15. June 2016

Master's Thesis in Informatics

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

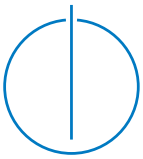
DEPARTMENT OF INFORMATICS
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

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

Acknowledgments

I would like to thank my advisor Prof. Dr.-Ing. Thomas Runkler for giving me the opportunity of creating my master's thesis at Siemens and for numerous suggestions about worthwhile directions for research. I am thankful to my supervisors Dr. Clemens Otte and Prof. Dr. Carl Henrik Ek for their excellent supervision, for their guidance and their help in understanding Bayesian reasoning. I am very appreciative of the great support of Dr. Steffen Udluft and Dr. Alexander Hentschel, who were always open to questions and who shared their expertise. Thanks to all of you for the numerous productive meetings and discussions and for your valuable advice.

I would also like to thank my office mates Stefan Depeweg and Daniel Hein for stimulating discussions and for their support. I am grateful for the positive and friendly environment created by the whole research group at Siemens.

I sincerely thank my family for their caring support throughout the years and above all I want to thank Hannah for her endless support, love and patience.

Abstract

In reinforcement learning, an agent has to learn how to make decisions in an unknown environment in order to maximize a numerical reward. In model-based reinforcement learning, the experience gained via interaction is represented as a transition model which can be used to simulate the system's future behaviour. This thesis is concerned with reducing the model bias introduced by choosing actions which are optimal with respect to an imperfect model. Instead of relying on a single deterministic model, gathered knowledge is represented using Gaussian processes which encode a probability distribution over all plausible transition models. By averaging over all of them, the expected long-term reward is calculated, which explicitly incorporates model uncertainties into long-term planning. A controller is formulated by applying Particle Swarm Optimization to this expected reward, directly choosing appropriate actions. Besides formally introducing these tools, this thesis investigates their effectiveness on a benchmark problem with the task of learning how to balance and navigate a bicycle. Thereby, multiple approaches of incorporating uncertainties are described and compared to the classic technique of deterministic predictions.

Zusammenfassung

Im Reinforcement Learning ist es die Aufgabe eines Agenten zu lernen, welche Entscheidungen in einer unbekannten Umgebung eine numerische Belohnung maximieren. Im modellbasierten Reinforcement Learning wird die durch Interaktion erworbene Erfahrung in einem Transitionsmodell repräsentiert, mit dem das zukünftige Verhalten des Systems simuliert werden kann. Diese Arbeit beschäftigt sich damit, den systematischen Fehler zu reduzieren, der durch Entscheidungsfindung auf Basis eines imperfekten Modells entsteht. Anstatt eines einzelnen deterministischen Modells wird Wissen mit Hilfe von Gaußprozessen dargestellt, die eine Wahrscheinlichkeitsverteilung über alle plausiblen Transitionsmodelle darstellen. Unter Berücksichtigung aller dieser Modelle wird die erwartete Belohnung über mehrere Zeitschritte errechnet, wodurch Modellunsicherheiten explizit in die Planung integriert werden. Durch Anwendung von Partikelschwarmoptimierung auf dem erwarteten Reward wird eine Entscheidungsstrategie formuliert, die Aktionen direkt mit Hilfe der Modelle wählt. Neben der formalen Beschreibung dieser Werkzeuge untersucht diese Arbeit ihre Effektivität anhand eines Beispielproblems mit der Aufgabe, ein Fahrrad zu balancieren und zu navigieren. Dabei werden mehrere Ansätze beschrieben, wie Unsicherheiten in die Planung integriert werden können, und mit dem klassischen Ansatz deterministischer Vorhersagen verglichen.

Contents

Chapter 1

Introduction

Machine learning can be understood as a combination of artificial intelligence and modern statistics. It is concerned with the development of algorithms which allow computers to find structure in data and extract relationships within it to represent it compactly. This allows the computer to generalize the observations presented and predict the distribution of data in unknown regions without being explicitly programmed. One branch of machine learning is reinforcement learning (RL). In RL, the task is to learn how to control a system by interaction and to find a strategy to achieve high-level goals. In contrast to other kinds of machine learning, reinforcement learning usually assumes that neither prior knowledge about the behaviour of the system nor an expert teacher are available. Instead, an uninformed agent starts off by taking suboptimal actions in order to explore and learn about the system. Having built a good understanding of the system, the agent can start exploiting this knowledge in order to achieve its goals.

Reinforcement learning is well-suited to solve problems which are easy to measure and generate data about but hard to describe and solve mathematically. These include both tasks in artificial intelligence traditionally solved by the development of specialized algorithms and problems in optimal control theory, where a dynamic physical system should be influenced in order to minimize long-term costs. In 1996, the chess computer Deep Blue was the first artificial intelligence to defeat a world champion in chess [Hsu02]. This victory was achieved using tree search and a mostly hand-crafted evaluation function able to judge the current board in a chess game. In contrast, the game of Go remained out of reach for artificial intelligences for a long time, since formulating an equivalent evaluation function is a hard task. In 2016, AlphaGo was the first AI to defeat a professional Go player [Sil+16]. Instead of relying on expert knowledge, AlphaGo was trained using reinforcement learning techniques. Starting off with information gathered from professional matches, AlphaGo gathered experience about Go by playing against itself and developed its strategies independently.

Dynamic systems in control theory are assumed to be described by a set of known differential equations. Optimal controllers are obtained by analyzing their algebraic structure and deriving solutions to the optimization problem of minimizing a cost function. While such controllers can be shown to be provably correct given the differential equations, these dynamics are often idealized or simplified. For sufficiently complicated systems, they cannot be formulated at all. In contrast, reinforcement learning does not require intricate knowledge about the dynamics but rather relies on interaction to identify its behaviour. In recent years, RL has successfully been used to control cars in autonomous driving [Kol+10] and to control complex industrial systems such as wind or gas turbines [Sch+07] whose behaviour cannot be described analytically in full detail.

