESS}} = \frac{1}{\sum_{i=1}^{N}\tilde{w}(\tau_i,\theta)^2}, \tag{111}$$

such as to obtain the lower bound estimate

$$\mathrm{E}_{\tau\sim p(\tau|\theta)}[R(\tau)] \geq \frac{1}{N}\sum_{i=1}^{N}\tilde{w}(\tau_i,\theta)R(\tau_i) \tag{112}$$

$$- \|R\|_\infty \sqrt{\frac{1-\delta}{\delta}\mathrm{ESS}(\theta)^{-1}}. \tag{113}$$

Refer to theorem 4.1 in [95] for details and proof regarding the lower bound in (110). The confidence parameter δ determines, similar to the KL-divergence in TRPO [133], how far the policy optimization can step away from known regions. In DD-OPG, this uncertainty estimate is employed as penalty

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

[133] Schulman, Levine, Abbeel, Jordan, and Moritz, "Trust region policy optimization," 2015

$$\mathrm{penalty}(\theta) = -\|R\|_\infty \gamma \sqrt{\hat{\mathrm{ESS}}(\theta)^{-1}}, \tag{114}$$

with penalty factor γ as an hyper-parameter to control exploration, i.e. following the objective estimate vs. risk awareness, i.e. staying within a trust region.

10.4 Model-Free Off-Policy Optimization

The surrogate model of the return distribution, as derived in Sec. 10.3, can now be directly incorporated for policy optimization. In related work, parametric search distributions (e.g. Gaussian) are employed as policy search distribution or hyperpolicy [173, 121, 95]. However, in high-dimensional spaces, as typically obtained with deep network policy representations, updating the full search distribution is challenging, and common approaches usually revert to heuristics to control a simplified, e.g., diagonal or block-wise search distribution's covariance matrix.

[173] Zhao, Hachiya, Tangkaratt, Morimoto, and Sugiyama, "Efficient sample reuse in policy gradients with parameter-based exploration," 2013

[121] Plappert, Houthooft, Dhariwal, Sidor, Chen, Chen, Asfour, Abbeel, and Andrychowicz, "Parameter space noise for exploration," 2017

[95] Metelli, Papini, Faccio, and Restelli, "Policy Optimization via Importance Sampling," 2018

Instead, the proposed model-free DD-OPG method fully optimizes a stochastic version of the surrogate objective to foster exploration and overcome local minima. At the same time, the stochastic evaluation mitigates the unfavorable complexity of computing the full importance sampling estimate based on all available data. Due to the empirical mixture distribution in (104), computing the likelihood of all observed trajectories under all policies is quadratic in the number of observed paths. Instead, the proposed method employs a selection criterion to construct a stochastic surrogate model based on a subset of rollouts in each policy optimization step. In particular, a predefined number of N_{max} rollout indices is drawn from the softmax distribution over the discrete set of available trajectory indices $\mathcal{I}$. The softmax is computed based on the normalized, empirical returns $\tilde{R}$ and a temperature factor λ.

$$p(\mathcal{I}|\tau_1, \ldots, \tau_N) = \frac{\exp(\tilde{R}(\tau_\mathcal{I})/\lambda)}{\sum_{j=1}^{N} \exp(\tilde{R}(\tau_j)/\lambda)}. \tag{115}$$

The temperature λ is used to trade off exploration against exploitation in the selection of reference trajectories. This scheme is closely related to prioritized experience replay [132]. There, instead of full trajectories, single state-action transitions are sampled from a softmax over the temporal difference error for deep q-network training. A study of the effect of temperature selection on the learning progress is shown in Sec. 11.3.

[132] Schaul, Quan, Antonoglou, and Silver, "Prioritized experience replay," 2015

Algorithm 3: Model-free DD-OPG

Input: Initial policy parameters θ_0
Empty trajectory replay buffer $\mathcal{D}_0 = \{\}$
repeat
  Sample trajectory: $\tau_i \sim p(\tau \,|\, \theta_i)$
  Update trajectory buffer: $\mathcal{D}_{i+1} = \mathcal{D}_i \cup (\tau_i, R_i)$
  Memory selection: $i_1, \ldots, i_{N_{max}} \overset{iid}{\sim} p(\mathcal{I}|\tau_1, \ldots, \tau_i)$
  Surrogate model: $\tilde{J}(\theta)$, penalty(θ)
  Lower bound optimization:
  $\theta_{i+1} = \underset{\theta}{\text{argmax}} \tilde{J}(\theta) - \text{penalty}(\theta)$
until converged or maximum iterations

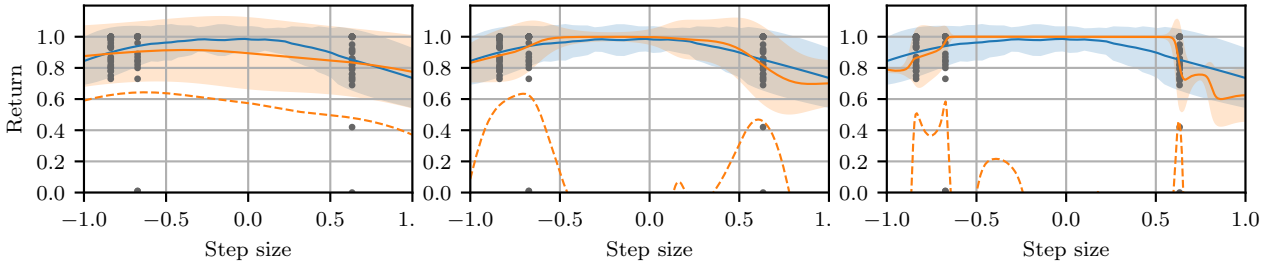
The full DD-OPG algorithm is detailed in Alg. 3. The main objective is to incorporate all available deterministic policy rollouts, not only the ones from the current iteration, into the surrogate model by means of the softmax replay selection. The lower bound expected return can then be fully optimized using standard optimization techniques. In practice, Adam [73] is employed, but other techniques, e.g., based on the natural policy gradient [119] could be incorporated as well.

[73] Kingma and Ba, "Adam: A method for stochastic optimization," 2014

[119] Peters and Schaal, "Natural actor-critic," 2008

11 Experimental Evaluation

The experimental evaluation of the proposed DD-OPG method is threefold. In Sec. 11.1, the resulting surrogate return model is visualized, highlighting different modeling options. A benchmark against state-of-the-art PG methods is shown in Sec. 11.2 to highlight fast and data-efficient learning. Finally, essential parts of the proposed algorithms and their effects on the final learning performance are highlighted in an ablation study in Sec. 11.3.



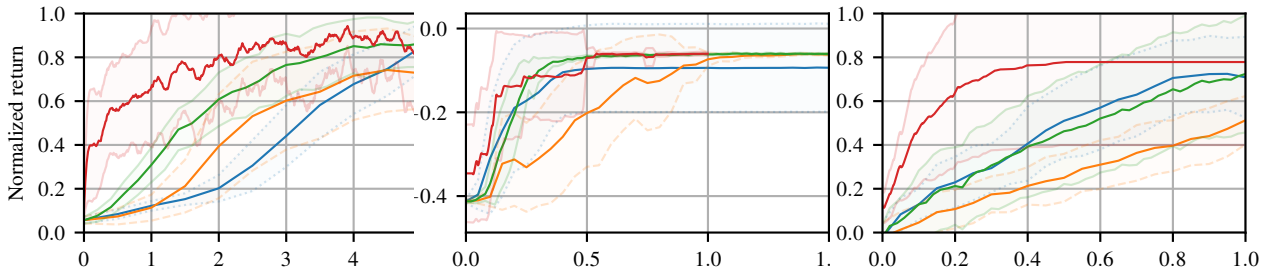
(a) $\log(\Sigma) = 0 \cdot \mathbf{I}$ (b) $\log(\Sigma) = -1 \cdot \mathbf{I}$ (c) $\log(\Sigma) = -2 \cdot \mathbf{I}$

Figure 30: Visualization of the surrogate return model. A cross-section along a random direction in parameter space is shown for parameters close to optimum. The ground truth mean and std ( ) of the return distribution is shown together with the mean and std estimate ( ) from the weighted importance sampling surrogate model. The lower confidence bound ($\delta = 0.2$, ---) is shown together with the model's input data (•••). Notice how more or less information is shared between points where data is available depending on the chosen lengthscale parameter Σ.

11.1 Surrogate Model

As discussed in Sec. 10.3, the proposed surrogate model can smoothly interpolate between the Monte Carlo estimate, the importance sampling estimate, and an average of all available returns. In Fig. 30, the available surrogate model predictions are visualized for multiple settings of the model hyper-parameter Σ. In particular, the estimate for expected return (—), return variance (visualizes one standard deviation), and the lower bound of the expected return (---) are pictured for policy evaluations along a random direction around the optimal policy θ^* for the cart pole environment (experimental details can be found in appendix D.2). Trajectory data, which is available to the estimator, is highlighted (•••). The ground-truth return distribution (mean +/- one std.) is computed using the standard MC estimator, based on independent policy rollouts, which are *not* part of the surrogate model.

Stepping from long lengthscales (cf. Fig. 30 (a)) to shorter lengthscales (cf. Fig. 30 (c)), the surrogate model predictions become more local. Most visibly in the lower-bound estimate, the ESS drops significantly when moving away from data points and small model lengthscales, resulting in much higher uncertainty.



(a) Cartpole, [-] $\times 10^5$ steps (b) Mountaincar, [-] $\times 10^5$ steps (c) Swimmer, [-] $\times 10^5$ steps

Figure 31: Policy gradient methods benchmark. The proposed method DD-OPG (——) is compared to standard REINFORCE (——), TRPO (——) and PPO (——) on three continuous control benchmark problems. Mean (——) and standard deviation ( ) of the average return (obtained from 10 independent random seeds) are plotted as a function of the system interaction steps (scaled by 10^5). Significant faster learning speed in the beginning is observed for the model-free off-policy method in comparison to the on-policy PG methods.

11.2 Policy Gradient Benchmark

The proposed DD-OPG method is evaluated in terms of data-efficiency and learning progress in comparison to state-of-the-art policy gradient methods based on Monte Carlo return estimates. In contrast, methods such as DDPG [86] employ TD learning for their value function model and are not part of this evaluation. The benchmark compares DD-OPG to the standard REINFORCE [169] baseline and both TRPO [133] and PPO [134]. All competitor algorithms employ, as it is common practice, the reward-to-go formulation and a linear feature-based baseline for variance reduction. For all methods, hyper-parameters are selected to achieve a maximal accumulated average return, i.e., fast and stable policy optimization. Details about the individual methods' configuration and the employed environments can be found in Appendix D.

The resulting learning performances are visualized in Fig. 31 for the cartpole, mountaincar and swimmer environment (left to right) [43]. For REINFORCE (——), TRPO (——), PPO (——), and DD-OPG (——), the mean average return (solid line) and its confidence intervals (one standard deviation as shaded area) are depicted, as obtained from 10 independent runs out of 10 random seeds for each environment and method. Notice that all competitors operate on fixed-sized batches of environment steps, therefore collecting many more potentially shorter rollouts at the beginning of the learning process to obtain the required number of interactions. In contrast, DD-OPG only obtains one rollout per iteration, no matter the actual length/number of interactions in this specific rollout. To compare the learning speed and data-efficiency between the batch-wise learning competitors and the rollout-based DD-OPG, the results are visualized as a function of collected environment interactions (scaled by 10^5) in Fig. 31.

With DD-OPG, rapid learning progress is achieved already, and the final performance of the competitive, state-of-the-art policy gradient methods is matched. In the hyper-parameter tuning phase, experiments with TRPO and PPO have been conducted based on smaller batch sizes. Still, due to the lack of data-efficient incorporation of off-policy data, no faster *and* stable learning progress could be achieved for these methods than the one visualized in Fig. 31. Notice the large vari-

[86] Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, and Wierstra, "Continuous control with deep reinforcement learning," 2015

[169] Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992

[133] Schulman, Levine, Abbeel, Jordan, and Moritz, "Trust region policy optimization," 2015

[134] Schulman, Wolski, Dhariwal, Radford, and Klimov, "Proximal policy optimization algorithms," 2017

[43] Duan, Chen, Houthooft, Schulman, and Abbeel, "Benchmarking deep reinforcement learning for continuous control," 2016

ance of the DD-OPG learning progress in the swimmer environment. Albeit the superior learning performance of DD-OPG on the swimmer environment, some of the runs got stuck in local minima, resulting in the large variance estimate. The stochastic memory selection partially achieves this trade-off between exploration and exploitation. A mix of prioritized trajectory replay and current trajectories is mandatory to prevent greedy exploitation of previously seen, local minima and to facilitate exploration. Our experiments show that it is mandatory to incorporate previously seen rollout data, as it is done in DD-OPG, to enable rapid progress already in the early stages of training.

11.3 Ablation Study

In the final DD-OPG algorithm, multiple aspects come together: i) the deterministic surrogate model, ii) the memory selection strategy, and iii) the optimization scheme. In this ablation study, we separate the individual components to analyze their effect on the final learning performance. Experiments are conducted on the cart pole environment, and results are averaged over three random seeds.

