Master's Thesis

Incorporating Uncertainty into Reinforcement Learning through Gaussian Processes

Markus Kaiser

Chair for Foundations of Software Reliability and Theoretical Computer Science
Department of Informatics
Technische Universität München

Computer Vision and Active Perception Lab
School of Computer Science and Communication
Kungliga Tekniska högskolan

Learning Systems
CT-RDA-BAM-LSY-DE
Siemens

SIEMENS

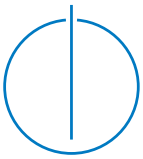
DEPARTMENT OF INFORMATICS
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis

Incorporating Uncertainty into Reinforcement Learning through Gaussian Processes

Integration von Unsicherheit in das Reinforcement Learning mit Hilfe von Gaußprozessen

Author:	Markus Kaiser
Supervisor:	Prof. Dr.-Ing. Thomas A. Runkler
Advisors:	Dr. Clemens Otte
	Prof. Dr. Carl Henrik Ek
Date:	15. June 2016



Statement

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15. June 2016

Abstract

This thesis deals with the construction of optimized flight paths for tracking shots using multicopters. The theory chapter defines appropriate metrics which are then used in an optimization problem to create spline trajectories. This solution is implemented in a software suite capable of simulating flights. Finally, after creating an interface to UAVs by Asctec, the paths are tested in an exemplary use case.

Sammanfattning

This thesis deals with the construction of optimized flight paths for tracking shots using multicopters. The theory chapter defines appropriate metrics which are then used in an optimization problem to create spline trajectories. This solution is implemented in a software suite capable of simulating flights. Finally, after creating an interface to UAVs by Asctec, the paths are tested in an exemplary use case.

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Konstruktion von optimierten Flugpfaden für Kamerafahrten mit Multikoptern. Im Theorieteil werden geeignete Metriken definiert, mittels derer in einem Optimierungsverfahren Splines als Trajektorien erzeugt werden. Diese Lösung wird in eine Software überführt und um eine Flugsimulation erweitert. Schließlich werden Schnittstellen zu Flugsystemen der Firma Asctec geschaffen und die Flugbahnen in einem Anwendungsfall getestet.

Contents

Todo list

Chapter 1

Introduction

Chapter 2

The Bicycle Benchmark

The bicycle benchmark is an example of a dynamic a computer should learn to control. It was originally defined by Randløv and Alstrøm in 1998 [RA98]. The task in this benchmark is to balance and navigate a simulated bicycle which travels at a constant speed.

The computer, or *agent*, takes the place of the rider of the bicycle. After fixed time intervals, it has to decide how to influence the bicycle by applying some torque T on the handle bars to steer or by displacing the center of mass of the bicycle via leaning over the bicycle by some distance d or both. In order to make its decision, the controller is given perfect information about the internal state of the simulation. Besides having to prevent the bicycle from falling over in the short-term, there is also the long-term goal of navigating to some predefined goal position.

The bicycle is modelled by non-linear differential equations which describe the steering and leaning behaviour of the bicycle. The speed of the back tyre is considered constant and independent of the actions of the agent. This results in two important angles to describe the system. The angle θ is measured between the front tyre and the frame of the bicycle and describes the straightness of the bicycles path. The angle ω measures the amount of tilt of the bicycle's frame compared to standing upright. If the absolute value of ω is greater than 12 degrees, the bicycle has fallen over. In addition to the two angles, their time-derivatives $\dot{\theta}$ and $\dot{\omega}$ define the dynamics of the bicycle. Together with the bicycle's position and rotation in euclidean space, this completely describes the current state of one instance of the bicycle benchmark and is summarized in table 2.1.

The conservation of angular momentum of the tyres results in interactions between θ and ω and their derivatives. The equations presented in the following describe the dynamics of the system as introduced in [RA98]. Two simplifying assumptions have been made: Firstly, the front fork is assumed to be vertical, which is unusual and makes balancing the bicycle more difficult but is not physically impossible. And secondly,

the equations are not an exact analytical description, as some second and higher order terms have been ignored.

An overview of the geometric interpretations of the state variables can be seen in figure 2.1. The interactions between tilt and lean are based on the conservation of angular momentum which is heavily influenced by the moments of inertia of the system. The moments of the tyre as displayed in figure 2.1c and the moment of the bicycle and cyclist combined I_{BC} were estimated by the original definition [RA98] as

$$\begin{aligned} I_{dc} &:= \quad M_d r^2 \\ I_{dv} &:= \frac{3}{2} M_d r^2 \\ I_{dl} &:= \frac{1}{2} M_d r^2 \\ I_{\mathrm{BC}} &:= \frac{13}{3} M_c h^2 + M_p (h + d_{\mathrm{CM}})^2. \end{aligned}$$

The dynamics also depend on multiple constants which are detailed in table 2.3.

The angular acceleration $\ddot{\omega}$ of the lean of the bicycle consists of three parts. The first part describes the gravitation acting on the bicycle and cyclist which pulls the bicycle in the direction it is already leaning. The second one are effects based on the conservation of angular momentum introduced via a cross-term dependent on $\dot{\theta}$. The centrifugal force applied because of the curved movement of the bicycle forms the last part. The center of mass can be displaced horizontally by the agent via the choice of d. The combination φ of this displacement and the lean angle of the bicycle is defined as

interpretation?

$$\varphi := \omega + \arctan\left(\frac{d}{h}\right).$$

With this, the angular acceleration $\ddot{\omega}$ can be calculated as

$$\ddot{\omega} := \frac{1}{I_{\mathrm{BC}}} \left(\sin\varphi \cdot Mgh - \cos\varphi \cdot \left(I_{dc}\dot{\sigma} \cdot \dot{\theta} + \mathrm{sgn}(\theta) \cdot v^2 \left(\frac{M_d r}{r_f} + \frac{M_d r}{r_b} + \frac{Mh}{r_{\mathrm{CM}}} \right) \right) \right).$$

The angular acceleration $\ddot{\theta}$ of the orientation of the front tyre is equal to that of the handlebars because the front fork is assumed to vertical. It is dependent on the torque T applied to the handlebars and the conservation of angular momentum introduced via a cross-term dependent on $\dot{\omega}$ and is calculated as

$$\ddot{\theta} := \frac{T - I_{dv}\dot{\sigma} \cdot \dot{\omega}}{I_{dl}}.$$

Table 2.1: Variables defining the current state of the bicycle system.

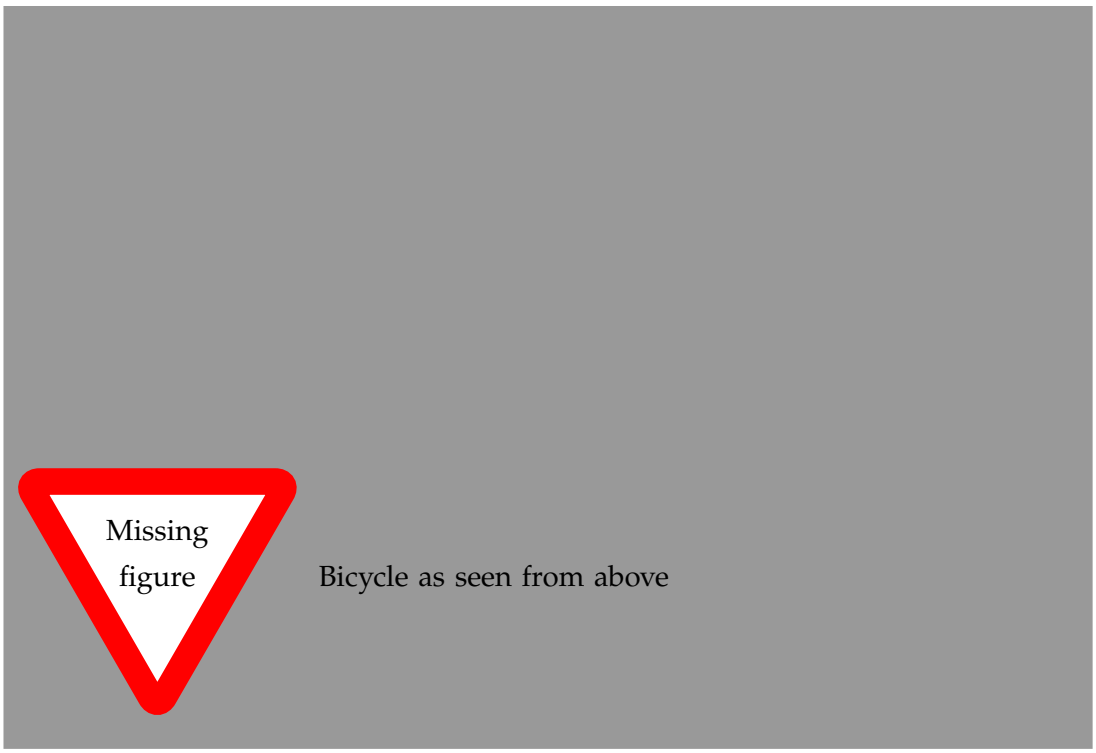
Notation	Description	Value range	Unit
θ	Angle between the frame and the front tyre	$-\pi/2$ to $\pi/2$	rad
$\dot{\theta}$	Rotational speed of the handlebars	-10 to 10	rad/s
ω	Tilt of the bicycle	$-\pi/15$ to $\pi/15$	rad
$\dot{\omega}$	Tilting speed of the bicycle	-10 to 10	rad/s
x	Global x-position of the front tyre	-100 to 100	m
y	Global y-position of the front tyre	-100 to 100	m
ψ	Global orientation of the bicycle	$-\pi$ to π	rad

Table 2.2: Actions which can be applied to the bicycle system.

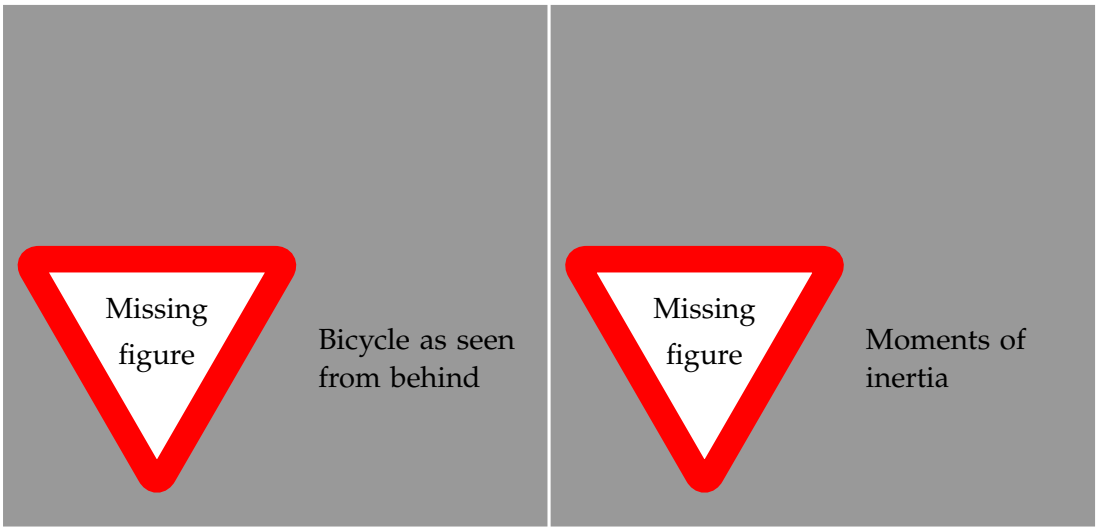
Notation	Description	Value range	Unit
d	The distance the agent leans sideways by displacing the center of mass	-0.02 to 0.02	m
T	The torque the agent applies on the handlebars	-2 to 2	N

Table 2.3: Physical constants and their values in the bicycle system [RA98].

Notation	Description	Value
CM	Center of mass of the bicycle and cyclist in total	
c	Horizontal distance between the point where the front tyre touches the ground and the saddle	66 cm
d_{CM}	Vertical distance between the centers of mass of the bicycle and the cyclist	30 cm
h	Vertical distance between the CM and the ground	94 cm
l	Distance between the points where the front and back tyres touch the ground	111 cm
M_c	Mass of the bicycle	15 kg
M_d	Mass of a tyre	1.7 kg
M_p	Mass of the cyclist	60 kg
r	Radius of a tyre	34 cm
v	Velocity of the bicycle	10 km/h
$\dot{\sigma}$	Angular velocity of the back tyre	$\dot{\sigma} = v/r$



(a) Bicycle as seen from above



(b) Bicycle as seen from behind.

(c) Moments of inertia.

Figure 2.1: Bicycle dynamics

These differential equations describe how leaning left or right as the driver and turning the handlebars interact with each other. These internal dynamics are the important factors when trying to balance the bicycle while ignoring the movement of the bicycle in space. In order to successfully navigate to the goal position, these movements have to be taken into consideration. The bicycle state contains three variables which locate the bicycle in space. The position of the bicycle is defined by the point where the front tyre touches the ground. This point is independent of the orientation of the handlebars given by the angle θ and identified via the two euclidean coordinates x and y. The last state variable ψ describes the orientation of the bicycle frame relative to this point as shown in figure 2.1a. If ψ is equal to zero, the back tyre points in x-direction if viewed from the point (x, y) and a positive ψ denotes a counter-clockwise rotation.

ψ rotation direction correct?

The back tyre of the bicycle moves at the constant speed v. The front and back tyre follow two circular paths with different radii but the same center as can be seen in figure 2.1a with the front tyre following the longer path. The radii r_f and r_b can be calculated as

$$r_f = \frac{l}{|\sin\theta|},$$
$$r_b = \frac{l}{|\tan\theta|},$$

respectively for θ not equal to zero. The singularity of θ approaching zero yields radii of arbitrary size as the bicycle's path becomes more and more straight. If θ is equal to zero, the bicycle's orientation does not change and it moves along a straight line. If it is not zero, the change of position and orientation can be calculated from the curved movement.

Since the two tyres are connected with a rigid frame and the front tyre travels on a longer path, it has to do so at a higher speed. They do however share the same angular velocity $v_\circ$ on their respective circular paths, since if θ remains constant, the bicycle rotates around the common center point. This angular velocity can be derived from the constant speed of the back tyre and is given by

$$v_\circ = \frac{v}{r_b}.$$

Combined with the direction of rotation on the circle defined by the sign of θ, this directly yields the derivative of the world orientation ψ:

$$\dot{\psi} = \operatorname{sgn}(\theta) \cdot v_\circ$$

The actual speed of the front tyre can be obtained from the common angular velocity $v_\circ$ and the radius of its path r_f. Together with the orientation of the front tyre, this gives the derivatives

$$\dot{x} = v_\circ \cdot r_f \cdot \cos(\psi + \theta)$$
$$\dot{y} = v_\circ \cdot r_f \cdot \sin(\psi + \theta)$$

of the position of the bicycle.

The original implementation of Randløv and Alstrøm uses an explicit Euler scheme to evolve the dynamics of the system for a time step. The changes of position and orientation are calculated using the exact analytical solution of moving the tyres along their circular paths. To improve accuracy, the implementation developed for this thesis implements a standard Runge-Kutta-Scheme as described in *Numerical Recipes* [Pre07]. The assumption that for a single time step, the bicycle moves along a fixed circular path is no longer correct when evaluating the dynamics with Runge-Kutta. Because of this, the changes in position and orientation are integrated into the scheme.