Besides its roots in AI and control, reinforcement learning has historically been studied in the context of psychology and animal learning [SB98]. Trial and error is the most natural form of learning for humans and animals. An infant learning how to move has no explicit teacher but rather interacts with the environment and observes its responses. This way, the infant can obtain knowledge both about the world around them and their own body. Humans continue to learn by trial and error throughout their lives. When learning a new skill such as riding a unicycle, humans explore their task through play in order to get a feeling for the system to be controlled. A human might be careful at first in order to avoid accidents, but after enough training, their confidence increases and more complicated maneuvers become possible.

This thesis is concerned with enabling computational agents to develop a measure of confidence about their understanding of a system. The combined experience an agent has gathered via interaction can be represented in a model of the world. Since there is only a limited amount of observations available, such a model is always imperfect and there may be multiple plausible explanations of the observations. Classical deterministic approaches of representing such a model have to decide on one explanation of the data and must trust it to always be correct. During decision-making, an agent may be forced to generalize and make predictions about the system without having observed appropriate data. If this generalization is wrong, the agent makes decisions based on wrong information, which introduces a bias.

Instead of trusting one single model, more reliable information could be obtained by considering the predictions of all plausible models and combining their results. The distribution of predictions of the different models yields a measure of uncertainty the agent has about the future development of the system. This uncertainty can be used during planning to avoid actions whose impact on the system is unknown and enables the agent to choose a conservative strategy to avoid risk.

In chapter 2, this thesis presents the bicycle benchmark, a simulation of a dynamic system. In this system, the agent takes the place of a cyclist and has to solve the task of both balancing a bicycle and navigating it towards a goal. This system is used throughout the thesis as an example of a reinforcement learning problem of optimal control. Chapter 3 introduces a mathematical formulation of the general RL problem and presents both Gaussian processes and the Particle Swarm Optimization-Policy (PSO-Policy). Gaussian processes are a Bayesian framework which can be used to represent probability distributions over functions and are used in this thesis to represent the knowledge of an agent about the system to be controlled. Based on this representation, the PSO-Policy allows the formulation of a controller. Having established the main tools used in this thesis, chapter 4 describes how they can be applied to the bicycle benchmark. It first introduces a deterministic approach to solving the benchmark and then discusses how information about uncertainties can be used to improve the agent's strategy.

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 handlebars 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 bicycle's 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 figures 2.1 and 2.2. 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 2.2b and the moment of the bicycle and cyclist combined I_{BC} were estimated by the original definition [RA98] as

$$I_{\mathrm{dc}} := \quad M_d r^2 \tag{2.1}$$

$$I_{\mathrm{dv}} := \frac{3}{2} M_d r^2 \tag{2.2}$$

$$I_{\mathrm{dl}} := \frac{1}{2} M_d r^2 \tag{2.3}$$

$$I_{\mathrm{BC}} := \frac{13}{3} M_c h^2 + M_p (h + d_{\mathrm{CM}})^2. \tag{2.4}$$

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 towards. The second one are effects based on the conservation of angular momentum introduced via a cross-term dependent on $\dot{\theta}$. The centrifugal force resulting from 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

$$\varphi := \omega + \arctan\left(\frac{d}{h}\right). \tag{2.5}$$

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

$$\begin{aligned} \ddot{\omega} := \frac{1}{I_{\mathrm{BC}}} \Bigg(& \sin\varphi \cdot Mgh \\ & - \cos\varphi \cdot \left(I_{\mathrm{dc}} \dot{\sigma} \cdot \dot{\theta} + \operatorname{sgn}(\theta) \cdot v^2 \left(\frac{M_d r}{r_f} + \frac{M_d r}{r_b} + \frac{Mh}{r_{\mathrm{CM}}} \right) \right) \Bigg). \end{aligned} \tag{2.6}$$

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

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	$-\pi/6$ to $\pi/6$	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 to 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	$\dot{\sigma} = v/r$

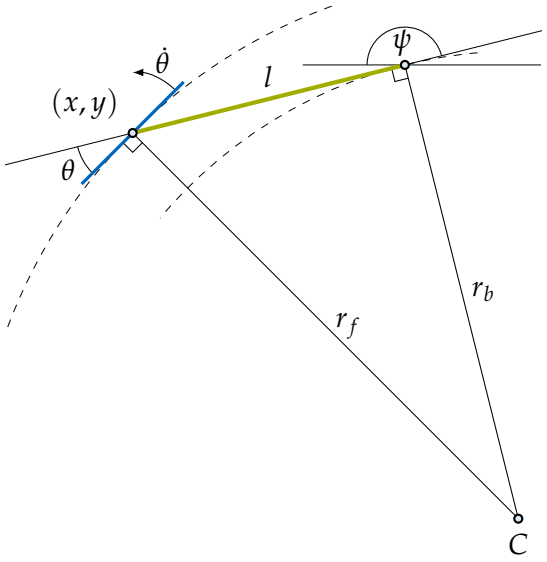
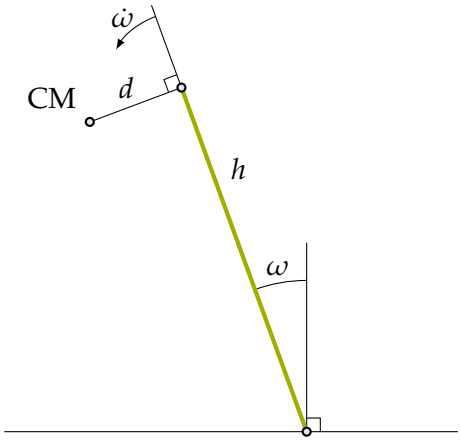
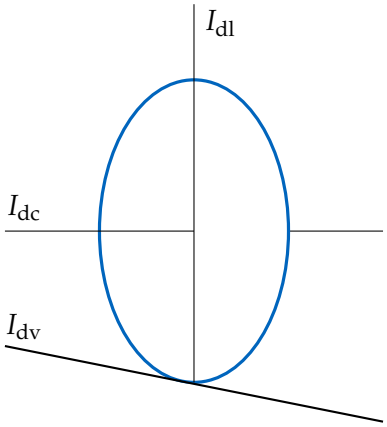