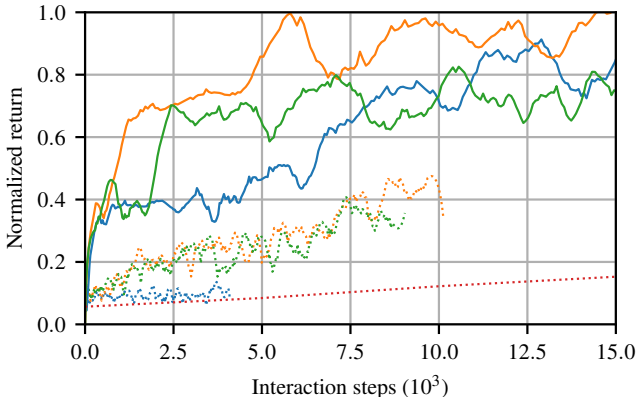


Figure 32: Ablation study of DD-OPG. The full DD-OPG model is constructed from the REINFORCE baseline by iteratively adding i) deterministic off-policy data incorporation and ii) full optimization of the surrogate model. Visualized is the mean learning progress from 3 random seeds on the cartpole environment. REINFORCE (dashed red line) is shown with DD-OPG optimizing only for one gradient step (dotted lines) and fully optimizing the surrogate model (solid lines). For DD-OPG, three levels of history are shown (blue: $N_{max} = 5$, green: $N_{max} = 20$, yellow: $N_{max} = 50$).

In the first experiment, DD-OPG is reconstructed, starting from the REINFORCE baseline. A visualization is shown in Fig. 32. In REINFORCE (red dotted line), only one policy gradient step is taken based on the current on-policy data. This behavior is comparable to DD-OPG with almost no memory ($N_{max} = 5$) and only one step gradient update (visualized as a blue dotted line). Learning performance is already increased by adding more memory paths (green: $N_{max} = 20$, yellow: $N_{max} = 50$). More significantly, the full optimization of the surrogate model (solid lines) achieves much faster progress.

In Fig. 33, the effect of the surrogate model's lengthscale parameter Σ is evaluated. Four different lengthscales $\log \Sigma$ are evaluated (red: 1.0, green: 2.0, yellow: 3.0, blue: 4.0). In this experiment, longer lengthscales improve learning speed despite the introduced model

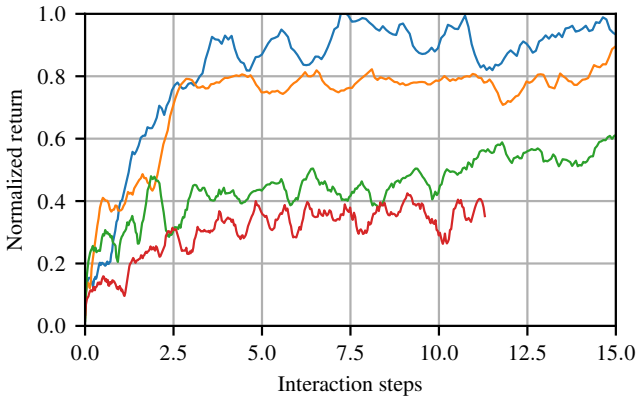


Figure 33: Effect of the surrogate hyper-parameter Σ on the learning progress. Learning speed increases from short lengthscales $\log \Sigma = 1.0$ (——) to $\log \Sigma = 2.0$ (——), $\log \Sigma = 3.0$ (——), and $\log \Sigma = 4.0$ (——). Visualized are DD-OPG mean learning curves from three random seeds as a function of the number of interaction steps with the cartpole environment (scaled by 10^3).

bias. In practice, a fixed log lengthscale of 3.0 is chosen in the benchmark experiments.

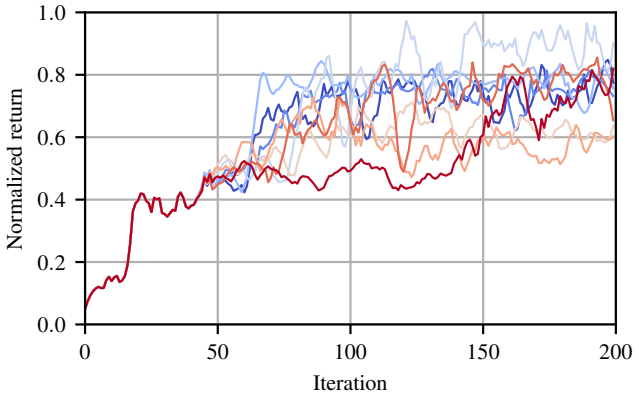


Figure 34: Learning progress for multiple temperature settings for softmax trajectory selection. From lowest temperature ($\lambda = 0.01$, blue) to highest temperature ($\lambda = 2.0$, red). Both too high and too low temperatures lead to suboptimal behaviour, either by too much exploration or too greedy behaviour.

The effects of the softmax temperature λ on the proposed prioritized trajectory replay and the learning progress are depicted in Fig. 34. Explorative behavior is favored for higher temperatures (red), whereas for low temperatures (blue), previous trajectories are selected more greedily. In this example, an intermediate temperature achieves the best trade-off exploration-exploitation trade-off.

12

Summary

The final part of this work revisited data-efficient RL by starting from a model-free perspective with almost no model assumptions. This work explores how fast and efficient learning is possible with minimal and explicit modeling assumptions. The work on Deep-Deterministic Off-Policy Gradients (DD-OPG), as presented in this part, is very much in contrast to the previously presented model-based RL methods, which inherently resulted in strong assumptions about the underlying nature of the system dynamics. Instead, the novel DD-OPG framework is build around a bias-free Monte Carlo estimator of the policy gradient and a single assumption about smoothness in action-space.

This work presents a new surrogate model of the RL return distribution inspired by importance sampling. It can incorporate off-policy data and *deterministic* rollouts to reduce estimator variance. Despite the promising results and the data-efficient learning progress, several exciting topics remain for future work.

The proposed surrogate model is motivated by its close connections to the importance sampling estimator, the interpretability of the model assumption in action space and its desirable behavior in the model limits. A detailed analysis of the resulting model assumptions in policy space, implied by the model assumptions in action space and an analysis of the resulting bias remains an open question.

The proposed optimization scheme empirically achieved good performance in our benchmark experiments, outperforming state-of-the-art methods. However, no additional parametric value function baseline (as in TRPO/PPO) is employed. However, extensions to other strategies for exploration vs. exploitation, for example, acquisition functions like Expected Improvement or Probability of Improvement from Bayesian Optimization [143], are to be explored and directly carry over to the proposed surrogate return model.

Finally, memory selection is required to scale the non-parametric model structure to typical deep RL applications. The proposed prioritized trajectory replay is only one possible option to address this challenge.

[143] Snoek, Larochelle, and Adams, "Practical bayesian optimization of machine learning algorithms," 2012

Epilogue

13

Conclusions

This work's key objective was to leverage the benefits of reinforcement learning methods to automatically and efficiently derive optimal control strategies in industrial problem settings. Several novel policy-search and model-learning methods have been proposed to achieve this objective. These methods are developed to match the limitations of industrial applications and achieve the desired data efficiency. This chapter summarizes the identified problems and solution contributions in the individual parts of this work.

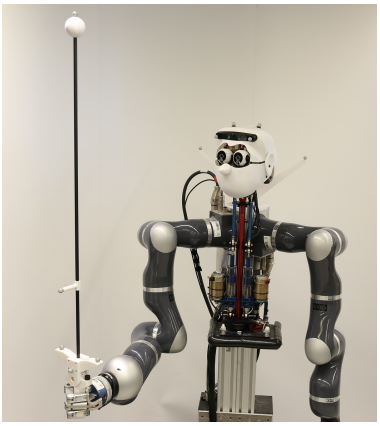


Figure 35: The humanoid upper-body robot, Apollo, as an example of an industrial system.

[38] Desborough and Miller, "Increasing customer value of industrial control performance monitoring—Honeywell's experience," 2001

Part I - Model-Based RL for Tuning of PID Controllers

As a starting point, the first part of this work focuses on a concrete industrial application, PID controller architectures, and their parameters' automatized tuning. According to [38], PID control is involved in 95% of today's controller designs in the process industry. Challenges for existing control design methods are mainly originating from complex, non-linear MIMO systems and coupled multivariate PID structures.

The first part of this work contributes PILCO-PID, a novel, model-based reinforcement learning method to automatically derive optimal PID control parameters for an unknown system. The main contributions in this part are

- **PILCO-PID: an extension of the PILCO to coupled, multivariate PID control architectures.**
  With PILCO, a data-efficient model-based RL method has been adapted to the specific PID control structures. PILCO efficiently learns about an unknown system and tunes the control parameters to achieve the desired control behavior.

- **Bayesian treatment of uncertainties in the model-based PID optimization.**
  A suitable representation of the PID controller allows propagating uncertainty in the model-based predictions. Given this formulation, the actual Gaussian state- and action-distribution can be computed. This Bayesian treatment allows to reason about uncertainty about the system behavior and enforces cautious control updates.

- **Analytically differentiable representation of the PID control architecture.**

Given an extended state representation, the PID controller can be represented as a linear operation. Thus analytic gradients are available to optimize the control parameters in an efficient, gradient-based optimization scheme.

- **Extensions to tracking control.**
  The presented framework can readily be extended to cascaded PID structures and tracking controllers by considering multiple different [32] and time-varying goal states.

[32] Deisenroth and Fox, "Multiple-target reinforcement learning with a single policy," 2011

The proposed PILCO-PID framework has been demonstrated in two real-world applications. Efficient learning of control policies has been demonstrated on a robotic task and an automotive stability control task. For both tasks, learning successful control policies was possible within a few real-world rollouts for data collection.

Despite the promising learning results of the proposed PILCO-PID method, several limitations of the state-of-the-art RL method are apparent in real-world applications. In particular, the sub-problems of model learning and model prediction limit the learning progress. Problems originating from unobserved states (POMDP), noise, and delays, have been found in the aforementioned industrial applications. These applications and systems are vital to successful policy optimization to learn dynamics models geared towards high-quality long-term predictions. Learning good long-term predictive models for RL is mostly unaddressed by current approaches. The second part of this work focuses on appropriate models, inference, and prediction techniques to solve industrial modeling tasks.

Part II - Learning Models for Model-Based RL

In the second part of this work, two novel model learning methods, Multi-Step Gaussian Processes (MSGP) (cf. Sec. 7) and Probabilistic Recurrent State-Space-Models (PR-SSM) (cf. Sec 8) are presented. Both methods have been developed to address the shortcomings of state-of-the-art model learning techniques in model-based RL with the perspective of industrial applications in mind.

With MSGP, a lightweight model learning technique has been introduced, which is tailored for model-learning within the PILCO [31] or PILCO-PID [41] framework. MSGP is a method to achieve improved long-term predictions, which can be straightforwardly applied to other model-based policy search frameworks. The main contributions of MSGP are

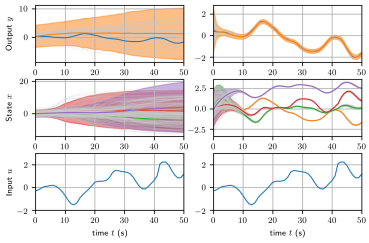


Figure 36: Learning to accurately predict long-term distributions of latent states and observations with the PR-SSM method.

[41] Doerr, Nguyen-Tuong, Marco, Schaal, and Trimpe, "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers," 2017

- **Optimizing the full trajectory distribution for a latent, auto-regressive GP dynamics model.**
  The methodology can be interpreted as optimizing a recurrent, la-

tent GP-NARX dynamics model by maximizing the likelihood of observed system trajectories. This allows incorporating process and observation noise terms and complex non-linear behavior to approximate the real system's long-term behavior.

- **Incorporate knowledge about the policy and action distribution.** Instead of optimizing a model for generic input/output sequences, the MSGP method is tailored for policy search applications, where a policy generates system inputs. Thus, the possible manifold of system inputs and the complexity of the learning problem is reduced by operating on the closed-loop dynamical behavior, given the specific policy class.

- **Incorporation of policy search approximations in model learning.** Approximations are required in the policy search step to obtain a tractable long-term predictive distribution and analytic gradients. The MSGP model-learning incorporates such approximations, e.g., moment matching approximations, to obtain the optimal approximate predictive distribution and thus model. Therefore, long-term predictions consider the errors introduced by the approximations made in the policy search step.

Benchmark results are given for artificial and real-world system identification tasks, comparing MSGP to several state-of-the-art non-linear system identification methods, including recurrent (deep) GPs. While MSGP is competitive to state-of-the-art system identification methods, it outperforms existing approaches on the robot policy search task. Using MSGP, iterative learning of a challenging robotic task from scratch is possible. In contrast, learning this task was impossible with the default GP-NARX model learning framework without providing additional prior knowledge, such as suitable GP hyper-parameters.

There remain two significant caveats in the model-learning framework, which are addressed by the second model-learning method presented in this work. In PILCO and related algorithms, a moment-matching approximation is employed, and a Gaussian distribution approximates future state distributions. However, more flexible predictive distributions would be required to capture a real system's complex, non-linear behavior over a long prediction horizon. At the same time, representing arbitrary complex predictive distributions is restricted by computational requirements. As a second drawback, auto-regressive models need to specify which historical information is required to capture the full state of a system. Specifying the model input for a real system is non-trivial and can easily lead to situations with too little information or too large model-input spaces.

The Probabilistic Recurrent State-Space Model (PR-SSM) is introduced to address the previously mentioned model-learning problems.

It comprises a novel model structure and efficient inference scheme for learning probabilistic and Markovian state-space models. The main novel contributions of PR-SSMs are

- **Representation of complex non-Gaussian state distributions in Gaussian-process state-space models.**
  The PR-SSM model can represent arbitrary latent-state distributions, which might arise from the learned non-linear system dynamics.

- **Efficient and computationally tractable inference scheme.**
  Based on GP priors and doubly stochastic variational inference, a novel model optimization criterion is derived, which is closely related to the one of powerful but deterministic RNNs or LSTMs. A computationally tractable trade-off between the accurate representation of the latent state distribution and computational requirements is possible with the sampling-based inference scheme.

- **Temporal dependencies in the state evolution allow to optimize of long-term predictions efficiently.**
  By maintaining the latent state distribution's temporal correlation and thereby enabling long-term gradients, efficient inference in latent space becomes feasible.