The goal of the bicycle benchmark is to drive the bicycle to a specific position which is assumed to be a circle at the origin of the coordinate system with a radius of 5. The bicycle starts in a position which is almost upright, with the state variables θ, $\dot{\theta}$, ω, $\dot{\omega}$ being sampled from Gaussian noise with a standard deviation of one percent of their value range.

The agent has to choose the two continuous actions d and T described in table 2.2 every 0.01 seconds, after which they are assumed constant for this time interval. After this time, the controller is once again presented with the current and exact values of the state variables and has to make a new decision. One run of the bicycle system creates a time series of bicycle states and the actions chosen by the controller. The episode ends in failure if the bicycle falls over and it ends in success if the bicycle reaches the goal. Since it is possible for the bicycle to drive in circles indefinitely, the episode also ends in failure after a certain amount of time has passed.

The next chapter introduces reinforcement learning as a general mathematical description of the problem posed with the bicycle benchmark. Model based reinforcement learning is one way to solve the bicycle benchmark which tries to learn the bicycle's dynamics using general function approximators and use them to make informed guesses about the future.

Chapter 3

Theoretical Background

The bicycle benchmark is a problem in a branch of machine learning called reinforcement learning. This chapter gives a brief introduction into reinforcement learning and defines the notation necessary to reason about the bicycle benchmark and similar problems in a mathematical way. The approach used in this thesis to solve it is called model-based reinforcement learning and is discussed next.

The models in model-based reinforcement learning are used to learn the dynamics of the system to be controlled in order to be able to make predictions about its future development. Gaussian Processes provide a framework to learn such dynamics in a way which adds information about the uncertainty of a prediction. This is achieved by learning a distribution over possible models rather than deciding on one single model. The uncertainty about the development in a single time step can be used to reason about the uncertainty of predictions multiple steps into the future.

Being able to make judgments about the future dependent on the next action to take allows the definition of a controller. Particle swarm optimization is introduced as a way to choose the best action to take based on a performance-measure of the predicted future states.

3.1 Reinforcement Learning

Consider an agent who wants to learn how to learn how to drive a bicycle to a certain position. In the case of a supervised learning environment, an expert who already knows how to solve the problem could tell this agent which actions lead to success. In the absence of such a teacher however, the agent must learn from interaction with the bicycle.

An agent might start by applying random actions to the system and quickly fall down, making it impossible to ever reach the goal. This gives the agent an opportunity to learn: It has to avoid falling down in order to have the chance of achieving its objective. A human agent who falls down from a bicycle feels pain when hitting the asphalt which they will try to avoid. Such feedback is called a *reward* (or in this case, a punishment) and is the basis on which the agent can learn to judge the viability of actions in a certain state.

After multiple trials, the agent might be able to avoid falling and be able to stabilize the bicycle. To achieve this, it may have built a basic understanding of how the bicycle system behaves. It might have recognized that a bicycle which is already leaning on one side if left alone will fall down because of the gravitational pull or that driving in a curve means that the centrifugal force pushes the bicycle to the outside. Note that to gain these insights, it is not necessary for the agent to have an understanding of the underlying physics. It is enough to observe situations in which the effects play out and generalize from there.

When the agent has learnt how to stabilize the bicycle by driving small corrective curves, it has not yet solved the original task of navigating the bicycle to a specific position. It will have to shift its focus from the short-term goal of avoiding falling over to the more high-level and long-term task of following a targeted trajectory. Following this trajectory might require some compromises concerning the previous step of avoiding to fall down, since it is easier to drive straight than it is to drive along a specific curve. In the case that reaching the goal is time-critical, the optimal trajectory would be constrained by the minimum radius of a curve the agent can handle. In the case of a perfect controller, it is constrained by the maximum lean angle ω.

The requirements an agent faces when solving the bicycle benchmark can be understood in a more general sense, which is formulated in the problem statement of reinforcement learning. It formalizes the ideas of and agent, it's interaction with a system and the positive or negative feedback it receives.

3.1.1 Problem Statement

Reinforcement learning is meant to describe the general problem of learning to control a system by interaction in order to achieve a predefined goal. In order to achieve this goal, the learning entity or *agent* has to decide on specific actions to influence its *environment*, which is everything outside of the agent. The boundary between agent and environment is a well-defined and narrow interface illustrated in figure 3.1. They

interact via this interface at specific discrete *time steps*, usually indexed with the natural numbers $\mathbb{N}$.

Figure 3.1: Agent-Environment-Interaction

At every such time step t, the agent can observe the environment and receives some information in form of its *state* $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the continuous space of all possible states. Based on this information, the agent has to decide on which action $a_t \in \mathcal{A}$ to perform. The space of all possible actions $\mathcal{A}$ is assumed to be continuous and remains constant for all time steps and states.

The decision-process an agent employs in order to choose an action is called the agent's *policy*. A policy is a function which maps states to actions. Policies are often encoded in closed forms such as linear functions [Dei10].

Definition 1 (Policy)
A *policy* π an agent follows encodes the choice it makes when faced with a decision. It is a function

$$\pi : \mathcal{S} \to \mathcal{A}$$

which maps the current state of the system to the action the agent will perform.

Once the agent has chosen an action for time step t, the *transition dynamics* of the system determine the state s_{t+1}. These dynamics are unknown to the agent and can be subject to probabilistic factors such as noise. While the agent will always observe one single element $s_{t+1} \in \mathcal{S}$ when prompted for the next decision, the transition dynamics return a probability distribution over the possible next states.

Many problems like the bicycle benchmark have a natural notion of states which are *terminal*. If the cyclist falls down or reaches the goal, the interaction between the agent and the environment comes to an end and there are no more decisions to make. Such tasks are called *episodic* and one sequence from a start state to a terminal state is called an *episode*. Notationally, $\mathcal{T}$ is defined to be the set of all terminal states with $\mathcal{T} \cap \mathcal{S} = \varnothing$. An episode ends when the transition dynamics return a state which is in $\mathcal{T}$. The set $\mathcal{S} \cup \mathcal{T}$ is called the *extended state space* $\mathcal{S}^+$ and combines both non-terminal and terminal states. A task which does not have terminal states is called *continuous* and has episodes of infinite length.

Definition 2 (Transition Dynamics)
The *transition dynamics* $\mathbf{f}$ of a system encode its physical behaviour. These dynamics

$$\mathbf{f} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+)$$

mapping a state and an action to a distribution over possibly terminal states stay constant over time but can be probabilistic in nature.

An important observation following from this definition is that the transition dynamics fulfill the Markov property [SB98]. This means that the distribution of the state s_{t+1} is independent of all states before s_t given s_t. In other words, the transition dynamics only depend on the present state of the system and are conditionally independent of the past.

At time step $t+1$, the agent observes the environment again in the form of the state s_{t+1}. Additionally, it also receives immediate feedback about this new state called the *reward* r_{t+1}. The reward is a quality assigned to each state and serves as a measure of how good a state to be in it is. This measure is independent of the future or past development of the system. It is obtained from a real-valued *reward function* $\mathbf{R}$ such that $r_{t+1} = \mathbf{R}(s_{t+1})$.

Definition 3 (Reward Function)
The *reward function* $\mathbf{R}$ assigns a quality to each state in the extended state space and has the signature

$$\mathbf{R} : \mathcal{S}^+ \to \mathbb{R}.$$

This reward is the immediate feedback an agent receives when interacting with the system.

The goal of the agent is to maximize the sum of all rewards earned throughout an episode. A greedy agent which is only concerned with the next immediate reward might not be the most successful, since it may be necessary to make a decision which is bad in the short term to gain an advantage in the long run, such as sacrificing a piece in chess to end up in a better position overall.

The *value function* is a measure for how good a policy behaves in the long run. Given a policy and a start state, it is defined as the expected sum of rewards earned in a time horizon T. Since the transition dynamics are assumed to be probabilistic, the states at all time steps greater than zero are random variables. Given a distribution of the state $\boldsymbol{s_t}$, the distribution for the next state $\boldsymbol{s_{t+1}}$ for all $t \in \mathbb{N}_{>0}$ can be calculated as

$$\mathrm{p}(\boldsymbol{s_{t+1}}) = \int \mathrm{p}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi)\, \mathrm{p}(\boldsymbol{s_t})\, \mathrm{d}\boldsymbol{s_t}$$

where

$$\mathrm{p}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi) = \begin{cases} \delta_{\boldsymbol{s_t}, \boldsymbol{s_{t+1}}} & \text{if } \boldsymbol{s_t} \in \mathcal{T} \\ \mathbf{f}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi(\boldsymbol{s_t})) & \text{otherwise.} \end{cases}$$

Here, δ denotes the Dirac-delta-function and the notation $\mathbf{f}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi(\boldsymbol{s_t}))$ means the probability of $\boldsymbol{s_{t+1}}$ under the distribution $\mathbf{f}(\boldsymbol{s_t}, \pi(\boldsymbol{s_t}))$. Once the dynamics have reached a terminal state, all future states will be this same state. This extends all episodes to potentially infinite length.

The time horizon T can be chosen freely but does not depend on either the current policy or state. The larger the time horizon, the more far-sighted a agent has to be to be successful. For large values of T, it can be helpful to focus on rewards earned in the near future and to weigh potential rewards further along with a smaller factor. This can be achieved using the discount factor γ which is a constant real number between zero and one. In the case of an infinite time horizon, γ must be chosen smaller than one to ensure the well-definedness of the function.

Definition 4 (Value Function)
Given transition dynamics $\mathbf{f}$, a policy π, a time horizon $T \in \mathbb{N} \cup \{\infty\}$ and a discount factor $0 \leq \gamma \leq 1$, the *value function* $\mathbf{V}^\pi$ denotes the expected long-term reward of a state and is given by

$$\mathbf{V}^\pi : \begin{cases} \mathcal{S}^+ \to \mathbb{R} \\ \quad s \mapsto \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \mathbf{R}(s_t) \,\middle|\, \mathbf{f}, \pi, s_0 = s \right]. \end{cases}$$

If the time horizon is infinite, γ must be smaller than 1.

Assume now a known distribution of possible start states $p(s_0)$. The sets $\mathcal{S}^+$ and $\mathcal{A}$ of states and actions together with the transition dynamics $\mathbf{f}$, the reward function $\mathbf{R}$ and this distribution $p(s_0)$ defines a fully observable Markov decision process (MDP) [SB98; Mur12].

The objective of the reinforcement learning problem stated here is to find the most successful policy to control this decision process under the assumption that no expert knowledge about the transition dynamics is available. The optimal policy π^* maximizes the expected value for all start states, that is it solves the optimization problem

$$\begin{aligned} \pi^* &\in \operatorname*{argmax}_\pi \mathbb{E}_{s_0}[V^\pi(s_0)] \\ &= \operatorname*{argmax}_\pi \int V^\pi(s_0)\, p(s_0)\, \mathrm{d}s_0. \end{aligned} \tag{3.1}$$

There is a multitude of different approaches of arriving at good policies. The following will introduce the important distinction of model-based and model-free methods and give a high level view of how the former will be used within this thesis.

3.1.2 Model-Based Reinforcement Learning

To find a good policy, an agent has to gain experience about its environment via interaction. In *direct reinforcement learning* [SB98], or model-free methods, this experience is used to update the current policy or an estimation of its value function directly. After enough time spent with the system, iteratively improving the current policy can converge to the optimal policy.

In *model-based* reinforcement learning, the experience is used to learn an internal representation of the transition dynamics $\mathbf{f}$. This allows the agent to make predictions about the future behaviour of the system and therefore approximatively evaluate the value function without actually interacting with the system. A closed form policy can then be found by solving the non-linear optimization problem proposed in equation (3.1).

Deisenroth [Dei10] describes an algorithmic scheme called PILCO in which phases of exploration on the real system to gather more data to improve the internal model alternate with phases of improvement of the current policy based on the internal simulation. This iteration allows the agent to explore promising directions in the state space using intermediate policies.

In this thesis it is assumed that all interaction with the system has to happen before any learning can take place. This assumption is sensible in the context of industrial applications where interactions with a system can be very expensive and dangerous and where a potentially bad agent cannot be allowed to choose actions to perform. Instead of allowing interaction, the agent is presented with a data set of observations of the transition dynamics in the form of tuples $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}})$. These observations can be obtained via a mix of random exploration and actions chosen by a sub-optimal controller.

Using an internal representation instead of the real transition dynamics to choose actions leads to one of the major drawbacks of model-based reinforcement learning. If this representation does not capture the important characteristics of the original system well, a policy found to be optimal on the simulation might not lead to good results on the correct dynamics. This effect is called the *model bias* of a policy.

Reducing this model bias is the main goal of explicitly representing uncertainties within this thesis. Instead of focusing on one single dynamics model, a probabilistic representation of all plausible models to explain the observed data will be considered. This yields a measure of uncertainty of predictions one step into the future and can be extended to long-term predictions as required in the value function. Assuming deterministic transition dynamics such as the bicycle benchmark, this measure does not describe a property of the system but rather the uncertainty about the model itself.

Gaussian Processes provide a framework to represent such a distribution over possible models for the transition dynamics. The following section will introduce their definition and derive how they can be used to solve regression problems. The last part of this chapter will be concerned with introducing a policy-representation based on these results which does not depend on a closed form solution.

3.2 Gaussian Process Regression

The transition dynamics $\mathbf{f} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+)$ are a function mapping states and actions to a probability distribution of following states. In order to estimate them using Gaussian Processes, some assumptions about the structure of this function are needed. First, it will be assumed that both the set of states $\mathcal{S}$ and the set of actions $\mathcal{A}$ are euclidean real valued vector spaces and that the set of terminal states $\mathcal{T}$ is empty, that is, $\mathcal{S}^+$ and $\mathcal{S}$ are assumed equal, requiring that episode endings have to be modelled separately. And secondly, the probability distribution of the following state is assumed to be unimodal. This unimodality can result from deterministic transition functions, such as the one of the bicycle benchmark defined in chapter 2, being disturbed slightly by Gaussian noise.

Estimating a function f on the basis of observations $\boldsymbol{y_i} = f(\boldsymbol{x_i}) + \epsilon_i \in \mathbb{R}$ with input vectors $\boldsymbol{x_i} \in \mathbb{R}^d$ and a noise term ϵ_i is a *regression problem*. Since the number of observations is finite and the function f is an infinite object, the estimation of f is uncertain and based on prior assumptions about its structure.

In classic control scenarios, these prior assumptions often follow from physical descriptions of the system to be modelled. While the physics of driving a bicycle is understood quite well and can be described using differential equations, a controller for a specific bicycle depends on some parameters γ such as the masses and measures detailed in table 2.3. In this setting, solving the regression problem corresponds to finding a choice of parameters γ^* which explain the observations of the true system best.