Figure 2.1: The bicycle system as seen from above. The position of the bicycle's green frame is defined by the x and y coordinates of the blue front tyre and the orientation of the frame of length l is measured by the angle ψ. Both the front and back tyre move along a dashed circular path with a shared center point C but different radii r_f and r_b. The radius of the front tyre is influenced by the steering angle θ.



(a) The bicycle system as seen from behind.



(b) The moments of inertia of a tyre.

Figure 2.2: Figure 2.2a shows the bicycle when viewed from behind. The angle ω describes the leaning of the bicycle. The center of mass is defined by the green bicycle's height h and the leaning distance d. Figure 2.2b shows the axes of the moments of inertia of the blue tyre. The axis I_{dc} describes the tyre's rotation due to movement of the bicycle. The axes I_{dv} and I_{dl} describe the tyre's rotation due to leaning and steering respectively.

T applied to the handlebars and the conservation of angular momentum introduced via a cross-term dependent on $\dot{\omega}$ and is calculated as

$$\ddot{\theta} := \frac{T - I_{\mathrm{dv}}\dot{\sigma}\dot{\omega}}{I_{\mathrm{dl}}}. \tag{2.7}$$

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 the x-axis. If ψ is equal to zero, the back tyre is located at a distance of l looking along the positive x-axis from the point (x, y) and a positive ψ denotes a counter-clockwise rotation.

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

$$r_f = \frac{l}{|\sin\theta|}, \tag{2.8}$$

$$r_b = \frac{l}{|\tan\theta|}, \tag{2.9}$$

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 around the common center point. The bike's angular velocity can be derived from the constant speed of the back tyre and is given by

$$v_\circ = \frac{v}{r_b}. \tag{2.10}$$

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:

$$\dot{\psi} = \operatorname{sgn}(\theta) \cdot v_\circ \tag{2.11}$$

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

$$\dot{x} = v_\circ \cdot r_f \cdot \cos(\psi + \theta) \tag{2.12}$$

$$\dot{y} = v_\circ \cdot r_f \cdot \sin(\psi + \theta) \tag{2.13}$$

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 works with a classical Runge-Kutta-Scheme as described in *Numerical Recipes 3rd Edition* [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.

Let Δ denote the function which maps a pair of state $\boldsymbol{s}$ and action $\boldsymbol{a}$ to its derivatives given by

$$\Delta(\boldsymbol{s}, \boldsymbol{a}) = \Delta\left(\left(\theta, \dot{\theta}, \omega, \dot{\omega}, x, y, \psi\right), \boldsymbol{a}\right) \coloneqq \left(\dot{\theta}, \ddot{\theta}, \dot{\omega}, \ddot{\omega}, \dot{x}, \dot{y}, \dot{\psi}\right). \tag{2.14}$$

This function can be calculated using equations (2.6), (2.7) and (2.11) to (2.13). Given a state $\boldsymbol{s_t}$ at time t and an action $\boldsymbol{a_t}$ which should be applied constantly for a time step τ, an approximation to the state $\boldsymbol{s_{t+\tau}}$ can be calculated using the classical Runge-Kutta-Scheme given by

$$\boldsymbol{s_{t+\tau}} = \boldsymbol{s_t} + \frac{\tau}{6}\left(\boldsymbol{k_1} + 2 \cdot \boldsymbol{k_2} + 2 \cdot \boldsymbol{k_3} + \boldsymbol{k_4}\right), \tag{2.15}$$

where the intermediate constants $\boldsymbol{k_1}$ to $\boldsymbol{k_4}$ are calculated as

$$\begin{aligned} \boldsymbol{k_1} &= \Delta(\boldsymbol{s_t}, \boldsymbol{a_t}) \\ \boldsymbol{k_2} &= \Delta(\boldsymbol{s_t} + \frac{\tau}{2} \cdot \boldsymbol{k_1}, \boldsymbol{a_t}) \\ \boldsymbol{k_3} &= \Delta(\boldsymbol{s_t} + \frac{\tau}{2} \cdot \boldsymbol{k_2}, \boldsymbol{a_t}) \\ \boldsymbol{k_4} &= \Delta(\boldsymbol{s_t} + \tau \cdot \boldsymbol{k_3}, \boldsymbol{a_t}). \end{aligned} \tag{2.16}$$

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 agent 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 agent. 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.

Using the bicycle benchmark as an example, the next chapter introduces reinforcement learning, a general mathematical description of the problem of learning how to control a dynamic system. This thesis focuses on model-based reinforcement learning, 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 system is an established benchmark 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 accumulated predictions.

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 teach an 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 navigating to the goal position. 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 learning task 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. 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 $\mathbb{N}$.