- **Integrated recognition model to scale the inference scheme to large datasets.**
  A novel recognition model learns a latent state estimate for each part of the trajectory data. This model enables learning in systems with unstable or slow dynamics and facilitates scalability to large datasets.

Robustness and high performance in model learning are demonstrated on real-world datasets in comparison to state-of-the-art methods. The scalability of the proposed inference scheme is demonstrated on a large-scale dataset from real-world robotic data.

Part III - Model Assumptions in Model-Free RL

Learning from the available interaction data in the previous parts of this work was limited to learning a model of the dynamics and employing this (probabilistic) model in a planning scheme to optimize the policy. Even for sophisticated, Bayesian treatments to make the best use of all available data, learning a model is always limited by the amount of available data and the number of prior assumptions. In situations where strong non-linearities or discontinuities are present, dynamics models cannot be efficiently learned. This is, prior knowledge about

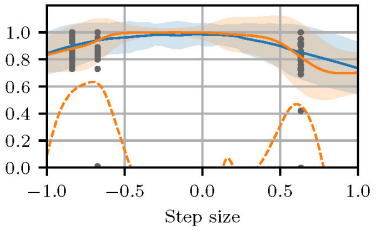


Figure 37: Predictions of the DD-OPG expected cost estimators based on action smoothness assumptions.

the dynamics or excessive amounts of data would be required to capture the actual system behavior. Instead, one might consider another, potentially smoother space to learn from and model the available data. In the final part of this work, the focus is thus shifted to *model-free* RL methods and how model assumptions in the space of expected return can be incorporated to enable fast and efficient learning.

In this part, the novel Deep Deterministic Off-Policy Gradient (DDOPG) method is presented. It is based on several insights on how to increase data efficiency by reintroducing model assumptions into model-free policy gradient methods.

- **Incorporation of all available data into the PG estimator.**
  A novel surrogate model of the expected return distribution is presented, which is inspired by importance sampling. This model allows the usage of all available on- and off-policy data, which even might originate from a deterministic policy.

- **Variance reduction for efficient learning.**
  Two mechanisms are presented to reduce the high variance of standard PG estimators. A normalized importance sampling is shown to act similar to a baseline term, which is commonly utilized for variance reduction. Furthermore, with a single model assumption about smoothness in action space, a trade-off between generalization between data points (bias) and variance is achieved.

- **Memory selection with prioritized trajectory replay.**
  A suitable memory selection scheme is introduced to scale the non-parametric surrogate model to large-scale problems typically encountered in deep RL applications.

The proposed optimization scheme empirically achieved good performance in our benchmark experiments, outperforming state-of-the-art methods, even though no additional parametric value function baseline (as in TRPO/PPO) is employed.

Outlook

Contributions in this work towards the deployment of Reinforcement Learning in industrial applications are primarily out of two categories: (i) Making the most out of available data, and (ii) tailoring parts of the RL method to the overall RL objective or domain-specific requirements. In both regimes, several questions are raised by this work, which remain open for future work.

The goal of leveraging data in the best possible way mainly resulted in novel contributions regarding efficient and approximately accurate Bayesian modeling and inference techniques. Despite these

advances, it is primarily unclear which functional mapping we should consider and what assumptions might be valid for a specific problem or a given class of problems. For example, in RL, it is unclear what assumptions and prior knowledge could help model a system's dynamics, let alone introducing meaningful prior structure into value or expected return models. Advances in meta-learning techniques, model- or architecture search and Bayesian hyper-parameter selection are pointing towards solutions for this problem. However, these methods are usually rather data-inefficient and not at all easily transferred to one-of-a-kind industrial problems.

In the second category, considerable overlap with more mature, related fields, such as time-series modeling, system identification, or optimal control, can be exploited to advance the field of model-based RL. Most commonly used RL methods still deploy simple, off-the-shelf sub-components, e.g., for model-learning, which results in poor performance. Continuing to tailor these methods to the structural assumptions and requirements in the MDP/POMDP-style RL setting and the specific problem classes introduced by real industrial applications is promising for future research.

Appendix

Appendix: PILCO-PID

Model-based RL with probabilistic, non-linear system models, typically results in complex state-action-distributions. The PILCO framework [31] is build around a GP model and a moment matching approximation to keep the long-term, state-action distribution analytically tractable. In particular, the state-action distribution in each time-step is Gaussian. In Sec. I of this work, a novel tuning method for multivariate and coupled PID controllers based on the PILCO framework has been proposed. Fundamentally, this method is build on the insight that any PID control structure can be rewritten as a linear state-feedback controller given an extended system state. Furthermore, the specific extensions of the system state can be written as linear functions of the state itself. Consequently, the proposed PILCO-PID variant can exploit the fact a Gaussian random variable remains Gaussian under linear transformation. Thus, the full, probabilistic interpretation of the original PILCO framework remains intact for the proposed PILCO-PID method. The following section states the required results for distributions and derivatives of linear transformations on Gaussian random variables.

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

A.1 Transformation of Gaussian Random Variables

For a linear transformation of a Gaussian random variable $\mathbf{X} \sim \mathcal{N}(\cdot \mid \mu_X, \Sigma_X) \in \mathbb{R}^D$ given by

$$\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b} \tag{116}$$

where $\mathbf{A} \in \mathbb{R}^{P \times D}$ and $\mathbf{b} \in \mathbb{R}^{P \times 1}$, it can be shown that the random variable $\mathbf{Y}$ is Gaussian as well, i.e.

$$\mathbf{Y} \sim \mathcal{N}(\mu_Y, \Sigma_Y) \in \mathbb{R}^P \tag{117}$$

Also the joint probability distribution of $\mathbf{X}$ and $\mathbf{Y}$ is Gaussian and given by

$$p(\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{bmatrix}) = \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_Y \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_X & \boldsymbol{\Sigma}_X \boldsymbol{C} \\ \boldsymbol{C}^T \boldsymbol{\Sigma}_X^T & \boldsymbol{\Sigma}_Y \end{bmatrix}) \tag{118}$$

where

$$\mu_Y = \boldsymbol{A}\mu_X + \mathbf{b}, \quad \Sigma_Y = \mathbf{A}\Sigma_X\mathbf{A}^T, \quad \mathbf{C} = \mathbf{A}^T \tag{119}$$

The non-zero partial derivatives of Y's sufficient statistics are given by

$$\frac{\delta \mu_Y}{\delta \mu_X} = A \in \mathbb{R}^{P \times D} \tag{120}$$

$$\frac{\delta \mu_Y}{\delta A} = \mu_X^T \otimes I \in \mathbb{R}^{P \times DP} \tag{121}$$

$$\frac{\delta \mu_Y}{\delta b} = I \in \mathbb{R}^{P \times P} \tag{122}$$

$$\frac{\delta \Sigma_Y}{\delta \Sigma_X} = A \otimes A \in \mathbb{R}^{P^2 \times D^2} \tag{123}$$

$$\frac{\delta (\Sigma_Y)_{kl}}{\delta A_{ij}} = \delta_{lj}(A^T \Sigma_X)_{ki} + \delta_{kj}(\Sigma_X A)_{il} \qquad \frac{\delta (\Sigma_Y)}{\delta A} \in \mathbb{R}^{P^2 \times PD} \tag{124}$$

$$\frac{\delta C_{kl}}{\delta A_{ij}} = \delta_{il}\delta_{kj} \qquad \frac{\delta C}{\delta A} \in \mathbb{R}^{PD \times D^2} \tag{125}$$

where $\otimes$ is the Kronecker product and δ_{ij} is the Kronecker delta (cf. [120] for useful matrix derivatives).

[120] Petersen and Pedersen, "The matrix cookbook," 2008

B

Appendix: MSGP

This chapter presents additional material about the Multi-Step Gaussian Process (MSGP) model. Information about the model learning benchmark can be found in Sec. B.1, including details about the employed reference methods in Sec. B.1.1 and datasets in Sec. B.1.2. The moment matching approximation utilized in the uncertainty propagation is summarized in Sec. B.2.

B.1 Model Learning Benchmark

The proposed model learning method MSGP is compared to several state-of-the-art methods. Details about the model configuration and the employed benchmark datasets can be found in the following sections.

B.1.1 Benchmark Methods

Each method utilizes 100 inducing inputs and moment matching is applied for long-term predictions.

(i) GP-NARX [76]
The system dynamics is modeled as a function of the history of observations and inputs. A squared exponential kernel with automatic relevance determination is employed and hyperparameters are optimized based on the maximum likelihood objective.

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

(ii) PILCO's GP-NARX [31]
PILCO adds penalty terms for large signal to noise ratios σ_f^2/σ_n^2 and extreme length-scales l_i^2 to the log marginal data likelihood to avoid numerical instabilities when optimizing the GP hyperparameters. Especially on real datasets, the standard GP hyperparameter optimization tends to underestimate the noise, which is prevented by this heuristic.

[31] Deisenroth and Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," 2011

(iii) NIGP [94]
This method explicitly accounts for uncertainty in the input by treating input points as deterministic and inflating the corresponding output uncertainty. This leads to state-dependent noise proportional to the local gradient of the GP posterior mean. For time-series data, both input and output noise are the same, which can be exploited by this framework.

[94] McHutchon and Rasmussen, "Gaussian process training with input noise," 2011

(iv) REVARB [91]
Recurrent Variational Bayes (REVARB) is a recent proposition to optimize the lower bound to the log-marginal likelihood $\log p(\boldsymbol{y})$ using variational techniques. This framework is based on the variational sparse GP framework [152], but allows for computation of (time-)recurrent GP structures and deep GP structures (stacking multiple GP-layers in each time-step). A Python-implementation is available online [92]. For our benchmark, we run REVARB using one (REVARB1) respectively two (REVARB2) hidden layers, where each layer is provided with 100 inducing inputs. We closely follow the original setup as presented by [91], performing 50 initial optimization steps based on fixed variances and up to 10000 steps based on variable variances. Unlike for the other benchmark methods, the autoregressive history of REVARB implicitly becomes longer when introducing additional hidden layers.

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[152] Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," 2009

[92] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, *REVARB implementation for RGP*, 2015

(v) MSGP
The proposed MSGP model learning method is implemented in Python using *autograd* to automatically derive the loss gradients. The latent GP-NARX model is initialized using a sparse GP model trained on the observations $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$. We optimized the GP hyperparameters and targets by gradient descent using the Adam optimizer.

B.1.2 Benchmark Datasets

The benchmarks datasets are composed of popular system identification datasets from related work [108, 76, 93]. For all of these problems, both the inputs and outputs are one-dimensional $D_u = D_y = 1$. However, the system's true state is higher dimensional such that an autoregressive history or an explicit latent state representation is required to capture the relevant dynamics. The number of historic inputs and outputs for the autoregressive methods is fixed a-priori for each dataset.

[108] Narendra and Parthasarathy, "Identification and control of dynamical systems using neural networks," 1990

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[93] Mattos, Damianou, Barreto, and Lawrence, "Latent Autoregressive Gaussian Processes Models for Robust System Identification," 2016

Synthetic Datasets

The synthetic benchmarks are taken from existing literature and are described by difference equations. The first four synthetic systems have been presented in [108], whereas the fifth originates from [76]. They have been used as benchmarks to assess robust system identification in the presence of outliers in [93]. The systems' training and test data as well as observation noise are summarized in Tab. 3.

Task	ODE	Training	Test	Noise
Synthetic system 1	$y_t = \frac{y_{t-1}y_{t-2}(y_{t-1}+2.5)}{1+y_{t-1}^2+y_{t-2}^2} + u_{t-1}$	$u_t = \mathrm{U}(-2,2)$ 300 samples	$u_t = \sin(2\pi t/25)$ 100 samples	$\mathcal{N}(0, 0.29)$
Synthetic system 2	$y_t = \frac{y_{t-1}}{1+y_{t-1}^2} + u_{t-1}^3$	$u_t = \mathrm{U}(-2,2)$ 300 samples	$u_t = \sin(2\pi t/25) + \sin(2\pi t/10)$ 100 samples	$\mathcal{N}(0, 0.65)$
Synthetic system 3	$y_t = (u_{t-1} - 0.8)u_{t-1}(u_{t-1} + 0.5) + 0.8y_{t-1}$	$u_t = \mathrm{U}(-1,1)$ 300 samples	$u_t = \sin(2\pi t/25)$ 100 samples	$\mathcal{N}(0, 0.07)$
Synthetic system 4	$y_t = 0.6y_{t-2} + 0.3\sin(3\pi u_{t-1}) + 0.1\sin(5\pi u_{t-1}) + 0.3y_{t-1}$	$u_t = \mathrm{U}(-1,1)$ 500 samples	$u_t = \sin(2\pi t/250)$ 500 samples	$\mathcal{N}(0, 0.18)$
Synthetic system 5	$y_t = y_{t-1} - 0.5\tanh(y_{t-1} + u_{t-1}^3)$	$u_t = \mathcal{N}(u_t\|0,1)$ $-1 \leq u_t \leq 1$ 150 samples	$u_t = \mathcal{N}(u_t\|0,1)$ $-1 \leq u_t \leq 1$ 100 samples	$\mathcal{N}(0, 0.0025)$

Table 3: Artificial datasets used in the benchmark experiments (cf. [93]).

Real-World Datasets

The real-world benchmark datasets are composed from typical system identification problems from technical systems like hydraulic actuators, furnaces, hair dryers or electrical motors. References to the individual datasets, training and test trajectory length and the utilized history for the GP-NARX models are summarized in Tab. 4.