In a Bayesian context, instead of deciding on one specific vector of parameters, it might be more interesting to derive a distribution $\mathrm{p}(\gamma^*)$ of probable parameter values which then represents the uncertainty about their true value. When making a prediction for a new input point $\boldsymbol{x}_*$, this uncertainty can be used to derive a predictive distribution $\mathrm{p}(y_* \mid \boldsymbol{x}_*, \gamma^*)$, which propagates this uncertainty through the model to the prediction.

This approach represents uncertainty about the correct choice of parameters but assumes that the predefined structure of the function is correct, making it a *parametric model*. Such structure has the advantage of making it easier to find the best set of parameters, since the search space is relatively limited. It does however limit the expressiveness of the model, which can lead to bad performance. A physical description of the system might be too idealized and not take account of all factors in reality, such as the assumptions of frictionless mechanics or limited turbulences in fluid mechanics. Accounting for all possible effects can make the model very complicated. This means

that both the number of parameters becomes large and it may be hard to interpret the model in a physical sense.

Non-parametric models are not based on insights about the concrete structure of the function to be modelled but rather only make assumptions about properties of the function itself, such as smoothness or differentiability. Instead of modelling a distribution of parameter values, a Bayesian non-parametric model is concerned with finding a distribution $p(f^*)$ of probable functions which represents the believe of the model about the function f to be estimated.

Gaussian processes (GPs) are a state-of-the-art framework for non-parametric regression. They are a way of representing a probability distribution over functions in a way which is both computationally feasible and allows for Bayesian inference. This section introduces Gaussian Processes and describes how to encode a prior distribution over functions to represent preference in the space of all possible functions f. Based on observed data, GPs can be used make predictions about the predictive distribution $p(y_* \mid x_*, f^*)$ taking all functions in the distribution $p(f^*)$ into account. Since these predictions are not computationally cheap, an extension of Gaussian Processes for large data sets, sparse Gaussian processes using pseudo-inputs [SG05], is introduced last.

3.2.1 Definition

Gaussian processes are a generalization of the Gaussian distribution to function spaces. A multivariate Gaussian $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ describes a distribution over the finitely many elements in the vector $\boldsymbol{x}$. Every such element $\boldsymbol{x_i}$ is normally distributed according to $\boldsymbol{x_i} \sim \mathcal{N}(\mu_i, \Sigma_{ii})$ with a particular dependency structure between them. For every pair $(\boldsymbol{x_i}, \boldsymbol{x_j})$, their covariance is given by $\mathrm{cov}[\boldsymbol{x_i}, \boldsymbol{x_j}] = \boldsymbol{\Sigma}_{ij}$.

Modelling functions in general requires an infinite number of random variables, one for every function value. An infinite number of possibly dependent random variables mapping from the same probability space to the same value space is called a *stochastic process* and is represented via a function.

Definition 5 (Stochastic Process)
Given a probability space $(\Omega, \mathcal{F}, P)$, an index set T and a measurable space Y, a *stochastic process* $\boldsymbol{X}$ is a function

$$\boldsymbol{X} : \begin{cases} T \times \Omega \to Y \\ (t, \omega) \mapsto \boldsymbol{X_t}(\omega) \end{cases}$$

mapping indices t to Y-valued random-variables. For a fixed $\omega \in \Omega$, $\boldsymbol{X}(\cdot, \omega)$ is called a *trajectory* of the process [Åst71].

The index set of a stochastic process can be an arbitrary set. It is often interpreted as a time index which can be both discrete and continuous. A Gaussian process is a particular stochastic process.

Definition 6 (Gaussian Process)
A stochastic process $\boldsymbol{X}$ is called a *Gaussian process* if for any finite subset $\tau \subseteq T$ of its index set, the random variables $\boldsymbol{X}_\tau$ have a joint Gaussian distribution [Åst71].

When using a Gaussian process $\boldsymbol{X}$ to model a function $f : A \to B$, the index set T is assumed to be A and all random variables are B-valued. The random variable $\boldsymbol{X_a}$ then models the function value $f(a)$ for all $a \in A$. Sampling a trajectory from $\boldsymbol{X}$ corresponds to sampling one possible function f^*.

Similar to the finite case, the random variables have a dependency structure. Instead of a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$, a Gaussian process is completely determined by a *mean function* $\mu_f(a) = \mathbb{E}[f(a)]$ and a *covariance function*

$$\begin{aligned} \mathcal{K}(a, a') &:= \mathbb{E}[(f(a) - \mu_f(a))(f(a') - \mu_f(a'))] \\ &= \operatorname{cov}[f(a), f(a')] \\ &= \operatorname{cov}[\boldsymbol{X_a}, \boldsymbol{X_{a'}}] \end{aligned}$$

with $a, a' \in A$. The mean function encodes the point wise mean over all trajectories which could be sampled from $\boldsymbol{X}$. The covariance function is also called a *kernel* and describes the interaction between different parts of the function. A function which is distributed according to a Gaussian process is denoted as $f \sim \mathcal{GP}(\mu_f, \mathcal{K})$.

For convenience it is often assumed that the prior mean function μ_f is constant zero. This assumption is without loss of generality [Raso6] since in the case the mean function is known to be different to zero, the observations $(\boldsymbol{X}, \boldsymbol{y})$ can be transformed

to $\boldsymbol{y'} = \boldsymbol{y} - \mu(\boldsymbol{X})$. The Gaussian process based on the observations $(\boldsymbol{X}, \boldsymbol{y'})$ then only models the differences to the mean function. It is the covariance functions which encode the assumptions about the underlying function.

3.2.2 Kernels

Gaussian processes are collections of random variables, any finite subset of which have a joint multivariate Gaussian distribution. For any pair $(\boldsymbol{X_i}, \boldsymbol{X_j})$ of these random variables, their covariance is given by the covariance function $\operatorname{cov}[\boldsymbol{X_i}, \boldsymbol{X_j}] = \mathcal{K}(i, j)$. The pairwise covariances in a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ are given by its *covariance matrix* $\boldsymbol{\Sigma}$. For any finite set of random variables, the matrix obtained by pairwise application of the covariance function is called the *Gram matrix*.

Definition 7 (Gram Matrix)
Given a non-empty set A, a function $\mathcal{K} : A^2 \to \mathbb{R}$ and two sets $X = \{x_i \in A \mid i \in [n]\}$ and $Y = \{y_j \in A \mid j \in [m]\}$. The $n \times m$ matrix

$$\mathcal{K}(X, Y) = \boldsymbol{K_{XY}} := \Big(\mathcal{K}(x_i, y_j)\Big)_{\substack{i \in [n], \\ j \in [m]}}$$

is called the *Gram matrix* of $\mathcal{K}$ with respect to X and Y [SSo2].

In order for the Gram matrix to be a valid covariance matrix $\boldsymbol{\Sigma}$ of a Gaussian distribution, it must be positive definite. *Kernels* are functions which fulfill the property that for every possible subset of random variables, or more generally every set of elements in their domain, their induced Gram matrix is positive definite.

Definition 8 (Kernel)
Given a non-empty set A, a function

$$\mathcal{K} : A^2 \to \mathbb{R}$$

is called a *(positive definite) kernel* or *covariance function*, if for any finite subset $X \subseteq A$, the Gram matrix $\mathcal{K}(X, X)$ is positive definite [SSo2].

The kernel is crucial in encoding the assumptions about the function a Gaussian process should estimate. It is a measure of *similarity* of different points in the observed data

and of new points to be predicted. A natural assumption to make is to assume that the closer together in the domain two points lie, the more similar their function values will be. Similarly, to predict a test point, training points close to it are probably more informative than those further away.

But closeness is not the only possible reason two points could be similar. Assume a function to be modeled which is a possibly noisy sinusoidal wave with a known frequency. Then, two points which are a multiple of wavelengths apart should also have similar function values. A kernel which is not only dependent on the distance between two points but also their position in the input space is called *non-stationary*. A simple example of such a non-stationary kernel is the linear kernel.

Definition 9 (Linear Kernel)
For a finite dimensional euclidean vector space $\mathbb{R}^d$, the *linear kernel* is defined as

$$\mathcal{K}_{\text{linear}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \boldsymbol{x}^T \boldsymbol{y} = \langle \boldsymbol{x}, \boldsymbol{y} \rangle .$$

Consider a function $f : \mathbb{R} \to \mathbb{R}$ which is distributed according to a Gaussian process with the linear kernel $f \sim \mathcal{GP}(\mathbf{0}, \mathcal{K}_{\text{linear}})$. According to the definition of Gaussian processes, for any two input numbers $x, y \in \mathbb{R}$ their corresponding random variables $\boldsymbol{f}_x$ and $\boldsymbol{f}_y$ have a joint Gaussian distribution

$$\begin{pmatrix} \boldsymbol{f}_x \\ \boldsymbol{f}_y \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathcal{K}(x,x) & \mathcal{K}(x,y) \\ \mathcal{K}(y,x) & \mathcal{K}(y,y) \end{bmatrix} \right)$$

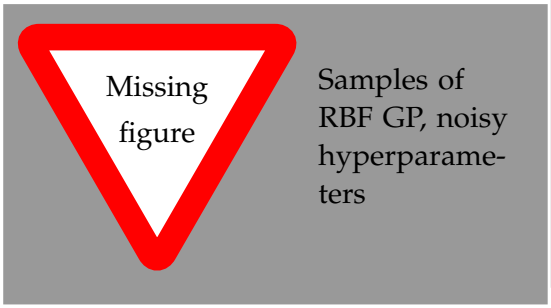
where $\mathcal{K} = \mathcal{K}_{\text{linear}}$. Assuming that both x and y are not equal to zero, the correlation coefficient ϱ of these two variables is given by

$$\begin{aligned} \varrho[\boldsymbol{f}_x, \boldsymbol{f}_y] &= \frac{\operatorname{cov}[\boldsymbol{f}_x, \boldsymbol{f}_y]}{\sqrt{\operatorname{var}[\boldsymbol{f}_x]}\sqrt{\operatorname{var}[\boldsymbol{f}_y]}} \\ &= \frac{\mathcal{K}(x,y)}{\sqrt{\mathcal{K}(x,x)}\sqrt{\mathcal{K}(y,y)}} = \frac{xy}{\sqrt{x^2}\sqrt{y^2}} \in \{-1, 1\} . \end{aligned}$$

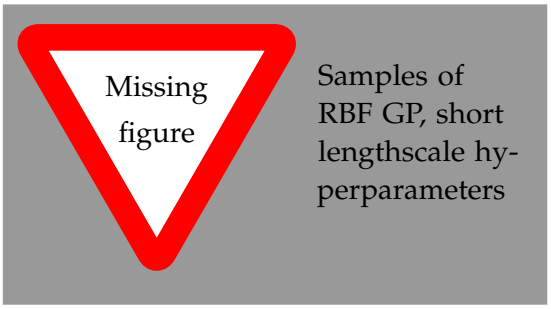
A correlation coefficient of plus or minus one implies that the value of one of the random variables is a linear function of the other. Any function drawn from this Gaussian process, such as the ones shown in figure 3.2a, is therefore a linear function. This observation generalizes to higher dimensions [Raso6]. Gaussian process regression with a linear kernel is equivalent to Bayesian linear regression.

(a) Samples of Linear GP

(b) Samples of RBF GP, good hyperparameters



(c) Samples of RBF GP, noisy hyperparameters



(d) Samples of RBF GP, short lengthscale hyperparameters

Figure 3.2: GP samples

Because of its restrictiveness, the linear kernel is not very relevant for real-world applications of Gaussian processes. As described above, the similarity of two data points $\boldsymbol{x}$ and $\boldsymbol{y}$ is often dependent on their relative position. A kernel which is a function of $\boldsymbol{x} - \boldsymbol{y}$ is called *stationary* and is invariant to translations in the input space. The most important stationary kernel is the squared exponential kernel.

Definition 10 (Squared Exponential Kernel)
For a finite dimensional euclidean vector space $\mathbb{R}^d$, the *squared exponential kernel* (or *RBF kernel*) is defined as

$$\mathcal{K}_{\text{SE}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sigma_f^2 \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{y})^T \boldsymbol{\Lambda}^{-1}(\boldsymbol{x}-\boldsymbol{y})\right).$$

The parameter $\sigma_f^2 \in \mathbb{R}_{>0}$ is called the *signal variance* and $\boldsymbol{\Lambda} = \text{diag}(l_1^2, \ldots, l_d^2)$ is a diagonal matrix of the squared *length scales* $l_i \in \mathbb{R}_{>0}$.

The similarity of two data points approaches one when they are close together and for larger distances approaches zero with exponential drop off. It can be shown that this kernel represents all infinitely differentiable functions [Raso6]. Gaussian processes with this covariance function are universal function approximators.

The squared exponential kernel is dependent on multiple parameters which influence its behaviour. In contrast to weight parameters in linear regression or constants in physical models, these parameters do not specify the estimated function but rather the prior believe about this function. In order to separate the two, they are called *hyperparameters*. The vector of all hyperparameters in a model is called $\boldsymbol{\theta}$.

The hyperparameters of the RBF kernel describe the expected dynamic range of the function. The signal variance σ_f^2 specifies the average distance of function values from the mean function. The different length scale parameters l_i roughly specify the distance of data points along their respective axis required for the function values to change considerably. Figure 3.2 compares sample functions drawn from Gaussian processes with squared exponential kernels with different hyperparameters.

These plots show continuous functions being drawn from their respective processes. It is however only possible to evaluate the Gaussian process at finitely many points and then connect the resulting samples. Drawing the function values of a finite amount of sample input points $\boldsymbol{X}_*$ from a Gaussian process prior is equivalent to drawing a sample from the Gaussian $\mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_*)$ where $\boldsymbol{K}_*$ is a short hand notation for $\mathcal{K}(\boldsymbol{X}_*, \boldsymbol{X}_*)$.

3.2.3 Predictions and Posterior

In order to use Gaussian processes for regression, it is necessary to combine observations with a Gaussian process prior $f \sim \mathcal{GP}(\boldsymbol{0}, \mathcal{K})$ in order to obtain a predictive posterior. The N data points observed are denoted as $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ with $\boldsymbol{y} = f(\boldsymbol{X}) + \mathcal{N}(\boldsymbol{0}, \sigma_n^2 \mathrm{I})$

and $|\boldsymbol{y}| = N$. The observed function values $\boldsymbol{y}$ are assumed to not be the true latent function values $\boldsymbol{f} = f(\boldsymbol{X})$ but rather have some additive Gaussian noise which is independent and identically distributed for all observations. The variance of this noise σ_n^2 is a hyperparameter of the Gaussian process model.