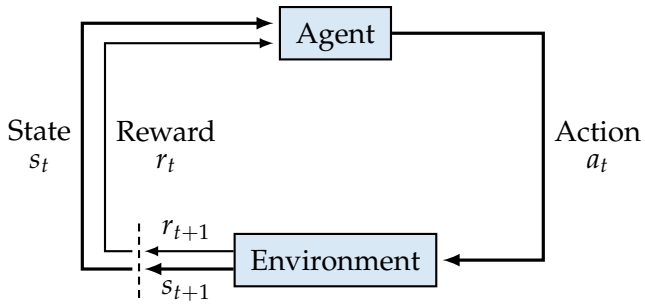


Figure 3.1: The interaction between an agent and its environment in reinforcement learning happens at discrete time steps. At every time step t, the agent observes the current state of the environment s_t and decides on an action a_t which should be performed. Based on this action, the environment evolves and is observed to be at the state s_{t+1} at the next time step. Additionally, the agent receives immediate feedback in form of the numeric reward r_{t+1}.

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 space of all possible states. Based on this information, the agent has to decide which action $a_t \in \mathcal{A}$ to perform. The space of all possible actions $\mathcal{A}$ is 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

$$\pi \colon \mathcal{S} \to \mathcal{A} \tag{3.1}$$

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 state s_{t+1} is sampled from the *transition dynamics*. 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} 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

$$\mathbf{f}\colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+) \tag{3.2}$$

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 together with the state and actions spaces, 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 called the *reward* r_{t+1}. The reward is a quality assigned to the state transition from s_t to s_{t+1} using the action a_t. The higher the reward, the better the state transition is considered to be. This measure is independent of the future or past development of the system. In this thesis, the reward for time step $t+1$ is considered to be independent of the state s_t and the action a_t and only depends on s_{t+1}. 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

$$\mathbf{R}\colon \mathcal{S}^+ \to \mathbb{R}. \tag{3.3}$$

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

$$\mathrm{p}(\boldsymbol{s_{t+1}}) = \int \mathrm{p}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi)\, \mathrm{p}(\boldsymbol{s_t})\, \mathrm{d}\boldsymbol{s_t} \tag{3.4}$$

where

$$\mathrm{p}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi) = \begin{cases} \delta(\boldsymbol{s_{t+1}} - \boldsymbol{s_t}) & \text{if } \boldsymbol{s_t} \in \mathcal{T} \\ \mathbf{f}(\boldsymbol{s_{t+1}} \mid \boldsymbol{s_t}, \pi(\boldsymbol{s_t})) & \text{otherwise.} \end{cases} \tag{3.5}$$

Here, δ denotes the Dirac-delta-distribution as defined in [Mur12, p. 39] 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 an 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 weight 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 value 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 accumulated reward of a state and is given by

$$\mathbf{V}^\pi \colon \begin{cases} \mathcal{S}^+ \to \mathbb{R} \\ \quad s \mapsto \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t\, \mathbf{R}(\boldsymbol{s_t}) \,\middle|\, \mathbf{f}, \pi, \boldsymbol{s_0} = s \right]. \end{cases} \tag{3.6}$$

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

Assume now a known distribution of possible start states $p(\boldsymbol{s_0})$. 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 $p(\boldsymbol{s_0})$ defines a fully observable Markov decision process (MDP) [SB98; Mur12].

The objective of the reinforcement learning problem 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

$$\begin{aligned} \pi^* &\in \operatorname*{argmax}_{\pi} \mathbb{E}_{s_0}[V^{\pi}(\boldsymbol{s_0})] \\ &= \operatorname*{argmax}_{\pi} \int V^{\pi}(\boldsymbol{s_0})\, p(\boldsymbol{s_0})\, d\boldsymbol{s_0}. \end{aligned} \tag{3.7}$$

There is a multitude of different approaches for learning 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 (3.7).

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 by exploiting the model. 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 exploration of a system like a gas turbine 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 original 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 $\mathbf{f}\colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+)$ is 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 lives in an infinite dimensional function space, 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 $\boldsymbol{\eta}$ 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 $\boldsymbol{\eta}^*$ 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 $\mathrm{p}(\boldsymbol{\eta}^*)$ 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 $\mathrm{p}(y_* \mid \boldsymbol{x}_*, \boldsymbol{\eta}^*)$, 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 account for all real-world factors, 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 modeling a distribution of parameter values, a Bayesian non-parametric model is concerned with finding a distribution $\mathrm{p}(f^*)$ of probable functions which represents the belief 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 $\mathrm{p}(y_* \mid \boldsymbol{x}_*, f^*)$ taking all functions in the distribution $\mathrm{p}(f^*)$ 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 reviewed 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 $\boldsymbol{x}$ [Gau09]. 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 $\mathrm{cov}[\boldsymbol{x_i}, \boldsymbol{x_j}] = \boldsymbol{\Sigma}_{ij}$.

Modeling 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

$$\boldsymbol{X}\colon \begin{cases} T \times \Omega \to Y \\ (t, \omega) \mapsto \boldsymbol{X_t}(\omega) \end{cases} \tag{3.8}$$

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 $f\colon A \to B$, 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*

$$\begin{aligned} \mathcal{K}(a, a') &:= \mathbb{E}[(f(a) - \mu_f(a))(f(a') - \mu_f(a'))] \\ &= \operatorname{cov}[f(a), f(a')] \\ &= \operatorname{cov}[\boldsymbol{X}_a, \boldsymbol{X}_{a'}] \end{aligned} \tag{3.9}$$

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 otherwise, 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 $\mathcal{K}\colon A^2 \to \mathbb{R}$ 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

$$\mathcal{K}(X, Y) = \boldsymbol{K_{XY}} := \Big(\mathcal{K}(x_i, y_j) \Big)_{\substack{i \in [n], \\ j \in [m]}} \tag{3.10}$$

is called the *Gram matrix* of $\mathcal{K}$ with respect to X and Y [SSo2]. The notation $[n]$ describes the set $\{1, \ldots, n\}$ of integers.