B.2 Moment Matching

In the moment matching approximation, an intractable distribution is approximated by a Gaussian distribution having its mean and variance. In order to propagate a Gaussian input through the non-linear GP dynamics model, the following integral has to be solved

$$p(f_{t+1}) = \int p(f_{t+1}|\bar{\boldsymbol{x}}_t)p(\bar{\boldsymbol{x}}_t)d\bar{\boldsymbol{x}}_t\,. \tag{126}$$

First and second moment of the predictive distribution can be calculated in closed form if a Gaussian-like kernel is employed as derived in [124]. The approximated predictive distribution for a Gaussian distributed input $\bar{\boldsymbol{x}}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\bar{x}}, \boldsymbol{\Sigma}_{\bar{x}})$ is given by

$$p(f_{t+1}) = \mathcal{N}(\mu_{f_{t+1,\mathrm{MM}}}, \sigma_{f_{t+1,\mathrm{MM}}})\,, \tag{127}$$

where the mean is calculated as

$$\mu_{f_{t+1,MM}} = \boldsymbol{q}\,\boldsymbol{\beta}^T\,, \tag{128}$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_m]^T = \boldsymbol{K}^{-1}\bar{\boldsymbol{y}}$ and

$$q_i = |\Lambda^{-1}\boldsymbol{\Sigma}_{\bar{x}} + \boldsymbol{I}|^{-1/2}\exp(-\frac{1}{2}(\boldsymbol{\mu}_{\bar{x}} - \bar{\boldsymbol{x}}_i)^T(\boldsymbol{\Sigma}_{\bar{x}} + \Lambda)^{-1}(\boldsymbol{\mu}_{\bar{x}} - \bar{\boldsymbol{x}}_i))\,. \tag{129}$$

[124] Quinonero-Candela, Girard, and Rasmussen, *Prediction at an uncertain input for Gaussian processes and relevance vector machines-application to multiple-step ahead time-series forecasting*, 2002

Task	Training	Test	History
Hydraulic actuator [110]	512	512	$l_y = l_u = 10$
Ball balancing [100]	500	500	$l_y = l_u = 10$
Electrical drives [167]	250	250	$l_y = l_u = 10$
Gas furnace [100]	148	148	$l_y = l_u = 3$
Hair dryer [100]	500	500	$l_y = l_u = 2$

Table 4: Summary of the real-world dataset characteristics. For each dataset, the lengths of the training and test trajectories are given together with the number of historic states employed for the NARX dynamics models.

The variance is obtained as

$$\sigma_{f_{t+1},\mathrm{MM}} = \sigma^2_{f_{t+1}} + \mathrm{Tr}(\boldsymbol{K}^{-1}(\boldsymbol{k}\boldsymbol{k}^T - \boldsymbol{L})) + \mathrm{Tr}(\boldsymbol{\beta}\boldsymbol{\beta}^T(\boldsymbol{L} - \boldsymbol{q}\boldsymbol{q}^T)), \quad (130)$$

and $\boldsymbol{L}$ is given by

$$L_{ij} = k(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{x}}_i)k(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{x}}_j)|2\Lambda^{-1}\boldsymbol{\Sigma}_{\bar{x}} + \boldsymbol{I}|^{-1/2} \exp(2(\boldsymbol{\mu}_{\bar{x}} - \bar{\boldsymbol{x}}_d)^T\Lambda^{-1}(2\Lambda^{-1} + \boldsymbol{\Sigma}_{\bar{x}}^{-1})^{-1}\Lambda^{-1}(\boldsymbol{\mu}_{\bar{x}} - \bar{\boldsymbol{x}}_d)), \quad (131)$$

where $\bar{\boldsymbol{x}}_d = 0.5(\bar{\boldsymbol{x}}_i + \bar{\boldsymbol{x}}_j)$. A detailed derivation of the moment matching approximation for GPs based on the SE kernel can be found in [124].

[124] Quinonero-Candela, Girard, and Rasmussen, *Prediction at an uncertain input for Gaussian processes and relevance vector machines-application to multiple-step ahead time-series forecasting*, 2002

C Appendix: PR-SSM

This supplementary material provides details about the derivations and configuration of the proposed PR-SSM in Sec. C.1. The following section C.2 elaborates on the model learning benchmark. This includes details about the reference methods and the employed datasets. Finally, additional experimental results from PR-SSM learning, the model learning benchmark, and the large scale experiment are summarized in Sec. C.3.

C.1 PR-SSM Model Derivations and Configuration

C.1.1 Evidence Lower Bound (ELBO)

Summarizing the model assumptions from the main paper, the model's joint distribution is given by

$$\begin{aligned} p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z}) = &\left[\prod_{t=1}^{T} p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\right] \\ &\left[\prod_{t=2}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{f}_t)\right] \\ &\left[\prod_{t=2}^{T}\prod_{d=1}^{D_x} p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d) p(\boldsymbol{z}_d)\right] \\ &p(\boldsymbol{x}_1)\,. \end{aligned} \tag{132}$$

The variational distribution over the unknown model variables is defined as

$$\begin{aligned} q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z}) = &\left[\prod_{t=2}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{f}_t)\right] \\ &\left[\prod_{t=2}^{T}\prod_{d=1}^{D_x} p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d) q(\boldsymbol{z}_d)\right] \\ &q(\boldsymbol{x}_1)\,. \end{aligned} \tag{133}$$

Together, the derivation of the ELBO is given below in (134) to (135).

Parameter	Initialization
Inducing inputs	$\boldsymbol{\zeta}_d \sim \mathcal{U}(-2, 2) \in \mathbb{R}^{P \times (D_x + D_u)}$
Inducing outputs	$q(\boldsymbol{z}_d) = \mathcal{N}(\boldsymbol{z}_d \mid \boldsymbol{\mu}_d, \boldsymbol{\Sigma}_d) \in \mathbb{R}^P$ $\mu_{d,i} \sim \mathcal{N}(\mu_{d,i} \mid 0, 0.05^2)$ $\boldsymbol{\Sigma}_d = 0.01^2 \cdot \boldsymbol{I}$
Process noise	$\sigma^2_{\mathrm{x},i} = 0.002^2 \quad \forall i \in [1, D_\mathrm{x}]$
Sensor noise	$\sigma^2_{\mathrm{y},i} = 1.0^2 \quad \forall i \in [1, D_\mathrm{y}]$
kernel hyper-parameters	$\sigma_f^2 = 0.5^2$ $l_i^2 = 2 \quad \forall i \in [1, D_x]$

Table 5: Default configuration for the initialization of the PR-SSM (hyper-) parameters $\theta_{\text{PR-SSM}}$. This configuration has been employed for all experiments in the benchmark section.

$$\log p(\boldsymbol{y}_{1:T}) \geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\log \frac{p(\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})}{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \right] \tag{134}$$

$$\begin{aligned}
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\log \frac{\left[\prod_{t=1}^T p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\right] \left[\prod_{t=2}^T p(\boldsymbol{x}_t \mid \boldsymbol{f}_t)\right] \left[\prod_{t=2}^T \prod_{d=1}^{D_x} p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d) p(\boldsymbol{z}_d)\right] p(\boldsymbol{x}_1)}{\left[\prod_{t=2}^T p(\boldsymbol{x}_t \mid \boldsymbol{f}_t)\right] \left[\prod_{t=2}^T \prod_{d=1}^{D_x} p(f_{t,d} \mid \hat{\boldsymbol{x}}_{t-1}, \boldsymbol{z}_d) q(\boldsymbol{z}_d)\right] q(\boldsymbol{x}_1)} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\log \frac{\left[\prod_{t=1}^T p(\boldsymbol{y}_t \mid \boldsymbol{x}_t)\right] \left[\prod_{d=1}^{D_x} p(\boldsymbol{z}_d)\right] p(\boldsymbol{x}_1)}{\left[\prod_{d=1}^{D_x} q(\boldsymbol{z}_d)\right] q(\boldsymbol{x}_1)} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\log \prod_{t=1}^T p(\boldsymbol{y}_t \mid \boldsymbol{x}_t) \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\sum_{d=1}^{D_x} \log \frac{p(\boldsymbol{z}_d)}{q(\boldsymbol{z}_d)} \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}, \boldsymbol{f}_{2:T}, \boldsymbol{z})} \left[\log \frac{p(\boldsymbol{x}_1)}{q(\boldsymbol{x}_1)} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T})} \left[\log \prod_{t=1}^T p(\boldsymbol{y}_t \mid \boldsymbol{x}_t) \right] + \mathbb{E}_{q(\boldsymbol{z})} \left[\sum_{d=1}^{D_x} \log \frac{p(\boldsymbol{z}_d)}{q(\boldsymbol{z}_d)} \right] + \mathbb{E}_{q(\boldsymbol{x}_1)} \left[\log \frac{p(\boldsymbol{x}_1)}{q(\boldsymbol{x}_1)} \right] \\
&= \sum_{t=1}^T \mathbb{E}_{q(\boldsymbol{x}_t)} \left[\log p(\boldsymbol{y}_t \mid \boldsymbol{x}_t) \right] + \sum_{d=1}^{D_x} \mathrm{KL}(q(\boldsymbol{z}_d) \parallel p(\boldsymbol{z}_d)) + \mathrm{KL}(q(\boldsymbol{x}_1) \parallel p(\boldsymbol{x}_1))
\end{aligned} \tag{135}$$

In the ELBO, as derived in (135), the last term is a regularization on the initial state distribution. For the full gradient-based optimization in the main paper, an uninformative initial distribution is chosen and fixed, such that the third term is dropped. In the stochastic optimization scheme, this term acts as a regularization preventing the recognition model to become overconfident in its predictions.

C.1.2 Model Configuration

The PR-SSM exhibits a large number of model (hyper-) parameters $\theta_{\text{PR-SSM}}$ which need to be initialized. However, empirically, most of these model parameters can be initialized to a default setting as given in Tab. 5. This default configuration has been employed for all benchmark experiments presented in the main paper.

Parameter	Initialization
Inducing points	$P = 20$
State samples	$N = 50$
Subtrajectories	$N_{\text{batch}} = 10$
	$T_{sub} = 100$
Latent space	$D_x = 4$

Table 6: Structural configuration of the PR-SSM as utilized in the benchmark experiments.

The PR-SSM's latent state dynamics model and noise models are configured to initially exhibit a random walk behavior. This behavior is clearly visible for the prediction based on the untrained model in Fig. 26 of the main text. The GP prior is approximating the identity function based on an identity mean function and almost zero inducing outputs (up to a small Gaussian noise term to avoid singularities). The inducing inputs are spread uniformly over the function's domain. The noise processes are initializes such as to achieve high correlations between latent states over time (i.e. small process noise magnitude). At the same time, a larger observation noise is required to obtain a inflation of predictive uncertainty over time. This inflation of predictive uncertainty is again clearly visible in Fig. 26 of the main text. Both noise terms are chosen in a way to obtain numerically stable gradients for both the sample based log likelihood and the backpropagation through time in the ELBO evaluation.

The number of samples used in the ELBO approximation, number of inducing points in the GP approximation and batch size are, in contrast, a trade-off between model accuracy and computational speed. The proposed default configuration empirically showed good performance whilst being computationally tractable.

Two tuning parameters remain, which are problem specific and have to be chosen for each dataset individually. Depending on the true system's timescales/sampling frequency and system order, the length of subtrajectories T_{sub} for minibatching and the latent state dimensionality D_x have to be specified manually. For the benchmark datasets we choose $T_{sub} = 100$ and $D_x = 4$.

C.2 Model Learning Benchmark Details

In the main text, the proposed PR-SSM's long-term predictive performance is compared to several state-of-the-art methods. The benchmark is set up similar to the evaluation presented in [39]. Details about the individual benchmark methods, their configuration and the employed datasets can be found in the following sections. Minor adjustments with respect to the setup in [39] will be pointed out in the

following. These modifications have been introduced to enable fair comparison between all benchmark methods.

	N_{train}	N_{test}	L_u, L_y
Actuator [110]	512	512	10
Ballbeam [100]	500	500	10
Drives [167]	250	250	10
Furnace [100]	148	148	3
Dryer [100]	500	500	2

Table 7: Summary of the real-world, non-linear system identification benchmark tasks. All datasets are generated by recording input/output data of actual physical plants. For each dataset, the lengths of training and test set are given together with the number of past input and outputs used for the NARX dynamics models.

C.2.1 Benchmark Methods

The proposed PR-SSM is evaluated in comparison to methods from three classes:

1. One-step ahead autoregressive models (GP-NARX, NIGP)
2. Multi-step ahead autoregressive models in latent space (REVARB, MSGP)
3. Markov state-space models (SS-GP-SSM)

To enable a fair comparison, all methods have access to a predefined amount of input/output data for initialization.

(i) GP-NARX [76]
The system dynamics is modeled as

$$y_{t+1} = f(y_t, \dots, y_{t-L_y}, u_t, \dots, u_{t-L_u}) \tag{136}$$

with a GP prior on f. The GP has a zero mean function and a squared exponential kernel with automatic relevance determination. The kernel hyper-parameters, signal variance and length-scales, are optimized based on the standard maximum likelihood objective. A sparse approximation [142], based on 100 inducing inputs is employed. Moment matching [58] is employed to obtain a long-term predictive distribution.

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[142] Snelson and Ghahramani, "Sparse Gaussian processes using pseudo-inputs," 2006

[58] Girard, Rasmussen, Quinonero-Candela, Murray-Smith, Winther, and Larsen, "Multiple-step ahead prediction for non linear dynamic systems—a Gaussian process treatment with propagation of the uncertainty," 2003

(ii) NIGP [94]
Noise Input GPs (NIGP) account for uncertainty in the input by treating input points as deterministic and inflating the corresponding output uncertainty, leading to state dependent noise, i.e. heteroscedastic GPs. The experimental results are based on the publicly available Matlab code. Since no sparse version is available, training is performed on the full training dataset. Training on the full dataset is however not possible for larger datasets and provides

[94] McHutchon and Rasmussen, "Gaussian process training with input noise," 2011

an advantage to NIGP. Experiments based on a random data subset of size 100 lead to decreased performance in the order of the GP-NARX results or worse.