Assuming further that given the latent function and the input points, the observations are conditionally independent, their likelihood is given by

$$\begin{aligned} \mathrm{p}(\boldsymbol{y} \mid f, \boldsymbol{X}) = \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{f}) &= \prod_{i=1}^{N} \mathrm{p}(y_i \mid f_i) \\ &= \prod_{i=1}^{N} \mathcal{N}(y_i \mid f_i, \sigma_n^2) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma_n^2 \mathrm{I}) \end{aligned}$$

because of the assumed noise model. Given some vector of hyperparameters $\boldsymbol{\theta}$, the definition of Gaussian processes yields a joint Gaussian distribution for the latent function values $\boldsymbol{f}$ given by

$$\mathrm{p}(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{0}, \boldsymbol{K}_N)$$

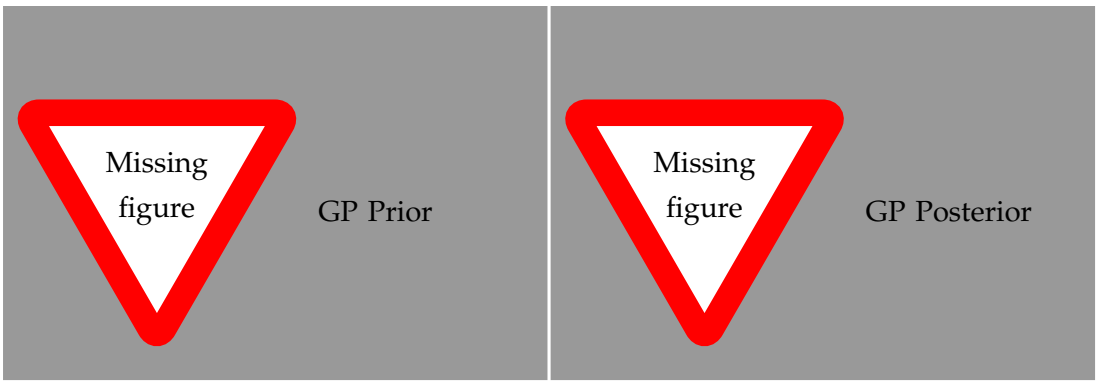
where $\boldsymbol{K}_N = \mathcal{K}(\boldsymbol{X}, \boldsymbol{X})$ denotes the Gram matrix of the observed data. Combining the two distributions according to the law of total probability yields the probability distribution of the outputs conditioned on the inputs and is given by

$$\begin{aligned} \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{f}) \, \mathrm{p}(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{f} \\ &= \int \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma_n^2 \mathrm{I}) \, \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{0}, \boldsymbol{K}_N) \, \mathrm{d}\boldsymbol{f} \\ &= \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K}_N + \sigma_n^2 \mathrm{I}). \end{aligned} \tag{3.2}$$

Note that this distribution is obtained by integrating over all possible latent function values $\boldsymbol{f}$ and thereby taking all possible function realizations into account. This integration is called the *marginalization* of $\boldsymbol{f}$. The closed form solution of the integral is obtained using well-known results about Gaussian distributions which are for example detailed in [P+08].

Now consider a set of test points $\boldsymbol{X}_*$ for which the predictive posterior should be obtained. By definition the latent function values $\boldsymbol{f}$ of the training set and the latent function values of the test set $\boldsymbol{f}_* = f(\boldsymbol{X}_*)$ have the joint Gaussian distribution

$$\mathrm{p}\left(\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{pmatrix} \,\middle|\, \boldsymbol{X}, \boldsymbol{X}_*, \boldsymbol{\theta} \right) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{pmatrix} \,\middle|\, \boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_N & \boldsymbol{K}_{N*} \\ \boldsymbol{K}_{*N} & \boldsymbol{K}_* \end{bmatrix} \right).$$



(a) GP Prior **(b)** GP Posterior

Figure 3.3: GP posterior

Adding the noise model to this distribution leads to the joint Gaussian of training outputs $\boldsymbol{y}$ and test outputs $\boldsymbol{f}_*$ which is given by

$$\mathrm{p}\left(\begin{pmatrix}\boldsymbol{y}\\ \boldsymbol{f}_*\end{pmatrix} \,\middle|\, \boldsymbol{X}, \boldsymbol{X}_*, \boldsymbol{\theta}\right) = \mathcal{N}\left(\begin{pmatrix}\boldsymbol{y}\\ \boldsymbol{f}_*\end{pmatrix} \,\middle|\, \boldsymbol{0}, \begin{bmatrix}\boldsymbol{K}_N + \sigma_n^2\mathrm{I} & \boldsymbol{K}_{N*}\\ \boldsymbol{K}_{*N} & \boldsymbol{K}_*\end{bmatrix}\right).$$

In this distribution, the training outputs $\boldsymbol{y}$ are known. The predictive posterior for the test outputs $\boldsymbol{f}_*$ can be obtained by applying the rules for marginalization of multivariate Gaussians, yielding another Gaussian distribution.

Lemma 11 (GP predictive posterior)
Given a latent function with a Gaussian process distribution $f \sim \mathcal{GP}(\boldsymbol{0}, \mathcal{K})$ and N training points $\boldsymbol{X}$ with noisy observations of the form $\boldsymbol{y} = f(\boldsymbol{X}) + \mathcal{N}(\boldsymbol{0}, \sigma_n^2\mathrm{I})$. The predictive posterior $\boldsymbol{f}_*$ of the test points $\boldsymbol{X}_*$ is then given by

$$\begin{aligned}\mathrm{p}(\boldsymbol{f}_* \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_*) &= \mathcal{N}(\boldsymbol{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \text{ where}\\ \boldsymbol{\mu}_* &= \boldsymbol{K}_{*N}\left(\boldsymbol{K}_N + \sigma_n^2\mathrm{I}\right)^{-1}\boldsymbol{y}\\ \boldsymbol{\Sigma}_* &= \boldsymbol{K}_* - \boldsymbol{K}_{*N}\left(\boldsymbol{K}_N + \sigma_n^2\mathrm{I}\right)^{-1}\boldsymbol{K}_{N*}.\end{aligned}$$

This predictive posterior makes it possible to evaluate the function approximation based on the input at arbitrary points in the input space. Since any set of these points always has a joint Gaussian distribution, the predictive posterior defines a new Gaussian

process, which is the posterior Gaussian process given the observations. This posterior process $\mathcal{GP}(\mu_{\text{post}}, \mathcal{K}_{\text{post}})$ has new mean and covariance functions given by

$$\begin{aligned}\mu_{\text{post}}(\boldsymbol{a}) &= \mathcal{K}(\boldsymbol{a}, \boldsymbol{X}) \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y} \\ \mathcal{K}_{\text{post}}(\boldsymbol{a}, \boldsymbol{b}) &= \mathcal{K}(\boldsymbol{a}, \boldsymbol{b}) - \mathcal{K}(\boldsymbol{a}, \boldsymbol{X}) \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \mathcal{K}(\boldsymbol{X}, \boldsymbol{b}).\end{aligned}$$

Note that the posterior mean function is not necessarily the constant zero function. Figure 3.3 shows samples from a pair of prior and posterior Gaussian processes.

Computing the inverse $\left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1}$ costs $\mathcal{O}(N^3)$ but can be done as a preprocessing step since it is independent of the test points. Predicting the mean of a single test point is a weighted sum of N basis functions $\mu_* = \boldsymbol{K}_{*N}\boldsymbol{\alpha}$ where $\boldsymbol{\alpha} = \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y}$ which can be precomputed. After this precomputation, predicting the mean of a single test point costs $\mathcal{O}(N)$. To predict its variance, it is still necessary to perform a matrix multiplication which costs $\mathcal{O}(N^2)$. Since all of these operations are dependent on the number of training points, evaluating Gaussian processes on large data sets can be computationally expensive. Before introducing sparse approximations with better asymptotic complexity, the next section deals with choosing good values for the vector of hyperparameters $\boldsymbol{\theta}$.

3.2.4 Choosing Hyperparameters

In the previous section, the hyperparameters $\boldsymbol{\theta}$ were assumed to be known and constant, that is, the prior assumptions about the function to be estimated were fixed. In this case, Gaussian processes do not have a training stage, since any test point can be predicted according to the predictive posterior. Usually however, the correct choice of hyperparameters is not clear a priori. A major advantage of Gaussian processes is the ability to select hyperparameters from training data directly instead of requiring a scheme such as cross validation.

In a fully Bayesian setup, the correct way to model uncertainty about hyperparameters is to assign them a prior $\mathrm{p}(\boldsymbol{\theta})$ and marginalize it to derive the dependent distributions

$$\begin{aligned}\mathrm{p}(f) &= \int \mathrm{p}(f \mid \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta} \\ \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}) &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}.\end{aligned} \tag{3.3}$$

Updating the believe about the distribution of the hyperparameters then becomes part of the process of obtaining a posterior model. A new distribution is obtained

by combining the prior with the likelihood of the training data observed using Bayes' theorem:

$$\begin{aligned} p(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}) &= \frac{p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\boldsymbol{y} \mid \boldsymbol{X})} \\ &= \frac{p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\theta} \end{aligned}$$

Is there an easy argument for this which does not rely on the likelihood?

The integration required in equation (3.3) is very hard in practice [Raso6, p. 109], since $\boldsymbol{y}$ is a complicated function of $\boldsymbol{\theta}$. Instead, a common approximation is to use a *maximum-a-postiori (MAP)* estimate of the correct hyperparameters. This estimate is obtained by maximizing $p(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y})$ and does not require evaluation of the denominator since it is constant.

For many choices of priors $p(\boldsymbol{\theta})$ this is still a hard problem. But assuming a flat prior which assigns almost equal probability to all choices of hyperparameters, it holds that

$$\begin{aligned} p(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}) &\propto p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) \\ &= \int p(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\theta})\, p(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta})\, d\boldsymbol{f}, \end{aligned}$$

that is, the posterior distribution is proportional to the likelihood term and can be obtained using a maximum likelihood estimate on the *marginal likelihood* after integrating out the function values $\boldsymbol{f}$. Optimizing this term is called a *type II maximum likelihood estimate (ML-II)*.

The marginal likelihood is an integral over a product of Gaussians obtained from the noise model and the distribution of function values according to the Gaussian process definition. It is given by

$$\begin{aligned} p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) &= \int p(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\theta})\, p(\boldsymbol{f} \mid \boldsymbol{\theta})\, d\boldsymbol{f} \\ &= \int \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma_n^2 \mathrm{I}) \cdot \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{0}, \boldsymbol{K}_N)\, d\boldsymbol{f} \\ &= \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K}_N + \sigma_n^2 \mathrm{I}) \end{aligned} \tag{3.4}$$

The solution of this integral is a Gaussian density function [P+08]. For practical reasons, it is convenient to minimize the negative logarithm of the likelihood which is given by

$$\mathcal{L}(\boldsymbol{\theta}) = -\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{2} \boldsymbol{y}^T \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y} + \frac{1}{2} \log |\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}| + \frac{N}{2} \log(2\pi).$$

The estimation of hyperparameters is the solution of the optimization problem

$$\boldsymbol{\theta}^* \in \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

and is calculated using standard approaches to non-convex optimization such as scaled conjugate gradient (SCD) techniques, since finding the derivatives of $\mathcal{L}$ is comparatively easy [Raso6]. The computational complexity of evaluating the likelihood term and its derivatives is dominated by the inversion of $\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}$.

Since this optimization scheme does not choose parameters of the function approximation directly but rather changes a small number of broad and high-level assumptions about it, overfitting does not tend to be a problem for Gaussian processes in general [Sneo7]. The sparse approximation of Gaussian processes presented in the next section chooses a small number of points in the input space to represent a large training set. The positions of these input points can be interpreted as hyperparameters to the original Gaussian process and induce a kernel function with many hyperparameters, where overfitting can become relevant.

3.2.5 Sparse Approximations using Inducing Inputs

A major drawback of Gaussian Processes in real-world applications is their high computational cost for large data sets. Assume a data set $(\boldsymbol{X}, \boldsymbol{y})$ with $|\boldsymbol{y}| = N$, then the operations on a posterior Gaussian Process are usually dominated by the inversion of the kernel matrix $\boldsymbol{K}_N$ which takes $\mathcal{O}(N^3)$ time. While this is only a preprocessing step, the cost of predicting the mean and variance of one test point remains $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ respectively. Additionally, these operations have a space requirement of $\mathcal{O}(N^2)$. The goal of sparse approximations of Gaussian Processes is to find model representations which avoid these quadratic complexities or at least restrict them to the training phase of finding hyperparameters. This section introduces one type of approximation based on representing the complete data set through a smaller set of points.

The most simple approach to achieve this is to only use a small subset of $M \ll N$ *inducing* points of the original training set and learn a normal Gaussian Process. This approach can work for data sets with a very high level of redundancy but does impose the problem of choosing an appropriate subset. While choosing a random subset can be effective [Sneo7], the optimal choice is dependent on the hyperparameters and both should therefore be chosen in a joint optimization scheme. This is a combinatorical optimization problem which can be very hard to solve in practice since the function to be optimized is very non-smooth.

To overcome this problem, *sparse pseudo input Gaussian processes (SPGP)* [Sneo7] lift the restriction of choosing inducing points from the training set and instead allow arbitrary positions in the input space. The original data set is replaced by a *pseudo data set* $(\bar{\boldsymbol{X}}, \bar{\boldsymbol{f}})$ of *pseudo inputs* $\bar{\boldsymbol{X}}$ and *pseudo targets* $\bar{\boldsymbol{f}} = f(\bar{\boldsymbol{X}})$ which are equal true latent function $f \sim \mathcal{GP}(\mathbf{0}, \mathcal{K})$. Since they are not true observations, they are assumed to be noise-free.

With known positions of the pseudo inputs and fixed hyperparameters $\boldsymbol{\theta}$, the predictive posterior of a Gaussian Process based on this pseudo data set for test points $(\boldsymbol{X}_*, \boldsymbol{f}_*)$ is given by

$$\mathrm{p}(\boldsymbol{f}_* \mid \boldsymbol{X}_*, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{K}_{*M}\boldsymbol{K}_M^{-1}\bar{\boldsymbol{f}}, \boldsymbol{K}_* - \boldsymbol{K}_{*M}\boldsymbol{K}_M^{-1}\boldsymbol{K}_{M*})$$

according to lemma 11 with the notation $\boldsymbol{K}_M = \mathcal{K}(\bar{\boldsymbol{X}}, \bar{\boldsymbol{X}})$ meaning the Gram matrix of the pseudo inputs compared to $\boldsymbol{K}_N = \mathcal{K}(\boldsymbol{X}, \boldsymbol{X})$, the Gram matrix of the original training data.