In order for the Gram matrix to be a valid covariance matrix $\mathbf{\Sigma}$ 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

$$\mathcal{K}\colon A^2 \to \mathbb{R} \tag{3.11}$$

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 [SS02].

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

$$\mathcal{K}_{\text{linear}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \langle \boldsymbol{x}, \boldsymbol{y} \rangle\,. \tag{3.12}$$

Consider a function $f\colon \mathbb{R} \to \mathbb{R}$ 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

$$\begin{pmatrix} \boldsymbol{f}_x \\ \boldsymbol{f}_y \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathcal{K}(x,x) & \mathcal{K}(x,y) \\ \mathcal{K}(y,x) & \mathcal{K}(y,y) \end{bmatrix}\right) \tag{3.13}$$

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

$$\begin{aligned} \varrho[\boldsymbol{f}_x, \boldsymbol{f}_y] &= \frac{\operatorname{cov}[\boldsymbol{f}_x, \boldsymbol{f}_y]}{\sqrt{\operatorname{var}[\boldsymbol{f}_x]}\sqrt{\operatorname{var}[\boldsymbol{f}_y]}} \\ &= \frac{\mathcal{K}(x,y)}{\sqrt{\mathcal{K}(x,x)}\sqrt{\mathcal{K}(y,y)}} = \frac{xy}{\sqrt{x^2}\sqrt{y^2}} \in \{-1, 1\}. \end{aligned} \tag{3.14}$$

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.

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.

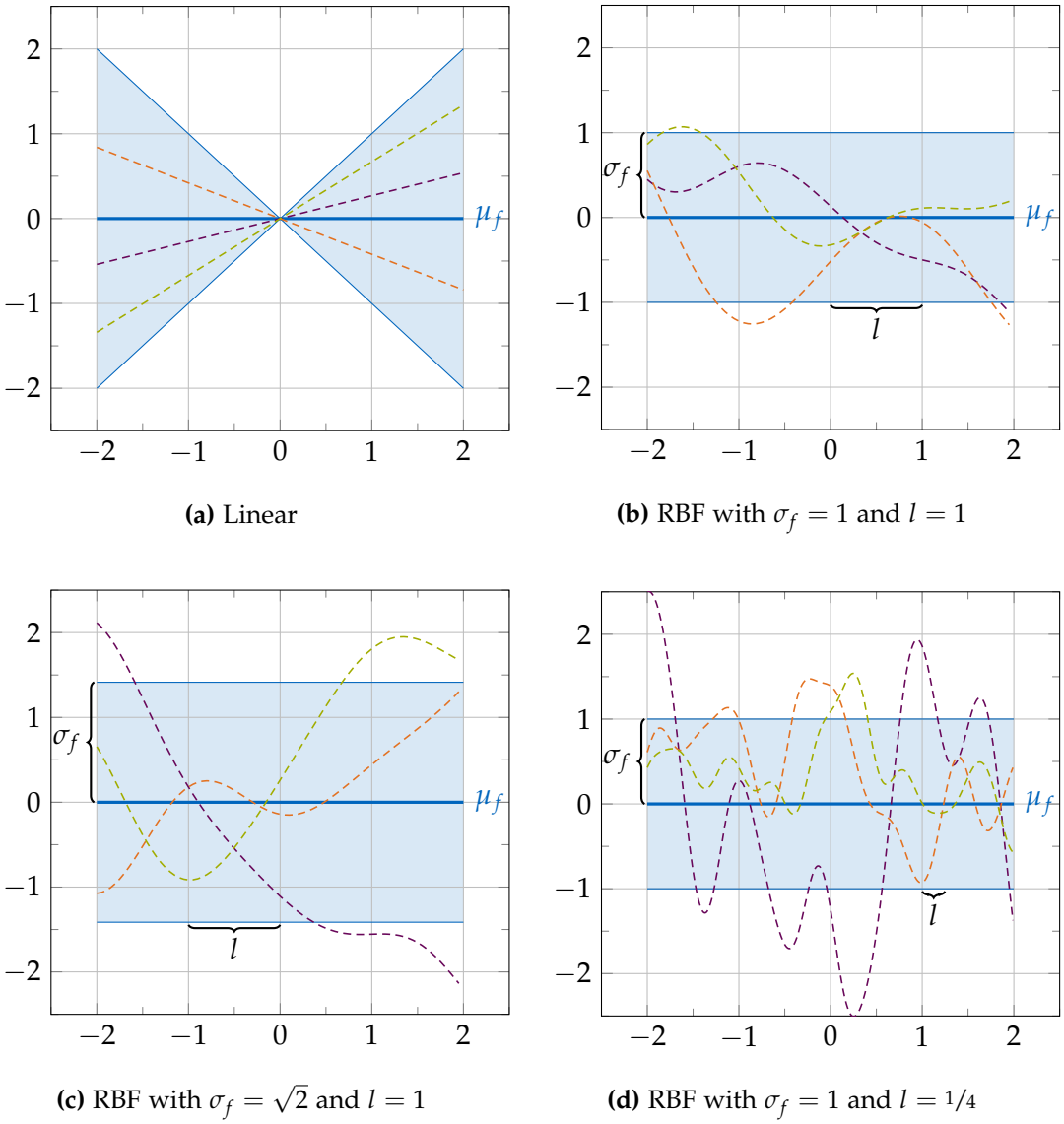
Definition 10 (Squared Exponential Kernel)
For a finite dimensional euclidean vector space $\mathbb{R}^d$, the *squared exponential kernel* (or *RBF kernel*) is defined as

$$\mathcal{K}_{\text{SE}}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sigma_f^2 \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{\Lambda}^{-1}(\boldsymbol{x} - \boldsymbol{y})\right). \tag{3.15}$$

The parameter $\sigma_f^2 \in \mathbb{R}_{>0}$ is called the *signal variance* and $\boldsymbol{\Lambda} = \operatorname{diag}(l_1^2, \ldots, l_d^2)$ 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 belief about this function. In order to separate the two, they are called *hyperparameters*. The vector of all hyperparameters in a model is called $\boldsymbol{\theta}$.