(iii) REVARB [91]
Recurrent Variational Bayes (REVARB) is a recent proposition to optimize the lower bound to the log-marginal likelihood $\log p(\boldsymbol{y})$ using variational techniques. This framework is based on the variational sparse GP framework [152], but allows for computation of (time-)recurrent GP structures and deep GP structures (stacking multiple GP-layers in each time-step). For our benchmark, we run REVARB using one (REVARB1) respectively two (REVARB2) hidden layers, where each layer is provided with 100 inducing inputs. We closely follow the original setup as presented by [91], performing 50 initial optimization steps based on fixed variances and up to 10000 steps based on variable variances. Unlike for the other benchmark methods, the autoregressive history of REVARB implicitly becomes longer when introducing additional hidden layers.

[91] Mattos, Dai, Damianou, Forth, Barreto, and Lawrence, "Recurrent Gaussian processes," 2015

[152] Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," 2009

(iv) MSGP [39]
MSGP is a GP-NARX model operating in a latent, noise free state, which is trained by optimizing its long-term predictions. The experimental results are obtained according to the configuration described in [39], again using 100 inducing points and moment matching.

[39] Doerr, Daniel, Nguyen-Tuong, Marco, Schaal, Toussaint, and Trimpe, "Optimizing Long-term Predictions for Model-based Policy Search," 2017

(v) SS-GP-SSM [149]
The Sparse-Spectrum GP-SSM is employing a sparse spectrum GP approximation to model the system's transition dynamics in a Markovian, latent space. The available Matlab implementation is restricted to a 2D latent space. In the experimental results, a default configuration is employed as given by: $K = 2000, N = 40, n\_basis\_u = n\_basis\_x = 7$. The variables are defined as given in the code published for [149].

[149] Svensson and Schön, "A flexible state–space model for learning nonlinear dynamical systems," 2017

C.2.2 Benchmark Datasets

The benchmarks datasets are composed of popular system identification datasets from related work [108, 76, 93]. They incorporate measured input output data from technical systems like hydraulic actuators, furnaces, hair dryers or electrical motors. For all of these problems, both inputs and outputs are one-dimensional $D_u = D_y = 1$. However, the system's true state is higher dimensional such that an autoregressive history or an explicit latent state representation is required to capture the relevant dynamics. The number of historic inputs and outputs for the autoregressive methods is fixed a-priori for each

[108] Narendra and Parthasarathy, "Identification and control of dynamical systems using neural networks," 1990

[76] Kocijan, Girard, Banko, and Murray-Smith, "Dynamic systems identification with Gaussian processes," 2005

[93] Mattos, Damianou, Barreto, and Lawrence, "Latent Autoregressive Gaussian Processes Models for Robust System Identification," 2016

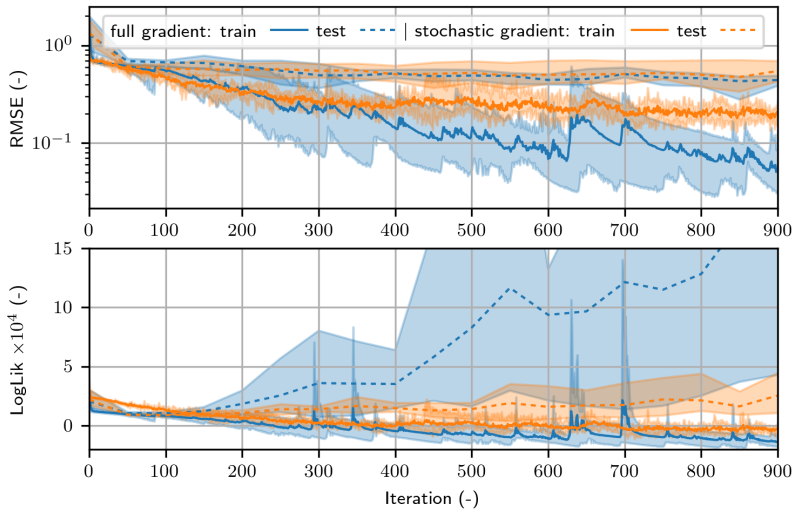


Figure 38: Comparison of the learning progress of the proposed method on the *Drive* dataset given the full ELBO gradient (—) and the stochastic gradient (—), based on minibatches and the recognition model. RMSE and log likelihood results over learning iterations are shown for the free simulation on training (—) and test (- - -) dataset. The full gradient optimization scheme overfitts (in particular visible in the log likelihood) and exposes a difficult optimization objective (cf. spikes in model loss). Stochastically optimizing the model-based on the proposed minibatched ELBO estimates and employing the recognition model significantly reduces overfitting and leads to more robust learning.

dataset as previously used in other publications. For model training, datasets are normalized to zero mean and variance one based on the available training data. References to the individual datasets, training and test trajectory length, and the utilized history for the GP-NARX models are summarized in Tab. 7.

C.3 Additional Results

C.3.1 Optimization Schemes Comparison

In Fig. 38, the RMSE and the negative log likelihood, which is obtained for the model's long-term prediction, is depicted over learning iterations for the training- (solid line) and test- (dotted line) set from the *Drives* dataset. The full gradient optimization (blue) obtains smaller training loss in comparison to the stochastic optimization scheme for both RMSE and negative log likelihood. The resulting test performance however indicates similar performance in terms of RMSE whilst showing clear overfitting of the full-gradient-based model in terms of log likelihood. Additionally, optimizing, based on the full gradient, is much more delicate and less robust as indicated by the spikes in loss and the higher variance of incurred optimization progress. Fig. 38 depicts mean (lines) and minimum to maximum intervals (shaded areas) of incurred loss, based on five independent model trainings.

C.3.2 Detailed Benchmark Results

In Tab. 8, detailed results are provided for the benchmark experiments. The reference learning methods in the presented benchmark are highly

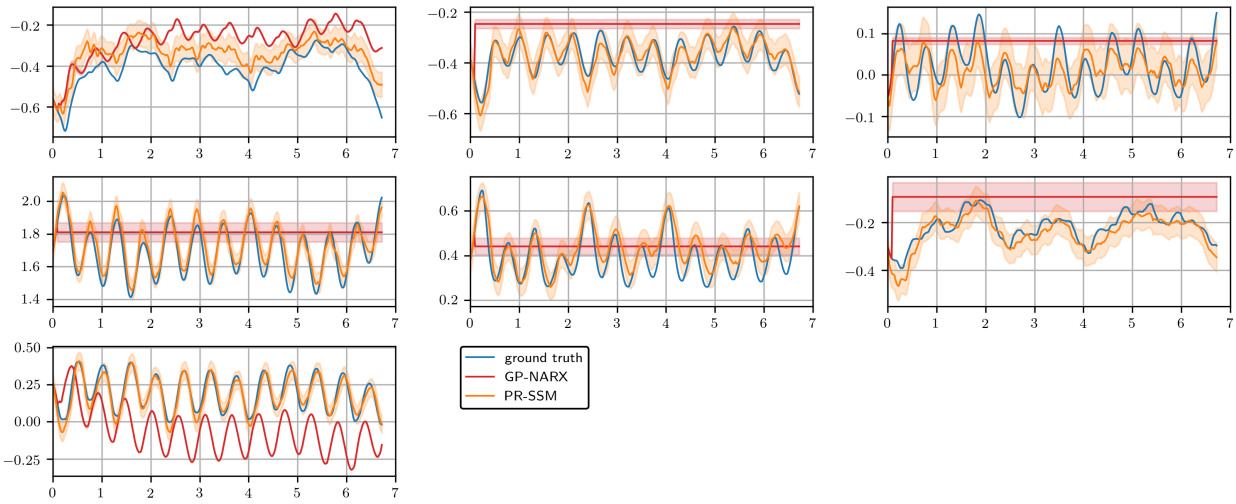


Figure 39: Detailed results from the Sarcos large scale task: Predictions from the GP-NARX model ( ) and the PR-SSM ( ) for all seven joint positions as obtained for the first test experiment on top of the measured, ground-truth joint positions (——). PR-SSM is clearly able to capture the robot arm dynamics, whereas the GP-NARX model only successfully captures a rough model of the robot arm dynamics for two out of seven joints.

deceptive to changes in the data pre-processing and the long-term prediction method. Therefore, results are detailed for GP-NARX, NIGP, REVARB 1, and REVARB 2 for all combinations of normalized/unnormalized training data and mean or moment matching predictions. The results for methods MSGP, SS-GP-SSM and PR-SSM are always computed for the normalized datasets using the method specific propagation of uncertainty schemes. The RMSE result (mean (std) over 5 independently learned models) is given for the free simulation on the test dataset in Tab. 8. For each dataset, the best result (solid underline) and second best result (dashed underline) is indicated. The proposed PR-SSM consistently outperforms the reference (SS-GP-SSM) in the class of Markovian state space models and robustly achieves performance comparable to the one of state-of-the-art latent, autoregressive models.

Obtaining uncertainty estimates is one key requirement for employing the long-term predictions, e.g. in model-based control. Therefore, only the predictive results based on the approximate propagation of uncertainty through moment matching is considered in the main paper, although better results in RMSE are sometimes obtained from employing only the mean predictions. A comparison of the predictive results based on mean and moment matching predictions on the *Drives* dataset is shown in Fig. 5. The results from the unnormalized datasets and moment matching are in line with the results published in [39].

C.3.3 Large Scale Experiment Details

The *Sarcos* task is based on a publicly available dataset comprising joint positions, velocities, acceleration and torques of a seven degrees-

of-freedom SARCOS anthropomorphic robot arm. This dataset has been previously used in [159, 168] in the task of learning the system's inverse dynamics, therefore mapping joint position, velocities, and accelerations to the required joint torques. This task can be framed as a standard regression problem, which is solved in a supervised fashion. In contrast, in this paper, we consider the task of learning the forward dynamics, i.e. predicting the joint positions given a sequence of joint torques. The system output is therefore given by the seven joint positions ($D_y = 7$). Joint velocities and acceleration, as latent states, are not available for learning but have to be inferred. The system input is given by the seven joint torques ($D_u = 7$).

The original training dataset (44.484 datapoints) recorded at 100 Hz has been downsampled to 50 Hz. It is split into 66 independent experiments as indicated by the discontinuities in the original time-series data. Six out of 66 experiments have been utilized for testing whereas the other 60 experiments remain for training. None of the reference methods from the model learning benchmark is out-of-the-box applicable to this large scale dataset. To obtain a baseline, the sparse GP-NARX model has been trained on a subset of training experiments (400 inducing points, approx. 2000 training data points). The PR-SSM can be directly trained on the full training dataset utilizing its stochastic, minibatched optimization scheme. PR-SSM is setup similar to the configuration described in the benchmark experiment but based on a 14 dimensional latent state ($D_x = 14$). Long-term prediction results on one of the test experiments are visualized in Fig. 39. PR-SSM robustly predicts the robot arm motions for all joints and clearly improves over the GP-NARX baseline. In contrast, the GP-NARX baseline can not predict the dynamics on 5 out of 7 joints.

	One-step-ahead autoregressive		Multi-step-ahead autoregressive in latent space			Markovian State-Space Models	
	Data unnormalized + Mean prediction				Default configuration		
Task	GP-NARX	NIGP	REVARB 1	REVARB 2	MSGP	SS-GP-SSM	**PR-SSM**
Actuator	0.645 (0.018)	0.752 (0)	0.496 (0.057)	0.565 (0.047)	0.771 (0.098)	0.696 (0.034)	0.502 (0.031)
Ballbeam	0.169 (0.005)	0.165 (0)	0.138 (0.001)	0.073 (0.000)	0.124 (0.034)	411.550 (273.043)	0.073 (0.007)
Drives	0.579 (0.004)	0.378 (0)	0.718 (0.081)	0.282 (0.031)	0.451 (0.021)	0.718 (0.009)	0.492 (0.038)
Furnace	1.199 (0.001)	1.195 (0)	1.210 (0.034)	1.945 (0.016)	1.277 (0.127)	1.318 (0.027)	1.249 (0.029)
Dryer	0.278 (0.003)	0.281 (0)	0.149 (0.017)	0.128 (0.001)	0.146 (0.004)	0.152 (0.006)	0.140 (0.018)
	Data unnormalized + Moment matching				Default configuration		
Task	GP-NARX	NIGP	REVARB 1	REVARB 2	MSGP	SS-GP-SSM	**PR-SSM**
Actuator	0.633 (0.018)	0.601 (0)	0.430 (0.026)	0.618 (0.047)	0.771 (0.098)	0.696 (0.034)	0.502 (0.031)
Ballbeam	0.077 (0.000)	0.078 (0)	0.131 (0.005)	0.073 (0.000)	0.124 (0.034)	411.550 (273.043)	0.073 (0.007)
Drives	0.688 (0.003)	0.398 (0)	0.801 (0.032)	0.733 (0.087)	0.451 (0.021)	0.718 (0.009)	0.492 (0.038)
Furnace	1.198 (0.002)	1.195 (0)	1.192 (0.002)	1.947 (0.032)	1.277 (0.127)	1.318 (0.027)	1.249 (0.029)
Dryer	0.284 (0.003)	0.280 (0)	0.878 (0.016)	0.123 (0.002)	0.146 (0.004)	0.152 (0.006)	0.140 (0.018)
	Data normalization + Mean prediction				Default configuration		
Task	GP-NARX	NIGP	REVARB 1	REVARB 2	MSGP	SS-GP-SSM	**PR-SSM**
Actuator	0.665 (0.014)	0.791 (0)	0.506 (0.092)	0.559 (0.069)	0.771 (0.098)	0.696 (0.034)	0.502 (0.031)
Ballbeam	0.357 (0.199)	0.154 (0)	0.141 (0.004)	0.206 (0.008)	0.124 (0.034)	411.550 (273.043)	0.073 (0.007)
Drives	0.564 (0.029)	0.369 (0)	0.605 (0.027)	0.376 (0.026)	0.451 (0.021)	0.718 (0.009)	0.492 (0.038)
Furnace	1.201 (0.001)	1.205 (0)	1.196 (0.002)	1.189 (0.001)	1.277 (0.127)	1.318 (0.027)	1.249 (0.029)
Dryer	0.282 (0.001)	0.269 (0)	0.123 (0.001)	0.113 (0)	0.146 (0.004)	0.152 (0.006)	0.140 (0.018)
	Data normalization + Moment matching				Default configuration		
Task	GP-NARX	NIGP	REVARB 1	REVARB 2	MSGP	SS-GP-SSM	**PR-SSM**
Actuator	0.627 (0.005)	0.599 (0)	0.438 (0.049)	0.613 (0.190)	0.771 (0.098)	0.696 (0.034)	0.502 (0.031)
Ballbeam	0.284 (0.222)	0.087 (0)	0.139 (0.007)	0.209 (0.012)	0.124 (0.034)	411.550 (273.043)	0.073 (0.007)
Drives	0.701 (0.015)	0.373 (0)	0.828 (0.025)	0.868 (0.113)	0.451 (0.021)	0.718 (0.009)	0.492 (0.038)
Furnace	1.201 (0.000)	1.205 (0)	1.195 (0.002)	1.188 (0.001)	1.277 (0.127)	1.318 (0.027)	1.249 (0.029)
Dryer	0.310 (0.044)	0.268 (0)	0.851 (0.011)	0.355 (0.027)	0.146 (0.004)	0.152 (0.006)	0.140 (0.018)

Table 8: Comparison of model learning methods on five real-world benchmark examples.