The true data set is independent given the latent function and can therefore be assumed independent given the pseudo data set which should be a good representation of it. The likelihood of the original data under the Gaussian process trained on the pseudo data set is given by

$$\begin{aligned}
\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) &= \prod_{i=1}^{N} \mathrm{p}(y_n \mid \boldsymbol{x}_n, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) \\
&= \prod_{i=1}^{N} \mathcal{N}\left(y_n \,\middle|\, \boldsymbol{K}_{nM}\boldsymbol{K}_M^{-1}\bar{\boldsymbol{f}}, \boldsymbol{K}_n - \boldsymbol{K}_{nM}\boldsymbol{K}_M^{-1}\boldsymbol{K}_{Mn} + \sigma_n^2\right) \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\bar{\boldsymbol{f}}, \operatorname{diag}\left(\boldsymbol{K}_N - \boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\boldsymbol{K}_{MN}\right) + \sigma_n^2\mathrm{I}\right) \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\bar{\boldsymbol{f}}, \operatorname{diag}\left(\boldsymbol{K}_N - \boldsymbol{Q}_N\right) + \sigma_n^2\mathrm{I}\right)
\end{aligned}$$

with $\boldsymbol{Q}_N \coloneqq \boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\boldsymbol{K}_{MN}$. The additive term σ_n^2 comes from the noise model assumed about the observations $\boldsymbol{y}$ in the original data set. Rather than using maximum likelihood on this term to learn the complete pseudo data set $(\bar{\boldsymbol{X}}, \bar{\boldsymbol{f}})$, the pseudo targets $\bar{\boldsymbol{f}}$ can be marginalized. This can be combared to the marginalization of the latent function values $\boldsymbol{f}$ in the derivation of Gaussian processes in equation (3.4). Assuming the pseudo targets to be distributed very similarly to the real data, a reasonable prior for them is given by

$$\mathrm{p}(\bar{\boldsymbol{f}} \mid \bar{\boldsymbol{X}}) = \mathcal{N}(\bar{\boldsymbol{f}} \mid \mathbf{0}, \boldsymbol{K}_M).$$

The marginalization is stated as the integral of a product of two Gaussian distributions which has a closed form solution and is given by

$$\begin{aligned}
\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \boldsymbol{\theta}) &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta})\, \mathrm{p}(\bar{\boldsymbol{f}} \mid \bar{\boldsymbol{X}})\, \mathrm{d}\bar{\boldsymbol{f}} \\
&= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta})\, \mathcal{N}(\bar{\boldsymbol{f}} \mid \mathbf{0}, \boldsymbol{K}_M)\, \mathrm{d}\bar{\boldsymbol{f}} \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \mathbf{0}, \boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\boldsymbol{K}_M \left(\boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\right)^T + \operatorname{diag}\left(\boldsymbol{K}_N - \boldsymbol{Q}_N\right) + \sigma_n^2 \mathrm{I}\right) \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \mathbf{0}, \boldsymbol{Q}_N + \operatorname{diag}\left(\boldsymbol{K}_N - \boldsymbol{Q}_N\right) + \sigma_n^2 \mathrm{I}\right).
\end{aligned}$$

This *SPGP marginal likelihood* can be interpreted as the marginal likelihood of a Gaussian process given the original data set $(\boldsymbol{X}, \boldsymbol{y})$ in equation (3.2). In this Gaussian process, the original kernel $\mathcal{K}$ is replaced by the kernel $\mathcal{K}_{\mathrm{SPGP}}$. With δ denoting the Kronecker-Delta it is defined as

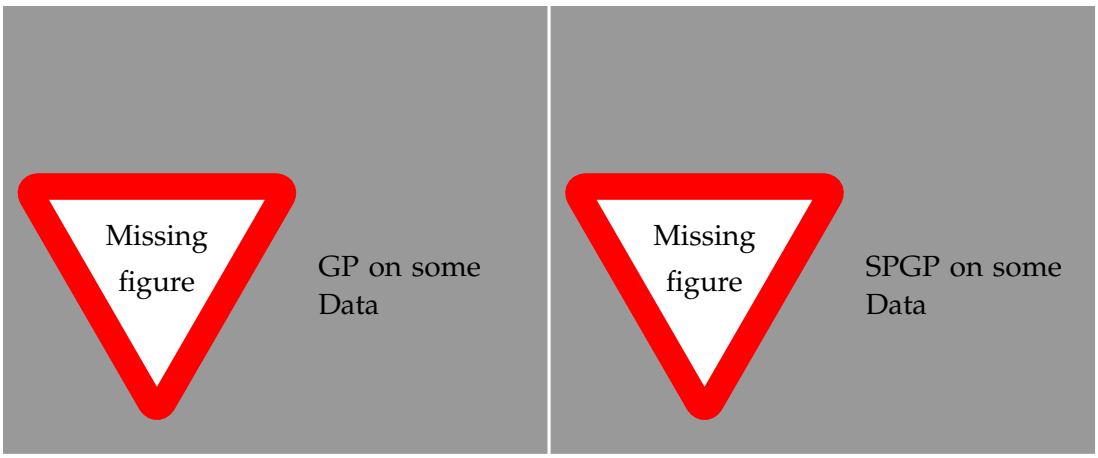
$$\begin{aligned}
\mathcal{Q}(\boldsymbol{a}, \boldsymbol{b}) &\coloneqq \boldsymbol{K}_{\boldsymbol{a}M}\boldsymbol{K}_M^{-1}\boldsymbol{K}_{M\boldsymbol{b}} \\
\mathcal{K}_{\mathrm{SPGP}}(\boldsymbol{a}, \boldsymbol{b}) &\coloneqq \mathcal{Q}(\boldsymbol{a}, \boldsymbol{b}) + \delta_{\boldsymbol{a},\boldsymbol{b}}\left(\mathcal{K}(\boldsymbol{a}, \boldsymbol{b}) - \mathcal{Q}(\boldsymbol{a}, \boldsymbol{b})\right).
\end{aligned}$$

This kernel is equal to $\mathcal{K}$ when both arguments are identical and equal to $\mathcal{Q}$ everywhere else. For well-chosen pseudo inputs, $\boldsymbol{Q}_N$ is a low-rank approximation of $\boldsymbol{K}_N$ [Sneo7]. Because of this identity, an SPGP is a normal Gaussian process with an altered kernel function. The pseudo inputs $\bar{\boldsymbol{X}}$ are hidden in the kernel matrix $\boldsymbol{K}_M$ and are additional hyperparameters to this kernel. This observation directly yields the SPGP predictive posterior using lemma 11.

Lemma 12 (SPGP predictive posterior)
Given a latent function with a sparse pseudo-input Gaussian process distribution $f \sim \mathcal{GP}(\mathbf{0}, \mathcal{K}_{\mathrm{SPGP}})$, N training points $\boldsymbol{X}$ with noisy observations of the form $\boldsymbol{y} = f(\boldsymbol{X}) + \mathcal{N}(\mathbf{0}, \sigma_n^2 \mathrm{I})$ and M positions of pseudo-inputs $\bar{\boldsymbol{X}}$. The predictive posterior $\boldsymbol{f}_*$ of the test points $\boldsymbol{X}_*$ is then given by

$$\begin{aligned}
\mathrm{p}(\boldsymbol{f}_* \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}, \bar{\boldsymbol{X}}) &= \mathcal{N}(\boldsymbol{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \text{ where} \\
\boldsymbol{\mu}_* &= \boldsymbol{Q}_{*N}\left(\boldsymbol{Q}_N + \operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y} \\
\boldsymbol{\Sigma}_* &= \boldsymbol{K}_* - \boldsymbol{Q}_{*N}\left(\boldsymbol{Q}_N + \operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{Q}_{N*}.
\end{aligned}$$

and $\boldsymbol{Q}_N \coloneqq \boldsymbol{K}_{NM}\boldsymbol{K}_M^{-1}\boldsymbol{K}_{MN}$.



(a) GP on some Data **(b)** SPGP on some Data

Figure 3.4: GP vs. SPGP

The predictive distribution as written in the previous equations can easily be compared to the predictive posterior of Gaussian processes in lemma 11. They do however still involve the inversion of matrices of size $N \times N$ and therefore do not offer computational improvements. Using the matrix inversion lemma [P+08], they can be rewritten to the form

fix line spacing in equation

$$\begin{aligned}
\boldsymbol{\mu}_* &= \boldsymbol{K}_{*M}\boldsymbol{B}^{-1}\boldsymbol{K}_{MN}\left(\operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2\mathrm{I}\right)^{-1}\boldsymbol{y} \\
\boldsymbol{\Sigma}_* &= \boldsymbol{K}_* - \boldsymbol{K}_{*M}\left(\boldsymbol{K}_M^{-1} - \boldsymbol{B}^{-1}\right)\boldsymbol{K}_{M*} \\
\boldsymbol{B} &= \boldsymbol{K}_M + \boldsymbol{K}_{MN}\left(\operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2\mathrm{I}\right)^{-1}\boldsymbol{K}_{NM},
\end{aligned}$$

which only involves the inversion of $M \times M$ matrices and one diagonal $N \times N$ matrix. Implemented this way, the calculation of all terms independent of the test points has a complexity of $\mathcal{O}(NM^2)$ and predicting means and variances takes $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ time respectively. The space requirement also drops to $\mathcal{O}(M^2)$.

Since the positions of the pseudo inputs $\bar{\boldsymbol{X}}$ are additional hyperparameters in $\mathcal{K}_{\text{SPGP}}$, they can be chosen together with the hyperparameters of the original kernel $\boldsymbol{\theta}$ using maximum likelihood as explained in section 3.2.4. Because they can be placed anywhere in the input space, the derivatives of the marginal likelihood by their positions are smooth functions [SG05]. This optimization chooses the positions in such a way that together with appropriate other hyperparameters, the original data is represented as good as possible. The curse of dimensionality of requiring exponentially many points

in a grid given the number of input dimensions does therefore not necessarily apply to the number of pseudo inputs needed in an SPGP approximation. Figure 3.4 shows that a surprisingly small number of pseudo inputs can be enough to represent the dynamics of a function.

With a large number of pseudo inputs, the number of hyperparameters can grow large. This implies the danger of overfitting since the altered Gaussian process has no direct connection to the original Gaussian process over the complete training set. As an alternative to the optimization of the SPGP marginal likelihood, Titsias proposed a variational approach [Tit09] which optimizes a lower bound of the marginal likelihood of the original Gaussian process. Instead of choosing a sparse model which explains the data well, this optimization chooses a sparse model which is as close as possible to the original full GP. Since this strategy leads to better convergence and more robust results in practice, these variational sparse approximation of full Gaussian processes is used to model transition dynamics within this thesis.

In order to solve the control problem of the bicycle benchmark, the next step after modelling the transition dynamics using Gaussian processes is to find a policy representation. Instead of a closed form representation of the policy, the choice of which action to take is made by directly optimizing over the value function using particle swarm optimization. This technique is presented in the next section.

3.3 Particle Swarm Optimization Policy

Given a starting state, the way to evaluate a policy in the general reinforcement learning setup is to evaluate its value function as defined in definition 4. Since the value is defined as the sum of expected rewards of all future states in the time horizon, calculating it correctly requires knowledge of the true transition function or, in the case of deterministic dynamics, interaction with the system. Model-based reinforcement learning is centered around learning a model of the transition function and use this representation to approximate the value function.

This allows simulated interaction with the system and can be used to choose optimal parameters for a parametric closed-form policy formulation. Similar to the comparison of parametric and non-parametric models in section 3.2, the performance of such policies heavily depends on the choice of function representation. In a Bayesian setting, the uncertainties about the transition model should be propagated through to an uncertainty about the choice of policy parameters and finally marginalized to obtain a distribution over good actions to take. In order to keep this process computationally

feasible, trade-offs must be made in the flexibility of policy representations. Deisenroth and Rasmussen recommend the use of rather limited linear policies or non-linear representations via RBF networks [DR11].

This thesis chooses the non-parametric approach of the *particle swarm optimization policy* described by Hein et al. in [Hei+16]. For a finite time horizon T, finding an optimal parametric policy using the approximated value function means finding policy parameters which result in good choices for actions at every time step. In contrast to this, PSO-P directly examines the *action-value-function* $\hat{\mathbf{V}}$ given by

$$\hat{\mathbf{V}}_{s_0} : \begin{cases} \mathcal{A}^T \to \mathbb{R} \\ (\boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}}) \mapsto \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \mathbf{R}(\boldsymbol{s_t}) \,\middle|\, f, \boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}} \right] \end{cases} \tag{3.5}$$

where f denotes a model of the transition function, $\boldsymbol{s_0}$ denotes a state and γ is the discount factor. PSO-P optimizes over the vector of all possible actions in the time horizon and then applies the best action found for time step zero to the system.

Definition 13 (PSO-P)
The *particle swarm optimization policy (PSO-P)* [Hei+16] chooses actions via direct optimization of the action value function $\hat{\mathbf{V}}$ and is defined as

$$\pi_{\text{PSO-P}} : \begin{cases} \mathcal{S} \to \mathcal{A} \\ \boldsymbol{s} \mapsto \boldsymbol{a_0^*}, \text{ where } \boldsymbol{a^*} \in \underset{\boldsymbol{a} \in \mathcal{A}^T}{\operatorname{argmax}} \hat{\mathbf{V}}_{\boldsymbol{s}}(\boldsymbol{a}) \cdot \end{cases}$$

The optimization over $\hat{\mathbf{V}}$ at time step t yields a vector of optimal actions for the complete time horizon. The PSO-Policy still only applies the first of these actions and repeats the optimization at time step $t+1$. This is done to both reduce model bias and profit from the more accurate information about $\boldsymbol{s_{t+1}}$ from the system.

The action value function is itself not probabilistic but is defined as an expected value dependent on all future states. Since the transition model is Bayesian, evaluating this expected values implies iterated marginalizations of beliefs about both model hyperparameters and intermediate states. The dependency of the action value on the different actions is therefore very complex and finding their gradients for the optimization is analytically intractable.

Particle swarm optimization is the heuristic technique which is used in PSO-P to solve this non-convex optimization problem and is not dependent on gradients. It works by

creating a set of candidate solutions in the search space. These particles move through the search space with the direction being dependent on their own function value and the function values of the other particles in the swarm. This section introduces a basic version of the algorithm and discusses the different choices of parameters.

3.3.1 Basic Particle Swarm Optimization

To choose an appropriate action for the current state, PSO-P needs to solve a non-convex optimization problem. Classical non-linear optimization schemes such as the scaled conjugated gradient technique or Newton methods [Preo7] rely on the evaluation of the first or even second derivatives of the objective function. In the setting of optimizing the action value function with respect to all actions at the different time steps, finding these gradients is a hard problem.

Particle swarm optimization is a heuristic technique which does not assume knowledge about the derivatives of the objective function and is therefore called *gradient-free*. It is a population-based optimization approach, where a number of solution candidates move through the domain of the target function in search of a good solution. The most well-known class of population-based approaches, which are often inspired by nature, are evolutionary or genetic algorithms. Genetic algorithms mimic natural selection by evaluating a generation of individuals and allowing the most successful to survive and recombine, giving rise to a new generation of candidates.

In contrast, the set of individuals, or *swarm*, remains constant in PSO. Instead of implementing survival of the fittest, PSO is based on social interaction. Every individual, or *particle*, flies in the search space with a velocity which is dynamically adjusted according to its own past experience and the experience of the other particles in the swarm. While they are initialized randomly throughout the space, they are expected to collapse on a single point which, ideally, should be the optimum.

More formally, PSO is mostly used in a non-convex optimization setting. Given some function $f : \mathcal{X} \to \mathbb{R}$, the problem is to find an $\boldsymbol{x}^* \in \mathcal{X}$ such that

$$\boldsymbol{x}^* \in \operatorname*{argmax}_{\boldsymbol{x} \in \mathcal{X}} f(x).$$

While variantes of PSO exist which can handle arbitrary constraints [Engo6], the basic variant presented in this thesis assumes that the domain $\mathcal{X}$ of f is a set of the form

$$\mathcal{X} := \left\{ \boldsymbol{x} \in \mathbb{R}^d \;\middle|\; x_i^{\min} \leq x_i \leq x_i^{\max} \right\}.$$

This set of feasible points $\mathcal{X}$ is an axis-parallel cuboid in the finite dimensional real vector space $\mathbb{R}^d$. The boundaries of this cuboid are defined by the vectors $\boldsymbol{x}^{\min}$ and $\boldsymbol{x}^{\max}$ which are both in $\mathbb{R}^d$ and specify the minimal and maximal value for every dimension.