(a) Linear

(b) RBF with $\sigma_f = 1$ and $l = 1$

(c) RBF with $\sigma_f = \sqrt{2}$ and $l = 1$

(d) RBF with $\sigma_f = 1$ and $l = 1/4$

Figure 3.2: Since the mean function μ_f is assumed to be constant zero, the kernel specifies the prior assumptions about the function. A dashed sample function can be drawn by sampling a multivariate Gaussian with the kernel's Gram matrix using a grid of discrete sampling positions. While samples from the linear kernel are always hyperplanes, the RBF kernel describes arbitrary smooth functions. The hyperparameters l and σ_f of the kernel describe the assumed dynamic range in x and y directions respectively.

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 the linear kernel and 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 $\mathcal{N}(\mathbf{0}, \boldsymbol{K}_*)$ 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 $f \sim \mathcal{GP}(\mathbf{0}, \mathcal{K})$ in order to obtain a predictive posterior. The N data points observed are denoted as $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ with $\boldsymbol{y} = f(\boldsymbol{X}) + \mathcal{N}(\mathbf{0}, \sigma_n^2 \mathrm{I})$ 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

$$\begin{aligned} \mathrm{p}(\boldsymbol{y} \mid f, \boldsymbol{X}) = \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{f}) &= \prod_{i=1}^{N} \mathrm{p}(y_i \mid f_i) \\ &= \prod_{i=1}^{N} \mathcal{N}(y_i \mid f_i, \sigma_n^2) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma_n^2 \mathrm{I}) \end{aligned} \tag{3.16}$$

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

$$\mathrm{p}(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{f} \mid \mathbf{0}, \boldsymbol{K}_N) \tag{3.17}$$

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

$$\begin{aligned} \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{f})\, \mathrm{p}(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{f} \\ &= \int \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma_n^2 \mathrm{I})\, \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{0}, \boldsymbol{K}_N)\, \mathrm{d}\boldsymbol{f} \\ &= \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K}_N + \sigma_n^2 \mathrm{I}). \end{aligned} \tag{3.18}$$

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

$$\mathrm{p}\left(\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{pmatrix} \,\middle|\, \boldsymbol{X}, \boldsymbol{X}_*, \boldsymbol{\theta} \right) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{pmatrix} \,\middle|\, \boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_N & \boldsymbol{K}_{N*} \\ \boldsymbol{K}_{*N} & \boldsymbol{K}_* \end{bmatrix} \right). \tag{3.19}$$

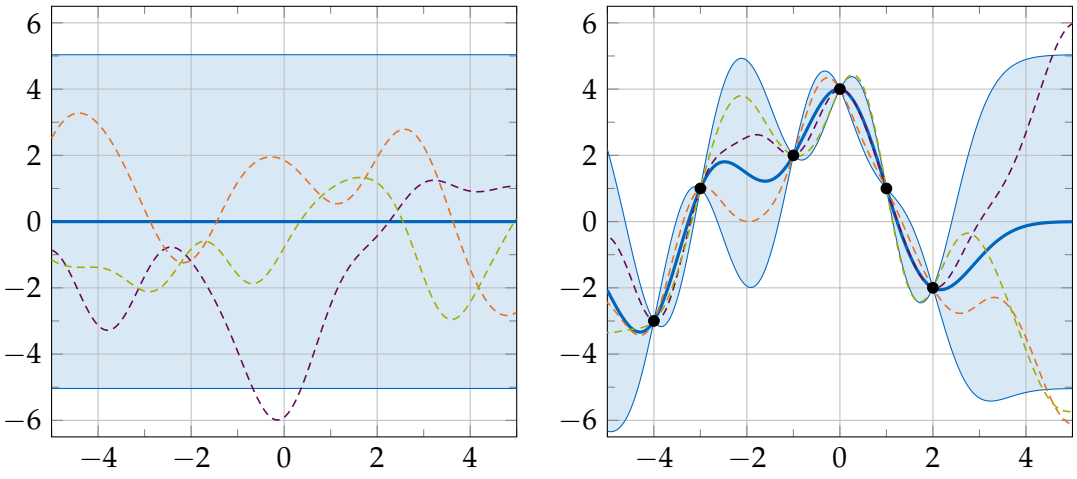
Adding the noise model to this distribution leads to the joint Gaussian of training outputs $\boldsymbol{y}$ and test outputs $\boldsymbol{f}_*$ which is given by

$$\mathrm{p}\left(\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{pmatrix} \,\middle|\, \boldsymbol{X}, \boldsymbol{X}_*, \boldsymbol{\theta} \right) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{pmatrix} \,\middle|\, \boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_N + \sigma_n^2 \mathrm{I} & \boldsymbol{K}_{N*} \\ \boldsymbol{K}_{*N} & \boldsymbol{K}_* \end{bmatrix} \right). \tag{3.20}$$

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 [P+08], yielding another Gaussian distribution $\mathrm{p}(\boldsymbol{f}_* \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_*)$.