Appendix: DD-OPG

In this chapter, additional material regarding the Deep Deterministic Off-Policy Gradient (DD-OPG) method and evaluation is presented. First, the proof of the weighted importance sampling estimator is presented in Sec. D.1. Details about the experimental evaluation are presented in Sec. D.2, which summarize the benchmark reference algorithms (cf. Sec. D.2.1) and test environments (cf. Sec. D.2.2).

D.1 Proof of Proposition 1

The weighted importance sampling estimator of the expected cost is given by

$$\hat{J}^{\text{WIS}}(\theta) = \frac{1}{\sum_{i=0}^{M} w(\tau_i, \theta)} \sum_{i=0}^{M} w(\tau_i, \theta) R(\tau_i)\,, \tag{137}$$

as derived in Sec. 10.2 of the main text. Talking the derivative with respect to the policy parameters, we obtain the policy gradient formulation from theorem 1 as shown in (145).

$$\nabla_\theta \hat{J}^{\text{WIS}}(\theta) = \nabla_\theta \left(\left(\sum_{i=1}^{N} \frac{p(\tau_i \mid \theta)}{\frac{1}{N}\sum_j p(\tau_i \mid \theta_j)} \right)^{-1} \right) \sum_{i=0}^{N} \frac{p(\tau_i \mid \theta)}{\frac{1}{N}\sum_j p(\tau_i \mid \theta_j)} R(\tau_i) + \tag{138}$$

$$\left(\sum_{i=1}^{N} \frac{p(\tau_i \mid \theta)}{\frac{1}{N}\sum_j p(\tau_i \mid \theta_j)} \right)^{-1} \sum_{i=0}^{N} \nabla_\theta \left(\frac{p(\tau_i \mid \theta)}{\frac{1}{N}\sum_j p(\tau_i \mid \theta_j)} R(\tau_i) \right) \tag{139}$$

$$= -\left(\sum_{i=1}^{N} w_i(\theta) \right)^{-2} \left(\sum_{i=1}^{N} \nabla_\theta w_i(\theta) \right) \left(\sum_{i=1}^{N} w_i(\theta) R(\tau_i) \right) + \tag{140}$$

$$\left(\sum_{i=1}^{N} w_i(\theta) \right)^{-1} \left(\sum_{i=1}^{N} \nabla_\theta w_i(\theta) R(\tau_i) \right) \tag{141}$$

$$= -\frac{1}{Z^2} \left(\sum_{i=1}^{N} \nabla_\theta w_i(\theta) \right) \left(\sum_{i=1}^{N} w_i(\theta) R(\tau_i) \right) + \frac{1}{Z} \left(\sum_{i=1}^{N} \nabla_\theta w_i(\theta) R(\tau_i) \right) \tag{142}$$

$$= \frac{1}{Z} \left(\sum_{i=1}^{N} \nabla_\theta w_i(\theta) R(\tau_i) - \sum_{i=1}^{N} \nabla_\theta w_i(\theta) \frac{\sum_{i=1}^{N} w_i(\theta) R(\tau_i)}{Z} \right) \tag{143}$$

$$= \frac{1}{Z} \left(\sum_{i=1}^{N} \nabla_\theta w_i(\theta) R(\tau_i) - \sum_{i=1}^{N} \nabla_\theta w_i(\theta) \hat{J}^{\text{WIS}}(\theta) \right) \tag{144}$$

$$\nabla_\theta \hat{J}^{\text{WIS}}(\theta) = \frac{1}{Z} \sum_{i=1}^{N} \nabla_\theta w_i(\theta) \left(R(\tau_i) - \hat{J}^{\text{WIS}}(\theta) \right) \tag{145}$$

Algorithm	Parameter	Range	Selected
REINFORCE	Batch size	[400, 5000]	5000
	Step size	[0.0001, 0.1]	0.03
TRPO	Batch size	[400, 5000]	5000
	Step size	[0.0001, 0.1]	0.1
PPO	Batch size	[400, 5000]	2000
	Step size	[0.0001, 0.2]	0.2

Table 9: Algorithm hyper-parameter configuration for the reference methods applied across all benchmark tasks.

Algorithm	Parameter	Symbol	Selected
DD-OPG	Temperature	λ	0.1
	Penalty	γ	0.05
	Lengthscale	$\log \Sigma$	$3\mathbf{I}$
	Path buffer	N_{max}	50

Table 10: DD-OPG default hyper-parameter configuration for the benchmark tasks.

D.2 Experimental Details

In the following section, details about the reference implementations of REINFORCE, TRPO and PPO and their parameter settings are summarized for the benchmark experiments and the ablation study. Information about the benchmark environments is given in Sec. D.2.2

D.2.1 Algorithm Configurations

The reference implementations of the benchmark algorithms REINFORCE, TRPO and PPO are from the Garage RL framework [26, 43]. A hyper-parameter grid search has been conducted for each algorithm and each environment on separate random seeds. The parameter ranges and selected hyper-parameters are indicated in Tab. 9. For the benchmark itself, ten runs have been conducted for each algorithm and each environment on the random seeds (404, 931, 159, 380, 858, 708, 16, 448, 136, 989). The configuration of the DD-OPG method is summarized in Tab. 10.

[26] contributors, *Garage: A toolkit for reproducible reinforcement learning research*, 2019

[43] Duan, Chen, Houthooft, Schulman, and Abbeel, "Benchmarking deep reinforcement learning for continuous control," 2016

D.2.2 Benchmark Environments

The benchmark environments are cartpole, mountaincar and swimmer from the Garage RL framework [26]. Details about the input and state dimensions, as well as the task horizons are listed in Tab. 11.

Environment	Inputs D_u	States D_x	Horizon H
Cartpole	1	4	100
Mountaincar	1	2	500
Swimmer	2	13	1000

Table 11: Information about the benchmark environments.

Bibliography

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. "Solving rubik's cube with a robot hand." In: *arXiv preprint arXiv:1910.07113* (2019) (cit. on p. 3).

[2] A. Alleyne and R. Liu. "A simplified approach to force control for electro-hydraulic systems." In: *Control Engineering Practice* 8.12 (2000), pp. 1347–1356 (cit. on p. 36).

[3] B. D. Anderson and J. B. Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007 (cit. on p. 11).

[4] C. W. Anderson. "Learning to control an inverted pendulum using neural networks." In: *IEEE Control Systems Magazine* 9.3 (1989), pp. 31–37 (cit. on pp. 37, 69).

[5] E. Archer, I. M. Park, L. Buesing, J. Cunningham, and L. Paninski. "Black box variational inference for state space models." In: *arXiv preprint arXiv:1511.07367* (2015) (cit. on p. 57).

[6] K. J. Åström and T. Hägglund. *PID controllers: theory, design, and tuning*. Vol. 2. Instrument society of America Research Triangle Park, NC, 1995 (cit. on p. 32).

[7] K. J. Åström and T. Hägglund. "Revisiting the Ziegler–Nichols step response method for PID control." In: *Journal of process control* 14.6 (2004), pp. 635–650 (cit. on p. 28).

[8] K. J. Åström, T. Hägglund, and K. J. Astrom. *Advanced PID control*. Vol. 461. ISA-The Instrumentation, Systems, and Automation Society Research Triangle . . ., 2006 (cit. on pp. 27, 30).

[9] K. J. Åström and R. M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010 (cit. on pp. 13, 26).

[10] K. J. Aström and R. M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010 (cit. on p. 53).

[11] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap. "Distributed distributional deterministic policy gradients." In: *arXiv preprint arXiv:1804.08617* (2018) (cit. on p. 3).

[12] J. Baxter and P. L. Bartlett. "Infinite-horizon policy-gradient estimation." In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 319–350 (cit. on pp. 88, 90).

[13] J. Bayer and C. Osendorfer. "Learning stochastic recurrent networks." In: *arXiv preprint arXiv:1411.7610* (2014) (cit. on p. 57).

[14] F. Berkenkamp, A. P. Schoellig, and A. Krause. "Safe controller optimization for quadrotors with Gaussian processes." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 491–496 (cit. on p. 51).

[15] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. "Dota 2 with large scale deep reinforcement learning." In: *arXiv preprint arXiv:1912.06680* (2019) (cit. on p. 3).

[16] J. Berner, T. Hägglund, and K. J. Åström. "Asymmetric relay autotuning - Practical features for industrial use." In: *Control Engineering Practice* 54 (Sept. 2016), pp. 231–245 (cit. on p. 30).

[17] S. A. Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013 (cit. on pp. 14, 40, 52, 55, 56).

[18] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert, and A. Knoll. "Learning Throttle Valve Control Using Policy Search." In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 49–64 (cit. on pp. 19, 67).

[19] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. "Variational inference: A review for statisticians." In: *Journal of the American Statistical Association* 112.518 (2017), pp. 859–877 (cit. on pp. 75, 77).

[20] J. B. Burl. *Linear optimal control: H (2) and H (Infinity) methods*. Addison-Wesley Longman Publishing Co., Inc., 1998 (cit. on p. 11).

[21] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer Science & Business Media, 2013 (cit. on p. 53).

[22] J. Q. Candela, A. Girard, J. Larsen, and C. E. Rasmussen. "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting." In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vol. 2. 2003, pp. II–701 (cit. on pp. 23, 25, 35).

[23] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine. "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning." In: *34th International Conference on Machine Learning (ICML)*. PMLR. 2017, pp. 703–711 (cit. on p. 87).

[24] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. "A recurrent latent variable model for sequential data." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015, pp. 2980–2988 (cit. on p. 57).

[25] P Cominos and N Munro. "PID controllers: Recent tuning methods and design to specification." In: *IET Proceedings on Control Theory and Applications* 149.1 (2002), pp. 46–53 (cit. on pp. 25, 26).

[26] T. garage contributors. *Garage: A toolkit for reproducible reinforcement learning research*. `https://github.com/rlworkgroup/garage`. 2019 (cit. on p. 134).

[27] A. C. Damianou and N. D. Lawrence. "Deep Gaussian Processes." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2013, pp. 207–215 (cit. on p. 54).

[28] C. Dann, G. Neumann, J. Peters, et al. "Policy evaluation with temporal differences: A survey and comparison." In: *Journal of Machine Learning Research* 15 (2014), pp. 809–883 (cit. on p. 4).

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II." In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197 (cit. on p. 40).

[30] M. Deisenroth and S. Mohamed. "Expectation propagation in Gaussian process dynamical systems." In: 25 (2012), pp. 2609–2617 (cit. on p. 31).

[31] M. P. Deisenroth and C. E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." In: *Proceedings of the 28th International Conference on Machine Learning (ICML)*. 2011, pp. 465–472 (cit. on pp. 5, 19, 38, 51, 53, 57, 64, 66, 67, 70, 110, 117, 119).

[32] M. P. Deisenroth and D. Fox. "Multiple-target reinforcement learning with a single policy." In: *ICML Workshop on Planning and Acting with Uncertain Models*. Citeseer. 2011 (cit. on pp. 25, 36, 37, 47, 110).

[33] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. "Gaussian processes for data-efficient learning in robotics and control." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2015), pp. 408–423 (cit. on pp. 8, 35).

[34] M. P. Deisenroth, G. Neumann, J. Peters, et al. "A survey on policy search for robotics." In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142 (cit. on pp. 10, 87, 90).

[35] M. P. Deisenroth, C. E. Rasmussen, and D. Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning." In: *Robotics: Science & Systems (RSS)*. Vol. 7. MIT Press Journals, 2011, pp. 57–64 (cit. on pp. 11, 19, 22, 24, 32, 35, 47, 67).

[36] M. P. Deisenroth, R. D. Turner, M. F. Huber, U. D. Hanebeck, and C. E. Rasmussen. "Robust filtering and smoothing with Gaussian processes." In: *IEEE Transactions on Automatic Control* 57.7 (2012), pp. 1865–1871 (cit. on p. 58).