The swarm of a PSO run consists of a finite set $\mathcal{P}$ of particles. These particles each have a position and velocity which are both updated for all particles simultaneously and at discrete time steps. The set of all time steps $\mathcal{T}$ is usually considered to be the natural numbers. At every such time step, the objective function value is evaluated for every particle. Since PSO models social interaction and communication, every particle has a neighbourhood of particles it communicates with. This communication influences the particle's velocity as it is attracted by particles with good performance.

Definition 14 (PSO Instance)
An instance of the *particle swarm optimization (PSO)* scheme on the objective function $f : \mathbb{R}^d \to \mathbb{R}$ is defined by the set of particles $\mathcal{P}$. For every time step in $\mathcal{T}$, the particles' positions and velocities are given by the functions

$$\mathbf{x} : \mathcal{P} \times \mathcal{T} \to \mathbb{R}^d \quad \text{and}$$
$$\mathbf{v} : \mathcal{P} \times \mathcal{T} \to \mathbb{R}^d$$

respectively. The position of particle i at time step t is denoted as $\mathbf{x}_i(t)$ and its velocity as $\mathbf{v}_i(t)$. A particle can be influenced by the set of its neighbours given by the neighbourhood function

$$\mathcal{N} : \mathcal{P} \to 2^{\mathcal{P}}.$$

The positions and velocities of the particles are defined using dynamic programming. The bounded search space allows for a uniform initial distribution of positions and velocities which are, for every particle $i \in \mathcal{P}$, defined as

$$\mathbf{x}_i(0), \mathbf{v}_i(0) \sim \boldsymbol{x}^{\min} + \mathbb{U}(0,1) \cdot \left(\boldsymbol{x}^{\max} - \boldsymbol{x}^{\min}\right),$$

where $\mathbb{U}(0,1)$ denotes the standard uniform distribution. The update of the positions only depends on the particle's current state, such that for every $t \in \mathcal{T}$ it is defined as

$$\mathbf{x}_i(t+1) \coloneqq \mathbf{x}_i(t) + \mathbf{v}_i(t).$$

This operation can move particles out of the set of feasible points. Engelbrecht discusses multiple possible boundary conditions such as circular algebra or reflection [Engo6]. The most simple solution is to stop the particle at the boundary.

Velocities can be thought of as the result of forces pulling on the particles. Ideally, these forces would originate from the optimum of the function, but this position is unknown. In classical optimization schemes, gradients are used as the forces as they point to local extrema. Since gradients are not available, the particles must rely on other sources of information. At the first time step, no information about the function or the search space is available, so particle velocities are initialized randomly. At later steps, the particles have already visited parts of the search space and have gained some knowledge.

This knowledge is represented by the personal best position $\mathbf{y}$ of every particle, which is the best position in the search space the particle has visited since the first time step. A particle is pulled towards its own personal best position since for the single individual, it is the best guess for the position of the maximum. It is defined as

$$\mathbf{y}_i(t) \coloneqq \mathbf{x}_i \left(\min \operatorname*{argmax}_{t' \in [t]} f(\mathbf{x}_i(t')) \right).$$

Since the particles are initialized randomly in the search space, most of these personal best positions are expected to not be very good. Particles at bad parts of the search space should communicate with their neighbours and be pulled towards the most successful. For every particle, the best position seen by any of its neighbours $\hat{\mathbf{y}}$ is defined as

$$\begin{aligned} \hat{\mathbf{y}}_i(t) &\coloneqq \mathbf{y}_{n^*}(t), \text{ where} \\ n^* &\in \operatorname*{argmin}_{n \in \mathcal{N}(i)} f(\mathbf{y}_n(t)). \end{aligned}$$

The update of the velocity of a particle is a linear combination of these two components and the previous velocity in order to simulate inertia. It is defined as

$$\mathbf{v}_i(t+1) \coloneqq \underbrace{\omega \cdot \mathbf{v}_i(t)}_{\text{inertia}} + \underbrace{\gamma_c \cdot r_c \cdot (\mathbf{y}_i(t) - \mathbf{x}_i(t))}_{\text{cognitive component}} + \underbrace{\gamma_s \cdot r_s \cdot (\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t))}_{\text{social component}}. \tag{3.6}$$

The real constants ω, γ_c and γ_s weigh the relative influences of the different components and $r_c, r_s \sim \mathbb{U}(0,1)$ introduce a stochastic element to the algorithm which enables some random exploration after the first time step. Since both the cognitive and social components are proportional to the distance of the particle to either the personal or the

social best position, velocities can become very large. A common solution to avoid this is *velocity clamping* which bounds the maximum velocity of a particle to a fraction ζ of the size of the input space.

Using the update steps for the positions and velocities, the behaviour of the swarm $\mathcal{P}$ can be calculated for an arbitrary number of time steps. The result of the optimization using PSO is the best position visited by any particle at any time step and is given by

$$\boldsymbol{x}^* \in \lim_{t \to \infty} \operatorname*{argmax}_{p \in \mathcal{P}} f(\mathbf{y}_p(t)).$$

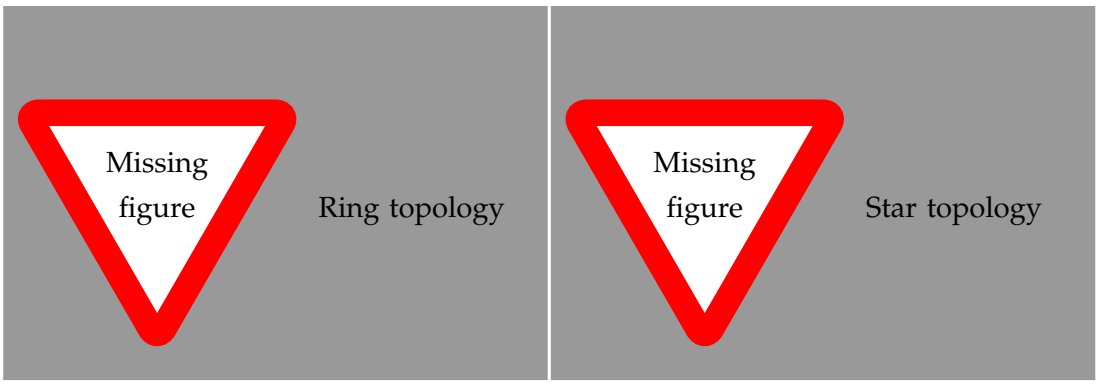
If f is bounded, this limit is well-defined since $f(\mathbf{y}_p(\cdot))$ is monotonically increasing for all $p \in \mathcal{P}$. Assuming a neighbourhood which transitively connects all particles, an arbitrary amount of time and a unique global optimum which gets visited by a particle at some time, the swarm should intuitively collapse to this global optimum, since the best particle attracts all other particles. While the heuristic is surprisingly successful in practice, the basic PSO algorithm can be shown to not converge to a single point or to the global optimum in scenarios which are not so ideal [Engo6].

Since PSO cannot judge whether it has converged to the limit, standard termination criteria for optimization have to be employed. Common problem-dependent choices are a minimum step size between successive best solutions, minimum improvement of the objective function between successive best solutions or a maximum number of iterations or iterations without improvement. Besides these criteria, there are many other variable parameters in PSO such as the number of particles or the structure of the neighbourhood function.

3.3.2 Choosing Parameters

The performance of basic PSO depends on choices for multiple parameters, namely the number of particles in the swarm, the structure of the neighbourhood, the number of iterations and the constants in the velocity update step. Since it is a heuristic algorithm, the correct settings of these parameters are problem-dependent and it is hard to give mathematical proofs. This subsection describes the empirical findings of this thesis and summarizes the recommendations given by Engelbrecht in [Engo6].

As with many other optimization algorithms, there are two important trade-offs when deciding on parameter values. PSO evaluates the objective function $\mathcal{O}(|\mathcal{P}| \cdot |\mathcal{T}|)$ times, about once per particle and time step. This introduces the first compromise between computational time and the quality of the result. Both a higher number of particles



(a) Ring topology **(b)** Star topology

Figure 3.5: PSO topologies

and time steps can be expected to improve the overall best position, up to a point of saturation where the algorithm converges. Therefore, this product should be as large as the available computational resources allow. The minimum number of function evaluations required is highly dependent on the structure of the search space. Too few iterations can terminate the search prematurely, not giving the swarm time to collapse to a good solution. On the other hand, too few particles can mean that relevant parts of the search space are never visited.

Dividing computational power between the number of particles and the number of time steps introduces the second trade-off of exploration and exploitation. The more particles there are in the swarm, the more diversity is there in the initial positions of the swarm. During the first few iterations of PSO where particles are just starting to influence each other, more unknown parts of the search space will be visited and therefore more of the search space will be explored. If $|\mathcal{P}|$ is increased in expense of the number of iterations, more particles mean a higher chance that some of them find good parts of the search space. However, fewer iterations can mean that there is less time for the swarm to collapse towards these good parts and search them more thoroughly to exploit the gained knowledge. Engelbrecht recommends a number of about 10 to 30 particles as a good starting point for an unknown problem but suggests cross-validation to improve the results. In this thesis, PSO is used to optimize in the space of all future actions in the time horizon. To ensure good coverage of this space, which for the bicycle benchmark is about 50-dimensional, 250 particles were used during the experiments.

The feature separating PSO from other population-based optimization schemes is

social interaction. Particles in the swarm communicate with their neighbours to obtain additional knowledge and move towards better parts of the search space. The structure of this neighbourhood is determined by the constant neighbourhood function $\mathcal{N}$ and the performance of PSO depends strongly on its structure. The information flow through this network is described by the average degree of connectivity and the average distance between two arbitrary particles.

The average degree of connectivity $\lambda_{\mathcal{N}}$ is the average fraction of all available particles a particle is connected to. It the degree of connectivity is high and the average distance between two arbitrary particles low, information about a good position in the search space propagates quickly to all particles and the swarm will tend to collapse sooner. For simple or unimodal objective functions, this behaviour can be very beneficial, since it lowers the number of iterations required for the swarm to collapse towards the global optimum. For more complex problems however, the swarm is in danger of quickly collapsing in a local optimum without good exploration of the search space.

Engelbrecht describes a number of different social network structures. This thesis uses the *ring topology* where every particle communicates with itself and a number of immediate neighbours as shown in figure 3.5a. A higher degree of connectivity can be achieved by increasing the number of immediate neighbours a particle is connected to. This topology guarantees transitive connections between every particle while still allowing for enough exploration since knowledge must potentially pass through several particles to reach a particle on the other side of the ring. Neighbourhoods move smoothly around the ring and overlap, which means that this topology does not favour the formation of clusters of particles in the search space. The special case of the ring topology where $\lambda_{\mathcal{N}}$ is one is the *star topology* as shown in figure 3.5b. In this topology, every particle is connected with every other particle, which implies that the PSO will quickly converge towards the globally best solution.

Having established a swarm of particles with a neighbourhood $\mathcal{N}$, the constants ω, γ_c and γ_s in the velocity update in equation (3.6) define the relative influences of the different kinds of information a particle has gathered to its trajectory. The cognitive weight γ_c and the social weight γ_s describe the trust of a particle towards its own experiences and the experiences of its neighbours. If γ_s is equal to zero, the communication of the particles is ignored and every particle performs a local optimization independent of the rest of the swarm. On the other hand, if γ_c is zero in a star topology, every particle would be attracted towards the single best known position and PSO turns into a stochastic hill-climber [Eng06]. The strength of PSO comes from weighing the two aspects, so a common choice is to set both constants to similar values. The inertia weight ω is meant to ensure smooth trajectories and is set to be between

Table 3.1: The PSO parameters used in this thesis.

Parameter	Description	Value
$\lvert\mathcal{P}\rvert$	Number of particles	250
$\lvert\mathcal{T}\rvert$	Maximum number of time steps	80
$\mathcal{N}$	Neighbourhood topology	Ring
$\lambda_{\mathcal{N}}$	Average connectivity in the neighbourhood	0.1
ω	Inertia weight	0.72981
γ_c	Cognitive weight	1.49618
γ_s	Social weight	1.49618
ζ	Velocity clamping factor	0.1

zero and one to allow the swarm to converge. Based on empirical studies, Eberhart and Shi suggest setting ω to 0.7291 and both γ_c and γ_s to 1.49618 [ES00].

Alex suggested these are actually based on a proof?

The velocity clamping factor ζ defines the maximum absolute value of the velocity per dimension to be given by $\boldsymbol{v}^{\max} = \zeta \cdot (\boldsymbol{x}^{\max} - \boldsymbol{x}^{\min})$. A small value avoids erratic movement of the particles by jumping over huge parts of the search space in one time step. Like the weight factors, the velocity clamping factor is defined to be constant throughout a PSO run within this thesis. It is chosen as 0.1 which allows a particle to cross the complete space in comparatively few iterations but still limits its dynamic range.

In the experiments of this thesis, the convergence of PSO did not present a problem. For other problem instances, time dependent values of the constants can be used to influence the behaviour of the algorithm. It may be interesting to switch focus from exploration in the early time steps to exploitation in the later time steps and finally force convergence. This can be achieved by adaptively changing the different weights or the velocity clamping factor [Eng06].

The choices for the parameters of the PSO used in this thesis are presented in table 3.1. The problem of choosing a vector of actions in PSO-P is very high dimensional. Because of this, the ratio of number of particles to number of time steps leans towards a higher than usual number of particles. This is compensated for with a high average connectivity in the ring neighbourhood. The different weights and the velocity clamping factor are standard choices since they proved to be successful.

3.4 Summary

This chapter presented the mathematical framework of reinforcement learning to describe solutions to the bicycle benchmark presented in chapter 2 with mathematical rigor. With Gaussian Processes and the PSO-Policy it introduced the two main tools used in this thesis to solve it in a Bayesian way. The next chapter introduces a standard approach to controlling the bicycle with PSO-P and compares it to two strategies which incorporate uncertainty into their decision making process.

Chapter 4

Uncertainties in the Bicycle Benchmark

Chapter 3 introduced reinforcement learning as a general mathematical framework to describe the problem of controlling a bicycle in the benchmark presented in chapter 2. This thesis is concerned with model-based reinforcement learning, where the transition function of the system to be controlled is represented with some function approximation which is learnt from observations of the system and is used to make predictions about the future. Learning a model of the true system introduces model bias, where actions considered to be good with respect to the model's predictions can show bad performance in reality because the model is incorrect. In order to reduce this bias, Gaussian Processes can be used which do not only yield one specific function approximation but a distribution over all plausible models. This uncertainty about the correct model can be propagated through to predictions about specific test points and instead of a single point, Gaussian processes predict a Gaussian distribution about possible function values.