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

$$\begin{aligned} \mathrm{p}(\boldsymbol{f}_* \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_*) &= \mathcal{N}(\boldsymbol{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \text{ where} \\ \boldsymbol{\mu}_* &= \boldsymbol{K}_{*N} \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I} \right)^{-1} \boldsymbol{y} \\ \boldsymbol{\Sigma}_* &= \boldsymbol{K}_* - \boldsymbol{K}_{*N} \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I} \right)^{-1} \boldsymbol{K}_{N*}. \end{aligned} \tag{3.21}$$



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

Figure 3.3: Figure 3.3a shows a GP prior with an RBF kernel. After observing the black data points, the mean function of the posterior GP in figure 3.3b is no longer constant zero. The dashed function samples of the posterior GP interpolate the data but can be different in-between. The shaded area represents the point wise mean plus and minus two times the standard deviation.

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

$$\begin{aligned} \mu_{\text{post}}(\boldsymbol{a}) &= \mathcal{K}(\boldsymbol{a}, \boldsymbol{X}) \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y} \\ \mathcal{K}_{\text{post}}(\boldsymbol{a}, \boldsymbol{b}) &= \mathcal{K}(\boldsymbol{a}, \boldsymbol{b}) - \mathcal{K}(\boldsymbol{a}, \boldsymbol{X}) \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \mathcal{K}(\boldsymbol{X}, \boldsymbol{b}). \end{aligned} \tag{3.22}$$

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 function value of a single test point is a weighted sum of N basis functions $\mu_* = \boldsymbol{K}_{*N}\boldsymbol{\beta}$ where $\boldsymbol{\beta} = \left(\boldsymbol{K}_N + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y}$ 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

$$\mathrm{p}(f) = \int \mathrm{p}(f \mid \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta} \tag{3.23}$$

$$\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}) = \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}. \tag{3.24}$$

Updating the belief 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:

$$\begin{aligned} \mathrm{p}(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}) &= \frac{\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})}{\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X})} \\ &= \frac{\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})}{\int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{\theta})\, \mathrm{d}\theta} \end{aligned} \tag{3.25}$$

The integration required in equation (3.24) is very hard in practice [Raso6], 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 $\mathrm{p}(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y})$ and does not require evaluation of the denominator since it is constant.

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

$$\begin{aligned} \mathrm{p}(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}) &\propto \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) \\ &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{f}, \end{aligned} \tag{3.26}$$

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

$$\begin{aligned} \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\theta})\, \mathrm{p}(\boldsymbol{f} \mid \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{f} \\ &= \int \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma_n^2 \mathrm{I}) \cdot \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{0}, \boldsymbol{K_N})\, \mathrm{d}\boldsymbol{f} \\ &= \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K_N} + \sigma_n^2 \mathrm{I}) \end{aligned} \tag{3.27}$$

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

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= -\log \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) \\ &= \frac{1}{2} \boldsymbol{y}^\mathsf{T} \left(\boldsymbol{K_N} + \sigma_n^2 \mathrm{I}\right)^{-1} \boldsymbol{y} + \frac{1}{2} \log |\boldsymbol{K_N} + \sigma_n^2 \mathrm{I}| + \frac{N}{2} \log(2\pi). \end{aligned} \tag{3.28}$$

The estimation of hyperparameters is the solution of the optimization problem

$$\boldsymbol{\theta}^* \in \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{3.29}$$

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 [Ras06]. 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 [Sne07]. 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 N training samples, 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 the cubic 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 to the true latent values of the function function $f \sim \mathcal{GP}(\boldsymbol{0}, \mathcal{K})$. 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

$$\mathrm{p}(\boldsymbol{f_*} \mid \boldsymbol{X_*}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{K_{*M}} \boldsymbol{K_M^{-1}} \bar{\boldsymbol{f}}, \boldsymbol{K_*} - \boldsymbol{K_{*M}} \boldsymbol{K_M^{-1}} \boldsymbol{K_{M*}}) \tag{3.30}$$

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

$$\begin{aligned}
\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) &= \prod_{i=1}^{N} \mathrm{p}(y_n \mid \boldsymbol{x_n}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) \\
&= \prod_{i=1}^{N} \mathcal{N}\left(y_n \,\middle|\, \boldsymbol{K_{nM}} \boldsymbol{K_M^{-1}} \bar{\boldsymbol{f}}, \boldsymbol{K_n} - \boldsymbol{K_{nM}} \boldsymbol{K_M^{-1}} \boldsymbol{K_{Mn}} + \sigma_n^2\right) \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \boldsymbol{K_{NM}} \boldsymbol{K_M^{-1}} \bar{\boldsymbol{f}}, \operatorname{diag}\left(\boldsymbol{K_N} - \boldsymbol{K_{NM}} \boldsymbol{K_M^{-1}} \boldsymbol{K_{MN}}\right) + \sigma_n^2 \mathrm{I}\right) \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \boldsymbol{K_{NM}} \boldsymbol{K_M^{-1}} \bar{\boldsymbol{f}}, \operatorname{diag}\left(\boldsymbol{K_N} - \boldsymbol{Q_N}\right) + \sigma_n^2 \mathrm{I}\right)
\end{aligned} \tag{3.31}$$

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 (3.27). Assuming the pseudo targets to be distributed very similarly to the real data, a reasonable prior for them is given by

$$\mathrm{p}(\bar{\boldsymbol{f}} \mid \bar{\boldsymbol{X}}) = \mathcal{N}(\bar{\boldsymbol{f}} \mid \boldsymbol{0}, \boldsymbol{K_M}). \tag{3.32}$$