[37] A. Der Kiureghian and O. Ditlevsen. "Aleatory or epistemic? Does it matter?" In: *Structural safety* 31.2 (2009), pp. 105–112 (cit. on p. 54).

[38] L. Desborough and R. Miller. "Increasing customer value of industrial control performance monitoring—Honeywell's experience." In: *Chemical Process Control–VI (Tuscon, Arizona* 98 (2001) (cit. on pp. 19, 109).

[39] A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, M. Toussaint, and S. Trimpe. "Optimizing Long-term Predictions for Model-based Policy Search." In: *Conference on Robot Learning (CORL)*. 2017, pp. 227–238 (cit. on pp. 5, 6, 52, 55, 82, 125, 127, 129, 149).

[40] A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. "Probabilistic Recurrent State-Space Models." In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1280–1289 (cit. on pp. 5, 6, 52, 149).

[41] A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, and S. Trimpe. "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2017 (cit. on pp. 5, 19, 53, 67, 69, 110, 149).

[42] A. Doerr, M. Volpp, M. Toussaint, S. Trimpe, and C. Daniel. "Trajectory-based off-policy deep reinforcement learning." In: *International Conference on Machine Learning (ICML)*. 2019, pp. 1636–1645 (cit. on pp. 6, 88, 149).

[43] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. "Benchmarking deep reinforcement learning for continuous control." In: *International Conference on Machine Learning (ICML)*. 2016, pp. 1329–1338 (cit. on pp. 102, 134).

[44] S. Eleftheriadis, T. Nicholson, M. Deisenroth, and J. Hensman. "Identification of Gaussian Process State Space Models." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5315–5325 (cit. on pp. 56, 58, 75, 76, 79, 82).

[45] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. "IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures." In: *arXiv preprint arXiv:1802.01561* (2018) (cit. on pp. 90, 94).

[46] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. "Model-based value estimation for efficient model-free reinforcement learning." In: *arXiv preprint arXiv:1803.00101* (2018) (cit. on p. 23).

[47] R. Föll, B. Haasdonk, M. Hanselmann, and H. Ulmer. "Deep Recurrent Gaussian Process with Variational Sparse Spectrum Approximation." In: *arXiv preprint arXiv:1711.00799* (2017) (cit. on p. 58).

[48] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. "Sequential neural models with stochastic layers." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 2199–2207 (cit. on p. 57).

[49] R. Frigola, Y. Chen, and C. E. Rasmussen. "Variational Gaussian process state-space models." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014, pp. 3680–3688 (cit. on pp. 56, 58, 74, 75, 77, 82).

[50] R. Frigola, F. Lindsten, T. B. Schön, and C. E. Rasmussen. "Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013, pp. 3156–3164 (cit. on pp. 56, 58, 63, 75, 82).

[51] R. Frigola and C. E. Rasmussen. "Integrated Pre-Processing for Bayesian Nonlinear System Identification with Gaussian Processes." In: *52nd IEEE Conference on Decision and Control (CDC)*. 2013, pp. 5371–5376 (cit. on p. 56).

[52] R. Frigola-Alcade. "Bayesian time series learning with Gaussian processes." In: *University of Cambridge* (2015) (cit. on p. 73).

[53] Z.-L. Gaing. "A particle swarm optimization approach for optimum design of PID controller in AVR system." In: *IEEE transactions on energy conversion* 19.2 (2004), pp. 384–391 (cit. on p. 29).

[54] Y. Gal, R. McAllister, and C. E. Rasmussen. "Improving PILCO with Bayesian neural network dynamics models." In: *Data-Efficient Machine Learning workshop, ICML*. Vol. 4. 2016, p. 34 (cit. on pp. 11, 22).

[55] O. Garpinger and T. Hägglund. "A software tool for robust PID design." In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 6416–6421 (cit. on p. 29).

[56] M. Gemici, C.-C. Hung, A. Santoro, G. Wayne, S. Mohamed, D. J. Rezende, D. Amos, and T. Lillicrap. "Generative Temporal Models with Memory." In: *arXiv preprint arXiv:1702.04649* (2017) (cit. on p. 57).

[57] A. Girard and C. E. Rasmussen. "Multiple-step ahead prediction for non linear dynamic systems–a gaussian process treatment with propagation of the uncertainty." In: () (cit. on pp. 23, 73).

[58] A. Girard, C. E. Rasmussen, J Quinonero-Candela, R Murray-Smith, O Winther, and J Larsen. "Multiple-step ahead prediction for non linear dynamic systems—a Gaussian process treatment with propagation of the uncertainty." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 15. 2003, pp. 529–536 (cit. on pp. 24, 54, 55, 83, 126).

[59] P. W. Goldberg, C. K. Williams, and C. M. Bishop. "Regression with input-dependent noise: A Gaussian process treatment." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 10. 1997, pp. 493–499 (cit. on p. 59).

[60] E. Greensmith, P. L. Bartlett, and J. Baxter. "Variance reduction techniques for gradient estimates in reinforcement learning." In: *Journal of Machine Learning Research* 5.Nov (2004), pp. 1471–1530 (cit. on p. 96).

[61] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. "Continuous deep q-learning with model-based acceleration." In: *International Conference on Machine Learning*. 2016, pp. 2829–2838 (cit. on pp. 22, 23).

[62] C. C. Hang, K. J. Åström, and W. K. Ho. "Refinements of the Ziegler–Nichols tuning formula." In: *IEE Proceedings D (Control Theory and Applications)*. Vol. 138. 2. IET. 1991, pp. 111–118 (cit. on p. 29).

[63] J. Hensman, N. Fusi, and N. D. Lawrence. "Gaussian processes for big data." In: *arXiv preprint arXiv:1309.6835* (2013) (cit. on p. 75).

[64] S. Hochreiter and J. Schmidhuber. "LSTM can solve hard long time lag problems." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1997, pp. 473–479 (cit. on p. 57).

[65] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. "Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?" In: *arXiv preprint arXiv:1811.02553* (2018) (cit. on p. 94).

[66] F. Jiang and Z. Gao. "An application of nonlinear PID control to a class of truck ABS problems." In: *IEEE Conference on Decision and Control (CDC)*. Vol. 1. 2001, pp. 516–521 (cit. on pp. 25, 26).

[67] T. Jie and P. Abbeel. "On a connection between importance sampling and the likelihood ratio policy gradient." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2010, pp. 1000–1008 (cit. on pp. 12, 90, 95, 96).

[68] M. A. Johnson and M. H. Moradi. *PID control - New Identification and Design Methods*. Springer, 2005 (cit. on pp. 26, 30).

[69] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. "Model-based reinforcement learning for Atari." In: *arXiv preprint arXiv:1903.00374* (2019) (cit. on p. 21).

[70] R. Kalman. "A new approach to linear filtering and prediction problems." In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45 (cit. on p. 57).

[71] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. "Deep variational bayes filters: Unsupervised learning of state space models from raw data." In: *International Conference on Learning Representations*. 2016 (cit. on p. 57).

[72] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard. "Most likely heteroscedastic Gaussian process regression." In: *Proceedings of the 24th International Conference on Machine learning (ICML)*. ACM. 2007, pp. 393–400 (cit. on p. 59).

[73] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 100).

[74] D. P. Kingma and M. Welling. "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (2013) (cit. on p. 78).

[75] J. Ko and D. Fox. "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models." In: *Autonomous Robots* 27.1 (2009), pp. 75–90 (cit. on p. 58).

[76] J. Kocijan, A. Girard, B. Banko, and R. Murray-Smith. "Dynamic systems identification with Gaussian processes." In: *Mathematical and Computer Modelling of Dynamical Systems* 11.4 (2005), pp. 411–424 (cit. on pp. 19, 22–24, 55, 67, 119, 120, 126, 127).

[77] R. G. Krishnan, U. Shalit, and D. Sontag. "Deep Kalman filters." In: *arXiv preprint arXiv:1511.05121* (2015) (cit. on p. 57).

[78] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. "Model-ensemble trust-region policy optimization." In: *arXiv preprint arXiv:1802.10592* (2018) (cit. on p. 23).

[79] S. Lange, M. Riedmiller, and A. Voigtländer. "Autonomous reinforcement learning on raw visual input data in a real world application." In: *The 2012 international joint conference on neural networks (IJCNN)*. IEEE. 2012, pp. 1–8 (cit. on p. 4).

[80] E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. "Benchmarking model-based reinforcement learning." In: *arXiv preprint arXiv:1907.02057* (2019) (cit. on pp. 11, 23, 24).

[81] M. Lefarov. "Model-Based policy Search for Learning Multivariate PID Gain Scheduling Control." MA thesis. University of Stuttgart, Apr. 2018 (cit. on pp. 20, 43, 48).

[82] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies." In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40 (cit. on p. 51).

[83] S. Levine and V. Koltun. "Guided Policy Search." In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. 2013 (cit. on pp. 11, 51).

[84] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal control*. John Wiley & Sons, 2012 (cit. on p. 13).

[85] B. Likar and J. Kocijan. "Predictive control of a gas–liquid separation plant based on a Gaussian process model." In: *Computers & chemical engineering* 31.3 (2007), pp. 142–152 (cit. on p. 55).

[86] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (2015) (cit. on pp. 89, 102).

[87] M. L. Littman and R. S. Sutton. "Predictive representations of state." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2002, pp. 1555–1561 (cit. on p. 55).

[88] L. Ljung. "System identification." In: *Signal analysis and prediction*. Springer, 1998, pp. 163–173 (cit. on pp. 13, 55).

[89] L. Ljung. "Perspectives on system identification." In: *Annual Reviews in Control* 34.1 (2010), pp. 1–12 (cit. on p. 53).

[90] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. "Automatic LQR Tuning Based on Gaussian Process Global Optimization." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2016 (cit. on pp. 12, 38, 39, 42, 69).

[91] C. L. C. Mattos, Z. Dai, A. Damianou, J. Forth, G. A. Barreto, and N. D. Lawrence. "Recurrent Gaussian processes." In: *arXiv preprint arXiv:1511.06644* (2015) (cit. on pp. 55–58, 67, 70, 75, 76, 79, 82, 83, 120, 127).

[92] C. L. C. Mattos, Z. Dai, A. Damianou, J. Forth, G. A. Barreto, and N. D. Lawrence. *REVARB implementation for RGP*. `https://github.com/zhenwendai/RGP`. [Online; accessed 30-Jan-2017]. 2015 (cit. on p. 120).

[93] C. L. C. Mattos, A. Damianou, G. A. Barreto, and N. D. Lawrence. "Latent Autoregressive Gaussian Processes Models for Robust System Identification." In: *IFAC-PapersOnLine* 49.7 (2016), pp. 1121–1126 (cit. on pp. 55, 68, 120, 121, 127).

[94] A. McHutchon and C. E. Rasmussen. "Gaussian process training with input noise." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2011, pp. 1341–1349 (cit. on pp. 55, 59, 67, 82, 119, 126).

[95] A. M. Metelli, M. Papini, F. Faccio, and M. Restelli. "Policy Optimization via Importance Sampling." In: *arXiv preprint arXiv:1809.06098* (2018) (cit. on pp. 89–91, 94, 95, 99).

[96] N. Meuleau, L. Peshkin, L. P. Kaelbling, and K.-E. Kim. "Off-policy policy search." In: *MIT Articical Intelligence Laboratory* (2000) (cit. on p. 95).

[97] N. Mishra, P. Abbeel, and I. Mordatch. "Prediction and Control with Temporal Segment Models." In: *arXiv preprint arXiv:1703.04070* (2017) (cit. on p. 57).

[98] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013) (cit. on pp. 3, 8, 12).

[99] T. M. Moerland, J. Broekens, and C. M. Jonker. "Model-based reinforcement learning: A survey." In: *arXiv preprint arXiv:2006.16712* (2020) (cit. on pp. 11, 21, 24).

[100] D. Moor. *DaISy: Database for the Identification of Systems*. `http://homes.esat.kuleuven.be/~smc/daisy/`. [Online; accessed 30-Jan-2017]. 2017 (cit. on pp. 122, 126).

[101] R. Munos. "Policy gradient in continuous time." In: *Journal of Machine Learning Research* 7.May (2006), pp. 771–791 (cit. on p. 89).

[102] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. "Safe and efficient off-policy reinforcement learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 1054–1062 (cit. on pp. 90, 94).

[103] R. Murray-Smith and A. Girard. "Gaussian Process priors with ARMA noise models." In: *Irish Signals and Systems Conference, Maynooth*. 2001, pp. 147–152 (cit. on p. 55).

[104] R. Murray-Smith, T. A. Johansen, and R. Shorten. "On transient dynamics, off-equilibrium behaviour and identification in blended multiple model structures." In: *European Control Conference (ECC)*. IEEE. 1999, pp. 3569–3574 (cit. on p. 55).

[105] R. Murray-Smith and D. Sbarbaro. "Nonlinear adaptive control using non-parametric Gaussian process prior models." In: *15th Triennial World Congress of the International Federation of Automatic Control (IFAC)* (2002) (cit. on p. 19).

[106] R. Murray-Smith, D. Sbarbaro, C. E. Rasmussen, and A. Girard. "Adaptive, cautious, predictive control with Gaussian process priors." In: *13th IFAC Symposium on System Identification*. IFAC proceedings volumes (2003), pp. 1195–1200 (cit. on pp. 4, 19, 54).

[107] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7559–7566 (cit. on p. 23).

[108] K. S. Narendra and K. Parthasarathy. "Identification and control of dynamical systems using neural networks." In: *IEEE Transactions on neural networks* 1.1 (1990), pp. 4–27 (cit. on pp. 120, 127).