Modelling the transition dynamics allows the prediction of a state s_{t+1} given a concrete pair of state and action (s_t, a_t) for the previous time step. Assuming a deterministic model which yields exactly one posterior state, the *long-term prediction* of states multiple time steps into the future reduces to iterated one-step predictions. Given a (deterministic) reward function $\mathbf{R}$ as detailed in definition 3, it is possible to evaluate the action value function $\hat{\mathbf{V}}$ of PSO-P and thus directly use the model to extract a policy, since the expected value in equation (3.5) is a simple sum of deterministic values.

In the Bayesian context however, the transition model predicts a distribution over posterior states. This complicates long-term predictions since the uncertainty of intermediate states has to be propagated through the (non-linear) transition model to accumulate the uncertainties of multiple predictions. Additionally, the original deterministic reward function possibly has to be adapted to make it possible to evaluate the expected reward for all time steps. Once these problems are solved and the action value function can be

calculated, PSO-P can be used in the same way as in the deterministic case to choose appropriate actions.

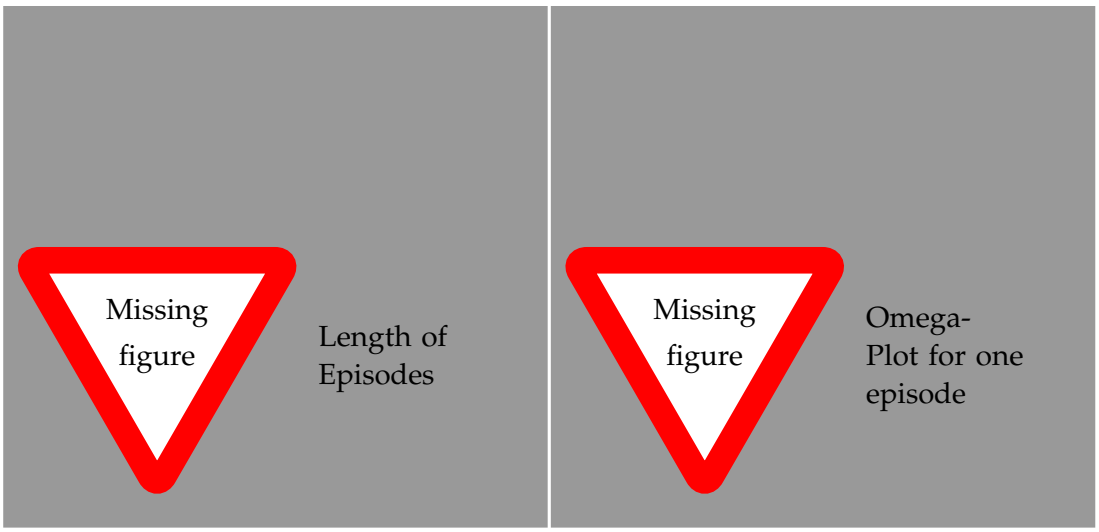
Based on the bicycle benchmark, this chapter compares the classical deterministic long-term predictions to two approaches of integrating uncertainty into the predictions. The first section describes the creation of data sets used to train the transition model and the design-choices made to obtain suitable models. These models are then interpreted as deterministic to obtain a base-line for comparison in the next section. The last two sections describe how to use uncertainties in the planning. The first approach is to use the one-step uncertainties of the Gaussian Process models in the reward function but to still create long-term predictions using deterministic states. The second fully Bayesian approach completely propagates state uncertainties both to the reward function and subsequent states.

4.1 Transition Dynamics

The goal in the bicycle benchmark is to learn to both balance a bicycle and to ride it to a target position. The state of a bicycle, as described in chapter 2, consists of the real valued vector $(\theta, \dot{\theta}, \omega, \dot{\omega}, x, y, \psi)$, where θ is the angle of the handlebars, ω is the vertical angle of the bicycle frame, x and y are euclidean coordinates and ψ is the orientation of the bicycle. The bicycle starts in almost upright position and the task of the controller is to choose the actions (d, T), where d is the horizontal leaning displacement of the driver and T is the torque applied to the handlebars at every time step. The transition dynamics are derived from a physical approximation of the system and are completely deterministic.

This thesis assumes that the actor is not allowed to interact with the system in order to try or improve its policy. In contrast, the agent is presented with a predefined data set of observations of the system obtained with a simple and sub-optimal controller. This constraint is meant to mimic industrial systems where it is comparatively cheap to obtain measurements of running systems but allowing an agent to explore is either very expensive or a security concern. It is therefore not possible to apply an on-line learning scheme on the system or to explore in specific directions in order to improve the dynamics model.

This section first describes how data sets are sampled from the bicycle benchmark in order to simulate this constraint. These data sets are then used to train the Gaussian Processes used in PSO-P to form a controller.



(a) Length of Episodes **(b)** Omega-Plot for one episode

Figure 4.1: Data Set properties

4.1.1 Data Sets

The bicycle benchmark's dynamics are introduced in chapter 2 by defining the derivatives of the state variables and choosing values for the relevant constants in table 2.3. Given a starting state and an appropriate number of actions, these derivatives can be used to approximate the future behaviour of the system using iterative numerical methods for approximating ordinary differential equations. For this thesis, the bicycle benchmark was implemented in Python [Ros95] with NumPy [WCV11] and using the classical Runge-Kutta scheme [Kut01].

In their experiments, Randløv and Alstrøm chose a time discretization of 0.01 seconds. During this time, the action applied by the agent remains constant and after one such time step, the agent can choose a new action This results in a controller frequency of 100 Hz. A high frequency gives the agent a high degree of control which is often not possible to achieve in real systems. The experiments in this thesis are based on the same time discretization of 0.01 seconds but only allow the actor to choose a new action every ten time steps, keeping it constant for the time steps in between. This yields a controller frequency of 10 Hz. Combined with the differential equations, the choice of time discretization completely defines the interface between the actor and the bicycle system. The resulting transition dynamics used for the interaction of the controller and

Algorithm 1 Sampling bicycle transitions

Let $\mathbf{f}_{\text{bicycle}}$ denote the transition function of the bicycle benchmark. The minimal and maximal values for the state variables and the actions can be found in tables 2.1 and 2.2.

function SAMPLEBICYCLESTATE
  $\left(\theta, \dot{\theta}, \omega, \dot{\omega}\right) \leftarrow \mathcal{N}(\mathbf{0}, 1/4 \cdot \text{diag}\left(\theta^{\max}, \dot{\theta}^{\max}, \omega^{\max}, \dot{\omega}^{\max}\right))$
  $x \leftarrow \mathbb{U}(x^{\min}, x^{\max})$
  $y \leftarrow \mathbb{U}(y^{\min}, y^{\max})$
  $\psi \leftarrow \mathbb{U}(-\pi, \pi)$
  return $\left(\theta, \dot{\theta}, \omega, \dot{\omega}, x, y, \psi\right)$

function SAMPLEACTION
  $d \leftarrow \mathbb{U}(d^{\min}, d^{\max})$
  $T \leftarrow \mathbb{U}(T^{\min}, T^{\max})$
  return (d, T)

function SAMPLETRANSITIONS(N)
  for $i \leftarrow 1, N$ **do**
    $\boldsymbol{s_i} \leftarrow$ SAMPLEBICYCLESTATE()
    $\boldsymbol{a_i} \leftarrow$ SAMPLEACTION()
    $\boldsymbol{s'_i} \leftarrow \mathbf{f}_{\text{bicycle}}(\boldsymbol{s_i}, \boldsymbol{a_i})$
  return $\left(\left(\boldsymbol{s_1}, \boldsymbol{a_1}, \boldsymbol{s'_1}\right), \left(\boldsymbol{s_2}, \boldsymbol{a_2}, \boldsymbol{s'_2}\right), \ldots, \left(\boldsymbol{s_N}, \boldsymbol{a_N}, \boldsymbol{s'_N}\right)\right)$

Algorithm 2 Sampling a bicycle trajectory

Let $\mathbf{f}_{\text{bicycle}}$ denote the transition function of the bicycle benchmark.

function SAMPLEBICYCLESTARTSTATE
  $(\_, \_, \_, \_, x, y, \psi) \leftarrow$ SAMPLEBICYCLESTATE()
  return $(0, 0, 0, 0, x, y, \psi)$

function SAMPLETRAJECTORY
  $\boldsymbol{s_0} \leftarrow$ SAMPLEBICYCLESTARTSTATE()
  $t \leftarrow 0$
  while $\boldsymbol{s_t}$ is not terminal **do**
    $\boldsymbol{a_t} \leftarrow$ SAMPLEACTION()
    $\boldsymbol{s_{t+1}} \leftarrow \mathbf{f}_{\text{bicycle}}(\boldsymbol{s_t}, \boldsymbol{a_t})$
    $t \leftarrow t + 1$
  return $\left(\left(\boldsymbol{s_0}, \boldsymbol{a_0}, \boldsymbol{s_1}\right), \left(\boldsymbol{s_1}, \boldsymbol{a_1}, \boldsymbol{s_2}\right), \ldots, \left(\boldsymbol{s_{t-1}}, \boldsymbol{a_{t-1}}, \boldsymbol{s_t}\right)\right)$

Algorithm 3 Sampling a bicycle data set

```
1: function SampleMixedBicycleDataSet(N)
2:     D ← ∅
3:     while |D| < N do
4:         T ← SampleTrajectory()
5:         R ← SampleTransitions(|T|)
6:         D ← D ∪ T ∪ R
7:     return D
```

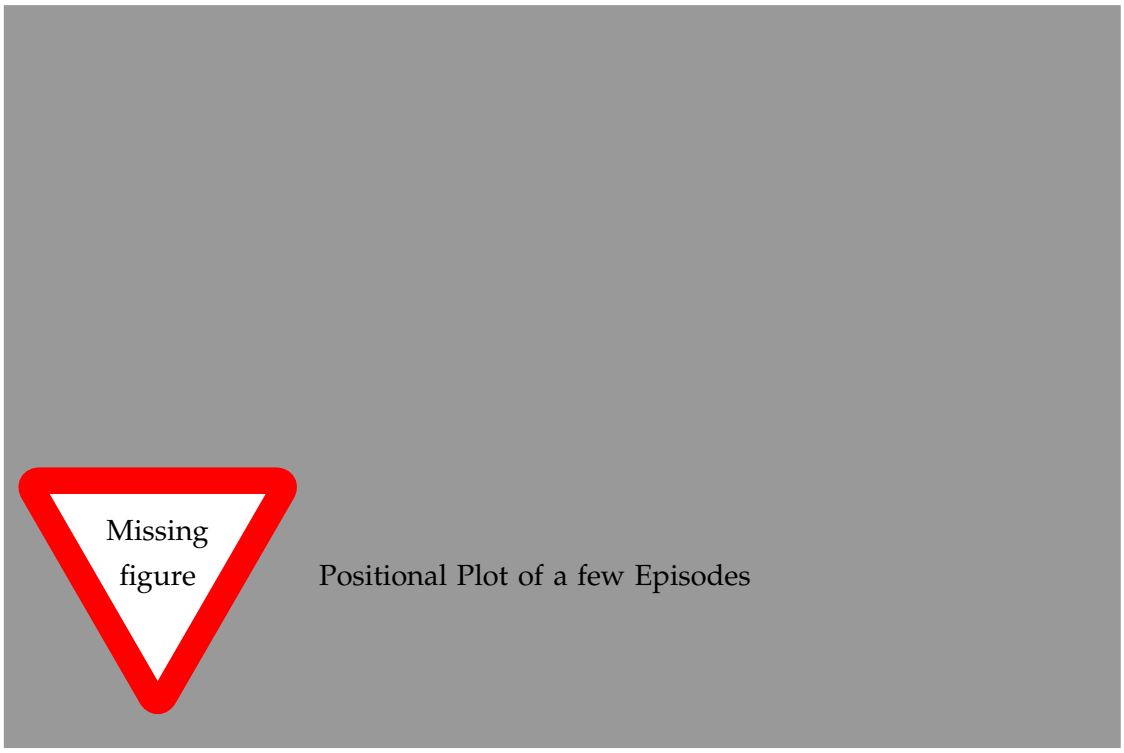


Figure 4.2: Positional Plot of a few Episodes

the system are called $\mathbf{f}_{\text{bicycle}}$ in the following.

Applying a controller to the bicycle benchmark produces time series beginning at some starting state and ending when the cyclist either falls down or reaches the goal. It is assumed that no expert knowledge is available, so the data sets available for learning transition dynamics should not be based a controller which can successfully balance the bicycle. Instead, this thesis chooses an uninformed controller which applies random actions to the system.

Algorithms 1 to 3 describe how data sets were created for the experiments. A data set consists of both complete trajectories and single random samples from the state space. A trajectory always starts in an upright position, that is, the state variables θ, $\dot{\theta}$, ω and $\dot{\omega}$ are all set to zero, while the remaining positional variables are sampled uniformly. This is both a sensible assumption and increases the mean lengths of the sampled trajectories when compared to more random starting states. As shown in figures 4.1a and 4.2, an average trajectory in the data set is quite short, since random actions are not suitable to balance the bicycle.

Figure 4.1b shows this in more detail, as it depicts the values of ω for a typical trajectory. While the bicycle starts in an upright position, it quickly starts leaning heavily towards one side and, since the controller does not choose actions to stabilize the bicycle, falls over. The sampled trajectories do not contain many state transitions where the bicycle drives straight or the actions counteract falling.

In order to reduce this bias, the data set also contains random samples from the complete state space as shown in algorithm 3. While those random samples add more balanced observations of the system, they also increase the difficulty of the learning problem. Not every combination of angles in the state space is sensible and can be reached from an upright starting state by applying actions. The transition models therefore also have to learn irrelevant information about the dynamics. A heuristic to reduce the amount of improbable states is to not sample the angles uniformly but rather to sample them from a broad random distribution around zero, resampling values which fall outside of the range of allowed values. Since terminal states are modelled separately, both the last transition of a trajectory and all samples which result in a terminal state are removed from the data set.

4.1.2 Gaussian Process Models

The Gaussian process models for the transition dynamics are trained using data sets of the form $\mathcal{D} = \{(\boldsymbol{s_i}, \boldsymbol{a_i}, \boldsymbol{s'_i}) \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mid i \in [N]\}$ of pairs of states and actions and their

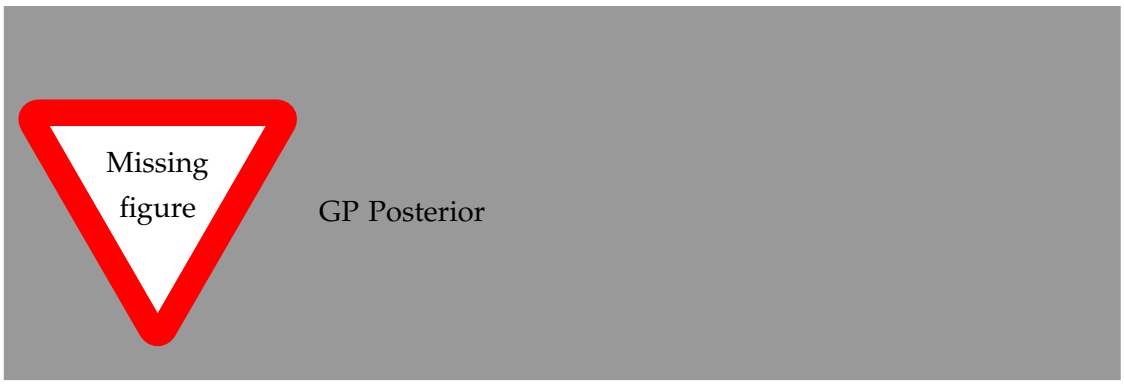


Figure 4.3: GP posterior

corresponding following state $s'_i = \mathbf{f}_{\text{bicycle}}(s_i, a_i)$. Since the transition dynamics of the bicycle benchmark are deterministic, these observations have no probabilistic element and they are not noisy. The model f of the transition dynamics is a compact statistical representation of this collected knowledge and is to be used to predict successive states of unobserved combinations of states and actions (s_*, a_*).