[109] O. Nelles. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer Science & Business Media, 2013 (cit. on p. 14).

[110] M. Nørgaard. *Hydraulic actuator dataset*. `http://www.iau.dtu.dk/nnbook/systems.html`. [Online; accessed 30-Jan-2017]. 2000 (cit. on pp. 122, 126).

[111] A. O'Dwyer. *Handbook of PI and PID controller tuning rules*. Vol. 57. World Scientific, 2009 (cit. on p. 28).

[112] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. "Deep exploration via bootstrapped DQN." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 4026–4034 (cit. on p. 89).

[113] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia. "Learning model-based planning from scratch." In: *arXiv preprint arXiv:1707.06170* (2017) (cit. on pp. 23, 24).

[114] R. Pascanu, T. Mikolov, and Y. Bengio. "On the difficulty of training recurrent neural networks." In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. 2013, pp. 1310–1318 (cit. on p. 81).

[115] J. R. Perez and P. B. Herrero. "Extending the AMIGO PID tuning method to MIMO systems." In: *IFAC Proceedings Volumes* 45.3 (2012), pp. 211–216 (cit. on p. 28).

[116] L. Peshkin and C. R. Shelton. "Learning from Scarce Experience." In: *International Conference on Machine Learning (ICML)*. Morgan Kaufmann Publishers Inc. 2002, pp. 498–505 (cit. on p. 95).

[117] V Peterka. "Bayesian system identification." In: *Automatica* 17.1 (1981), pp. 41–53 (cit. on p. 54).

[118] J. Peters and S. Schaal. "Reinforcement learning of motor skills with policy gradients." In: *Neural networks* 21.4 (2008), pp. 682–697 (cit. on pp. 8, 87, 90).

[119] J. Peters and S. Schaal. "Natural actor-critic." In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190 (cit. on p. 100).

[120] K. B. Petersen, M. S. Pedersen, et al. "The matrix cookbook." In: *Technical University of Denmark* 7 (2008), p. 15 (cit. on p. 118).

[121] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. "Parameter space noise for exploration." In: *arXiv preprint arXiv:1706.01905* (2017) (cit. on pp. 89, 91, 99).

[122] D. Precup, R. S. Sutton, and S. P. Singh. "Eligibility Traces for Off-Policy Policy Evaluation." In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*. Citeseer. 2000, pp. 759–766 (cit. on p. 95).

[123] S. J. Qin and T. A. Badgwell. "An overview of industrial model predictive control technology." In: *AIche symposium series*. Vol. 93. 316. New York, NY: American Institute of Chemical Engineers, 1971-c2002. 1997, pp. 232–256 (cit. on p. 19).

[124] J. Quinonero-Candela, A. Girard, and C. E. Rasmussen. *Prediction at an uncertain input for Gaussian processes and relevance vector machines-application to multiple-step ahead time-series forecasting*. Tech. rep. Danish Technical University, 2002 (cit. on pp. 54, 66, 121, 122).

[125] J. Quinonero-Candela, C. E. Rasmussen, and C. K. Williams. "Approximation methods for Gaussian process regression." In: *Large-scale kernel machines* (2007), pp. 203–224 (cit. on p. 63).

[126] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, et al. "Imagination-augmented agents for deep reinforcement learning." In: 30 (2017), pp. 5690–5701 (cit. on pp. 23, 24).

[127] D. Reeb, A. Doerr, S. Gerwinn, and B. Rakitsch. "Learning Gaussian Processes by Minimizing PAC-Bayesian Generalization Bounds." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018 (cit. on p. 149).

[128] L. Righetti, M. Kalakrishnan, P. Pastor, J. Binney, J. Kelly, R. C. Voorhies, G. S. Sukhatme, and S. Schaal. "An autonomous manipulation system based on force control and optimization." In: *Autonomous Robots* 36.1-2 (2014), pp. 11–30 (cit. on p. 38).

[129] D. E. Rivera, M. Morari, and S. Skogestad. "Internal model control: PID controller design." In: *Industrial & engineering chemistry process design and development* 25.1 (1986), pp. 252–265 (cit. on p. 29).

[130] M. R. Rudary and S. P. Singh. "A nonlinear predictive state representation." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2004, pp. 855–862 (cit. on p. 55).

[131] H. Salimbeni and M. Deisenroth. "Doubly Stochastic Variational Inference for Deep Gaussian Processes." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 4591–4602 (cit. on p. 75).

[132] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay." In: *arXiv preprint arXiv:1511.05952* (2015) (cit. on pp. 95, 100).

[133] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. "Trust region policy optimization." In: *International Conference on Machine Learning (ICML)*. 2015, pp. 1889–1897 (cit. on pp. 88, 91, 99, 102).

[134] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on pp. 88–91, 98, 102).

[135] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. "Policy gradients with parameter-based exploration for control." In: *International Conference on Artificial Neural Networks*. Springer. 2008, pp. 387–396 (cit. on p. 91).

[136] C. R. Shelton. "Policy improvement for POMDPs using normalized importance sampling." In: *Proceedings of the 17th conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann Publishers Inc. 2001, pp. 496–503 (cit. on pp. 90, 95).

[137] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010 (cit. on pp. 25, 26).

[138] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587 (2016), pp. 484–489 (cit. on p. 3).

[139] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. "Deterministic policy gradient algorithms." In: *ICML*. 2014 (cit. on p. 89).

[140] S. P. Singh, M. L. Littman, N. K. Jong, D. Pardoe, and P. Stone. "Learning predictive state representations." In: *Proceedings of the 20th International Conference on Machine Learning (ICML)*. 2003, pp. 712–719 (cit. on p. 55).

[141] E. Snelson and Z. Ghahramani. "Sparse Gaussian processes using pseudo-inputs." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2005, pp. 1257–1264 (cit. on p. 39).

[142] E. Snelson and Z. Ghahramani. "Sparse Gaussian processes using pseudo-inputs." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 18. 2006, p. 1257 (cit. on pp. 16, 63, 82, 126).

[143] J. Snoek, H. Larochelle, and R. P. Adams. "Practical bayesian optimization of machine learning algorithms." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012, pp. 2951–2959 (cit. on p. 105).

[144] I. Sutskever, J. Martens, and G. E. Hinton. "Generating text with recurrent neural networks." In: *Proceedings of the 28th International Conference on Machine Learning (ICML)*. 2011, pp. 1017–1024 (cit. on p. 57).

[145] R. S. Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming." In: *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224 (cit. on p. 23).

[146] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge, 1998 (cit. on pp. 7, 37).

[147] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 21).

[148] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. "Policy gradient methods for reinforcement learning with function approximation." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2000, pp. 1057–1063 (cit. on p. 87).

[149] A. Svensson and T. B. Schön. "A flexible state–space model for learning nonlinear dynamical systems." In: *Automatica* 80 (2017), pp. 189–199 (cit. on pp. 75, 82, 127).

[150] I. Szita and A. Lörincz. "Learning Tetris using the noisy cross-entropy method." In: *Neural computation* 18.12 (2006), pp. 2936–2941 (cit. on p. 12).

[151] G. Tesauro. "Temporal difference learning and TD-Gammon." In: *Communications of the ACM* 38.3 (1995), pp. 58–68 (cit. on p. 3).

[152] M. K. Titsias. "Variational Learning of Inducing Variables in Sparse Gaussian Processes." In: *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 5. 2009, pp. 567–574 (cit. on pp. 75, 77, 120, 127).

[153] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30 (cit. on p. 4).

[154] S. Trimpe and R. D'Andrea. "The Balancing Cube: A Dynamic Sculpture As Test Bed for Distributed Estimation and Control." In: *IEEE Control Systems* 32.6 (Dec. 2012), pp. 48–75 (cit. on p. 39).

[155] R. Turner, M. Deisenroth, and C. Rasmussen. "State-space inference and learning with Gaussian processes." In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2010, pp. 868–875 (cit. on pp. 56, 58, 63, 82).

[156] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double q-learning." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016 (cit. on p. 12).

[157] P. Van Overschee and B. De Moor. *Subspace identification for linear systems: Theory - Implementation - Applications*. Springer Science & Business Media, 2012 (cit. on pp. 55, 57).

[158] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. "Translating videos to natural language using deep recurrent neural networks." In: *arXiv preprint arXiv:1412.4729* (2014) (cit. on p. 57).

[159] S. Vijayakumar and S. Schaal. "LWPR: An O(n) algorithm for incremental real time learning in high dimensional space." In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*. 2000 (cit. on p. 130).

[160] J. Vinogradska, B. Bischoff, J. Achterhold, T. Koller, and J. Peters. "Numerical quadrature for probabilistic policy search." In: *IEEE transactions on pattern analysis and machine intelligence* 42.1 (2018), pp. 164–175 (cit. on p. 23).

[161] M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel. "Meta-Learning Acquisition Functions for Transfer Learning in Bayesian Optimization." In: *International Conference on Learning Representations*. 2019 (cit. on p. 149).

[162] J. Wang, A. Hertzmann, and D. J. Fleet. "Gaussian process dynamical models." In: 18 (2005), pp. 1441–1448 (cit. on p. 23).

[163] J. M. Wang, D. J. Fleet, and A. Hertzmann. "Gaussian process dynamical models for human motion." In: *IEEE transactions on pattern analysis and machine intelligence* 30.2 (2008), pp. 283–298 (cit. on p. 58).

[164] C. J. Watkins and P. Dayan. "Q-learning." In: *Machine learning* 8.3-4 (1992), pp. 279–292 (cit. on p. 12).

[165] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. "Embed to control: A locally linear latent dynamics model for control from raw images." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015, pp. 2746–2754 (cit. on p. 57).

[166] P. Wawrzynski and A. Pacut. "Truncated importance sampling for reinforcement learning with experience replay." In: *Proc. CSIT Int. Multiconf* (2007), pp. 305–315 (cit. on p. 90).

[167] T. Wigren. *Input-Output Data Sets for Development and Benchmarking in Nonlinear Identification*. Tech. rep. Department of Information Technology, Uppsala University, 2010 (cit. on pp. 122, 126).

[168] C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*. MIT Press, 2005 (cit. on pp. 15, 16, 24, 54, 63, 65, 130).

[169] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Reinforcement Learning*. Springer, 1992, pp. 5–32 (cit. on pp. 12, 88, 90, 91, 93, 102).

[170] A. Wilson, A. Fern, and P. Tadepalli. "Using trajectory data to improve Bayesian optimization for reinforcement learning." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 253–282 (cit. on pp. 12, 91).

[171] A. Wischnewski. "Control of highly automated and autonomous vehicles in critical driving situations." MA thesis. University of Stuttgart, 2017 (cit. on p. 43).

[172] T Yamamoto and S. Shah. "Design and experimental evaluation of a multivariable self-tuning PID controller." In: *IET Proceedings on Control Theory and Applications* 151.5 (2004), pp. 645–652 (cit. on p. 30).

[173] T. Zhao, H. Hachiya, V. Tangkaratt, J. Morimoto, and M. Sugiyama. "Efficient sample reuse in policy gradients with parameter-based exploration." In: *Neural computation* 25.6 (2013), pp. 1512–1547 (cit. on pp. 90, 94, 99).

[174] J. G. Ziegler, N. B. Nichols, et al. "Optimum settings for automatic controllers." In: *trans. ASME* 64.11 (1942) (cit. on p. 28).

Publications

This work is based on results, which have been previously published in the following papers. An outline of these contributions and the individual parts of the thesis is given in Sec. 1.

A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, and S. Trimpe. "Model-based Policy Search for Automatic Tuning of Multivariate PID Controllers." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2017

A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, M. Toussaint, and S. Trimpe. "Optimizing Long-term Predictions for Model-based Policy Search." In: *Conference on Robot Learning (CORL)*. 2017, pp. 227–238

A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. "Probabilistic Recurrent State-Space Models." In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1280–1289

A. Doerr, M. Volpp, M. Toussaint, S. Trimpe, and C. Daniel. "Trajectory-based off-policy deep reinforcement learning." In: *International Conference on Machine Learning (ICML)*. 2019, pp. 1636–1645

I contributed to the following publications during my Ph.D, which are not part of this thesis.

D. Reeb, A. Doerr, S. Gerwinn, and B. Rakitsch. "Learning Gaussian Processes by Minimizing PAC-Bayesian Generalization Bounds." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018

M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel. "Meta-Learning Acquisition Functions for Transfer Learning in Bayesian Optimization." In: *International Conference on Learning Representations*. 2019

Dissertation

Models for Data-Efficient Reinforcement Learning on Real-World Applications

Andreas Doerr

Large-scale deep *Reinforcement Learning* is strongly contributing to many recently published success stories of *Artificial Intelligence*. These techniques enabled computer systems to autonomously learn and master challenging problems, such as playing the game of *Go*, on human levels or above. Naturally, the question arises which problems could be addressed with these Reinforcement Learning technologies in *industrial applications*.

So far, machine learning technologies based on (semi-) supervised learning create the most visible impact in industrial applications. For example, image, video or text understanding are primarily dominated by models trained and derived autonomously from large-scale data sets with modern (deep) machine learning methods. Reinforcement Learning, on the opposite side, however, deals with temporal decision-making problems and is much less commonly found in the industrial context. In these problems, current decisions and actions inevitably influence the outcome and success of a process much further down the road.

This work strives to address some of the core problems, which prevent the effective use of Reinforcement Learning in industrial settings. The methods and solutions proposed in this work are grounded in the challenges experienced when operating with real-world hardware systems. With applications on a humanoid upper-body robot or an autonomous model race car, the proposed methods are demonstrated to successfully model and master their complex behavior.

University of Stuttgart
Germany

BOSCH

MAX PLANCK INSTITUTE
FOR INTELLIGENT SYSTEMS