Besides predicting a concrete following state, the model should provide a measure about the uncertainty of its predictions. Since there is no randomness in the dynamics themselves, this uncertainty comes from the imperfect information about the true system dynamics and is dependent on the location of both the training data and the required predictions. If a query is made to the model in a part of the state space in which it has not seen many observations, the model should express its uncertainty and not assume that its best guess is close to the truth.

The Gaussian processes presented in section 3.2 represent a distribution over all plausible transition dynamics given a data set. In figure figure 4.3, the x-axis represents pairs of states and actions while the y-axis represents the successive state. Since the observations are noise-free, the GP is completely certain about predicting them and, since it assumed a smooth RBF-prior, it is also confident about predicting states closed to the observed data. Between the data points, uncertainties are higher since there are many different models which are plausible. Gaussian processes are called *non-degenerate*, since for predictions far away from the training set, the predicted uncertainty does not converge to zero. In contrast, for parts of the input space without any knowledge, the GP falls back to the prior assumptions about uncertainties and the mean function. Given a large enough data set which is spread out through the complete state and action space, the model becomes more and more confident about its predictions and

converges towards the true transition dynamics.

Gaussian processes as presented in this thesis can only model functions with univariate output. Approximating successive states requires multivariate predictions however. While there do exist extensions of Gaussian processes for multidimensional output [Raso6], a common solution is to train D separate GPs, one for every dimension in the state space $\mathbb{R}^D$. While this requires longer training time, it allows choosing a different set of hyperparameters for every dimension. Since the training set does not contain transitions which result in terminal states, the transition models do not know about them. The signature of the function represented by the transition model is $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, where $\mathcal{S} = \mathbb{R}^D$ and $\mathcal{A} = R^k$. Similar to the absence of terminal states, the transition models are also not aware of the rectangular boundaries of the state space described in table 2.1, which means that it is possible for the transition models to predict illegal states, which also have to be handled separately.

All models are trained using the squared exponential kernel presented in definition 10. The bicycle benchmark represents a physical system, which makes smoothness of the transition function a natural assumption. The RBF kernel is the standard choice in Gaussian process regression when no special knowledge about the shape of the transition function is available.

maybe mention other kernels? Is there anything to say about them?

Opposed to learning successive states directly, the training targets for the dth dimension are the differences to the current state given by

$$\Delta s_{i,d} := \mathbf{f}_{\text{bicycle}}(\boldsymbol{s_i}, \boldsymbol{a_i})_d - s_{i,d} = s'_{i,d} - s_{i,d},$$

where $i \in [N]$ and $d \in [D]$. This can be advantageous since differences vary less than the original function [Dei10]. Learning differences can also introduce independences in the data, since predicting the change in position of the bicycle only depends on the direction of movement but not on the previous position. Having learned models for the differences, the mean and the variance of the Gaussian posterior state distribution $\mathrm{p}(f_d(\boldsymbol{s}_*, \boldsymbol{a}_*))$ is given by

$$\begin{aligned}\mathbb{E}[f_d(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] &= s_{*,d} + \mathbb{E}[\Delta s_{*,d} \mid \boldsymbol{s}_*, \boldsymbol{a}_*],\\ \operatorname{var}[f_d(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] &= \operatorname{var}[\Delta s_{*,d} \mid \boldsymbol{s}_*, \boldsymbol{a}_*],\end{aligned}$$

respectively, since the prior state is considered constant and non-probabilistic. Uncertainties in the predictions only originate from the amount of confidence expressed by the models for the differences. The values of the expected value and variances are calculated according to lemma 11.

Since the different Gaussian processes are trained independently of each other and their training sets only contain their respective output dimension, their predictions are conditionally independent given the input. With the predictive distribution for the single dimension being Gaussian, the joint predictive state distribution is also Gaussian with a diagonal covariance matrix and is given by

$$\begin{aligned}
\mathrm{p}(f(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*) &= \mathcal{N}\big(f(\boldsymbol{s}_*, \boldsymbol{a}_*) \,\big|\, \boldsymbol{\mu}_f, \boldsymbol{\Sigma}_f\big), \text{ where} \\
\boldsymbol{\mu}_f &= \begin{pmatrix} \mathbb{E}[f_1(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] \\ \vdots \\ \mathbb{E}[f_D(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] \end{pmatrix} \\
\boldsymbol{\Sigma}_f &= \operatorname{diag}\left(\operatorname{var}[f_1(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*], \cdots, \operatorname{var}[f_D(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*]\right).
\end{aligned}$$

This diagonal covariance matrix illustrates the implicit independence assumption of the different output dimensions introduced by training one model per output dimension. While this assumption is not true in most cases, it can be used as an approximation and generally yields good results.

The state of the bicycle system is given by a vector $(\theta, \dot{\theta}, \omega, \dot{\omega}, x, y, \psi)$ composed of the internal dynamics of the bicycle and its position and orientation in euclidean space. During simulation with the transition model, the coordinates were transformed to polar coordinates given by

$$\begin{aligned}
\varphi(x, y) &:= \operatorname{atan2}(y, x) \\
r(x, y) &:= \sqrt{x^2 + y^2},
\end{aligned}$$

where atan2 is the arctangent function with two arguments. Polar coordinates uniquely represent a two-dimensional point by its angle to the x-axis and its distance to the origin. This representation both increases model performance and simplifies calculating the bicycle's relative position to the goal in the origin.

Additionally, representing both φ and ψ as numbers between $-\pi$ and π leads to a loss of information. While two angles with absolute value close to π but opposite signs are close together on a circle, their representations have a large euclidean distance. A Gaussian Process using the RBF-Kernel cannot recognize their similarity. In this case, it is possible to choose a specialized periodic variant of the squared exponential kernel which recognizes periodicity. Equivalently, an angle can be represented as a complex number on the unit circle, replacing it by its sine and cosine. Therefore, the internal representation of a bicycle state in the simulation is given by a vector

$$\boldsymbol{s} = (\theta, \dot{\theta}, \omega, \dot{\omega}, \varphi, r, \psi) \in \mathbb{R}^7$$

which is transformed to

$$\hat{\boldsymbol{s}} = (\theta, \dot{\theta}, \omega, \dot{\omega}, \sin\varphi, \cos\varphi, r, \sin\psi, \cos\psi) \in \mathbb{R}^9$$

when presented to the GPs. The transition model consists of seven Gaussian processes, each with nine-dimensional input.

The models are implemented in Python using Titsias's sparse variational GP regression implemented in *GPy* [GPy12] and trained using expectation maximization as presented in section 3.2.4. The optimization of the likelihood function is calculated using scaled conjugated gradients with multiple restarts to avoid local minima. The performance of the transition models is highly dependent on the size of the training set N and the number of inducing inputs M. For N smaller than 35000, the performance of the transition models for long-term predictions was not good enough to allow PSO-P to succeed for any of the approaches presented below. Conversely, for large N and M larger than 250, the models are good enough such that PSO-P finds perfect solutions for all approaches. The experiments in this thesis focus on choices for N and M which are inbetween these extremes and where information about the model uncertainties can be used to improve performance. The next section presents the classic approach of long-term predictions without the use of uncertainty information, which is used as a baseline for comparison for the following techniques.

4.2 Predictions without Uncertainties

- Terminal states

4.2.1 Deterministic Bicycle Reward Function

Definition 15 (Deterministic Bicycle Reward Function)
Given the set $\mathcal{S}^+$ of possible states of the bicycle benchmark, the *deterministic bicycle reward function* is defined as

$$\mathbf{R}_{\text{bicycle}} : \begin{cases} \mathcal{S}^+ \to \mathbb{R} \\ \quad s \mapsto \begin{cases} 2 & \text{if } s \in \mathcal{T}_{\text{goal}} \\ 0 & \text{if } s \in \mathcal{T}_{\text{fallen}} \\ 1 - \frac{|\psi_s - \varphi_s|}{\pi} & \text{otherwise} \end{cases} \end{cases}$$

where ψ_s and φ_s denote the respective angles in state s. It assigns constant reward for terminal states and reward inversely proportional to the angle between the bicycle's heading and the goal otherwise.

4.2.2 Long-Term predictions

4.2.3 Results and Problems

4.3 Predictions with One-Step Uncertainties

4.3.1 Probabilistic reward function

4.3.2 Long-Term predictions

4.3.3 Results and Problems

4.4 Predictions with Multi-Step Uncertainties

4.4.1 Linearization

4.4.2 Truncation of Gaussians

4.4.3 Results and Problems

Chapter 5

Conclusion

5.1 Discussion of the Approaches

5.2 Possible Improvements

Appendix A

Lists of Figures, Tables and Algorithms

List of Figures

List of Tables

List of Algorithms

Appendix B

Bibliography

[Åst71] Karl J. Åström. *Introduction to Stochastic Control Theory*. Elsevier, Feb. 27, 1971. 318 pp. ISBN: 9780080955797.

[BCF10] Eric Brochu, Vlad M. Cora, and Nando de Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *arXiv:1012.2599 [cs]* (Dec. 12, 2010). arXiv: 1012.2599. URL: http://arxiv.org/abs/1012.2599 (visited on 02/01/2016).

[DA] David Duvenaud and Ryan P. Adams. "Black-Box Stochastic Variational Inference in Five Lines of Python". In: (). URL: http://people.seas.harvard.edu/~dduvenaud/papers/blackbox.pdf (visited on 04/03/2016).

[Dam15] Andreas Damianou. "Deep Gaussian processes and variational propagation of uncertainty". PhD thesis. University of Sheffield, 2015. URL: http://etheses.whiterose.ac.uk/id/eprint/9968 (visited on 02/01/2016).

[Dei10] Marc Peter Deisenroth. "Efficient Reinforcement Learning using Gaussian Processes". PhD thesis. KIT Scientific Publishing, 2010. URL: http://www.cs.washington.edu/research/projects/aiweb/media/papers/tmppqidj5 (visited on 04/19/2016).

[DFR15] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian processes for data-efficient learning in robotics and control". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 37.2 (2015), pp. 408–423. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6654139 (visited on 02/01/2016).

[DR11] Marc Deisenroth and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472. URL: http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Deisenroth_323.pdf (visited on 02/01/2016).

[DRF] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning". In: (). URL: http://core.ac.uk/download/pdf/241164.pdf (visited on 02/01/2016).

[Eng06] Andries P. Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006. URL: http://dl.acm.org/citation.cfm?id=1199518 (visited on 02/01/2016).

[ES00] Russ C. Eberhart and Yuhui Shi. "Comparing inertia weights and constriction factors in particle swarm optimization". In: *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. Vol. 1. IEEE, 2000, pp. 84–88. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=870279 (visited on 05/13/2016).

[GPy12] GPy. *GPy: A Gaussian process framework in python*. GitHub. 2012. URL: https://github.com/SheffieldML/GPy (visited on 05/20/2016).

[Hei+16] Daniel Hein et al. "Reinforcement Learning with Particle Swarm Optimization Policy (PSO-P) in Continuous State and Actionspaces". In: 7.3 (July 2016).

[Kut01] Wilhelm Kutta. "Beitrag zur näherungweisen Integration totaler Differentialgleichungen". In: (1901). URL: http://www.citeulike.org/group/1448/article/813805 (visited on 05/20/2016).

[McK10] Wes McKinney. "Data Structures for Statistical Computing in Python". In: Proceedings of the 9th Python in Science Conference. 2010, pp. 51–56. URL: https://conference.scipy.org/proceedings/scipy2010/mckinney.html (visited on 05/20/2016).

[MR16] Rowan McAllister and Carl Edward Rasmussen. "Data-Efficient Reinforcement Learning in Continuous-State POMDPs". In: *arXiv preprint* (2016). DOI: arXiv:1602.02523. URL: http://arxiv.org/abs/1602.02523 (visited on 02/28/2016).

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Aug. 24, 2012. 1098 pp. ISBN: 9780262018029.

[P+08] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. "The matrix cookbook". In: *Technical University of Denmark* 7 (2008), p. 15. URL: http://www.cim.mcgill.ca/~dudek/417/Papers/matrixOperations.pdf (visited on 02/01/2016).

[Pre07] William H. Press. *Numerical Recipes: The Art of Scientific Computing*. 3rd Edition. Cambridge University Press, Sept. 6, 2007. 1195 pp. ISBN: 9780521880688.

[RA98] Jette Randløv and Preben Alstrøm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." In: *ICML*. Vol. 98. Citeseer, 1998, pp. 463–471. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.3038&rep=rep1&type=pdf (visited on 04/15/2016).

[Ras06] Carl Edward Rasmussen. "Gaussian processes for machine learning". In: (2006). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.3414 (visited on 02/01/2016).

[RN10] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010. 1153 pp. ISBN: 9780136042594.

[Ros95] Guido Rossum. *Python Reference Manual*. Amsterdam, The Netherlands, The Netherlands: CWI (Centre for Mathematics and Computer Science), 1995.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. 1998. URL: https://books.google.de/books?hl=de&lr=&id=CAFR6IBF4xYC (visited on 02/01/2016).

[SG05] Edward Snelson and Zoubin Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: *Advances in neural information processing systems*. 2005, pp. 1257–1264. URL: http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2005_543.pdf (visited on 02/01/2016).

[Sne07] Edward Lloyd Snelson. "Flexible and efficient Gaussian process models for machine learning". PhD thesis. Citeseer, 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.4041&rep=rep1&type=pdf (visited on 02/01/2016).

[SS02] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Jan. 2002. 658 pp. ISBN: 9780262194754.

[Tit09] Michalis K. Titsias. "Variational learning of inducing variables in sparse Gaussian processes". In: *International Conference on Artificial Intelligence and Statistics*. 2009, pp. 567–574. URL: http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS09_Titsias.pdf (visited on 02/01/2016).

[TL10] Michalis K. Titsias and Neil D. Lawrence. "Bayesian Gaussian process latent variable model". In: *International Conference on Artificial Intelligence and Statistics*. 2010, pp. 844–851. URL: http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_TitsiasL10.pdf (visited on 02/01/2016).

[Wat+15] Manuel Watter et al. "Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2746–2754. URL: http://papers.nips.cc/paper/5964-embed-to-control-a-locally-linear-latent-dynamics-model-for-control-from-raw-images.pdf (visited on 04/03/2016).

[WCV11] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering* 13.2 (Mar. 2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37.