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

$$\begin{aligned}
\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \boldsymbol{\theta}) &= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) \, \mathrm{p}(\bar{\boldsymbol{f}} \mid \bar{\boldsymbol{X}}) \, \mathrm{d}\bar{\boldsymbol{f}} \\
&= \int \mathrm{p}(\boldsymbol{y} \mid \boldsymbol{X}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{f}}, \boldsymbol{\theta}) \, \mathcal{N}(\bar{\boldsymbol{f}} \mid \boldsymbol{0}, \boldsymbol{K_M}) \, \mathrm{d}\bar{\boldsymbol{f}} \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \boldsymbol{0}, \boldsymbol{K_{NM}} \boldsymbol{K_M^{-1}} \boldsymbol{K_M} \left(\boldsymbol{K_{NM}} \boldsymbol{K_M^{-1}}\right)^{\mathsf{T}} + \operatorname{diag}\left(\boldsymbol{K_N} - \boldsymbol{Q_N}\right) + \sigma_n^2 \mathrm{I}\right) \\
&= \mathcal{N}\left(\boldsymbol{y} \,\middle|\, \boldsymbol{0}, \boldsymbol{Q_N} + \operatorname{diag}\left(\boldsymbol{K_N} - \boldsymbol{Q_N}\right) + \sigma_n^2 \mathrm{I}\right).
\end{aligned} \tag{3.33}$$

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 (3.18). In this Gaussian process, the original kernel $\mathcal{K}$ is replaced by the kernel $\mathcal{K}_{\mathrm{SPGP}}$. With $\mathbb{I}$ denoting the indicator function, it is defined as

$$\begin{aligned}
\mathcal{Q}(\boldsymbol{a}, \boldsymbol{b}) &\coloneqq \boldsymbol{K_{aM}} \boldsymbol{K_M^{-1}} \boldsymbol{K_{Mb}} \\
\mathcal{K}_{\mathrm{SPGP}}(\boldsymbol{a}, \boldsymbol{b}) &\coloneqq \mathcal{Q}(\boldsymbol{a}, \boldsymbol{b}) + \mathbb{I}(\boldsymbol{a} = \boldsymbol{b}) \left(\mathcal{K}(\boldsymbol{a}, \boldsymbol{b}) - \mathcal{Q}(\boldsymbol{a}, \boldsymbol{b})\right).
\end{aligned} \tag{3.34}$$

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 $f \sim \mathcal{GP}(\mathbf{0}, \mathcal{K}_{\text{SPGP}})$, 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

$$\begin{aligned} \mathrm{p}(\boldsymbol{f}_* \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}, \bar{\boldsymbol{X}}) &= \mathcal{N}(\boldsymbol{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \text{ where} \\ \boldsymbol{\mu}_* &= \boldsymbol{Q}_{*N} \left(\boldsymbol{Q}_N + \operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2 \mathrm{I} \right)^{-1} \boldsymbol{y} \\ \boldsymbol{\Sigma}_* &= \boldsymbol{K}_* - \boldsymbol{Q}_{*N} \left(\boldsymbol{Q}_N + \operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2 \mathrm{I} \right)^{-1} \boldsymbol{Q}_{N*}. \end{aligned} \tag{3.35}$$

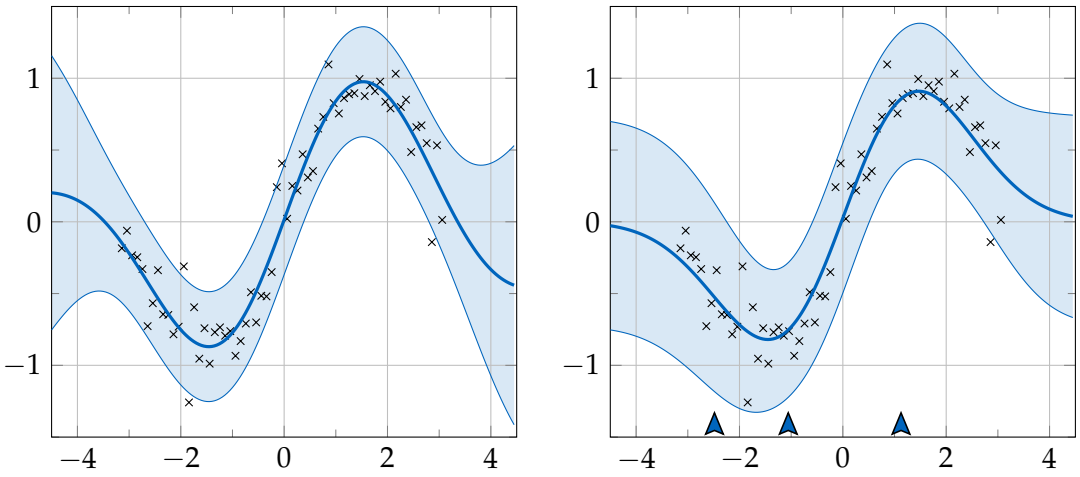
and $\boldsymbol{Q}_N \coloneqq \boldsymbol{K}_{NM} \boldsymbol{K}_M^{-1} \boldsymbol{K}_{MN}$.

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

$$\begin{aligned} \boldsymbol{\mu}_* &= \boldsymbol{K}_{*M} \boldsymbol{B}^{-1} \boldsymbol{K}_{MN} \left(\operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2 \mathrm{I} \right)^{-1} \boldsymbol{y} \\ \boldsymbol{\Sigma}_* &= \boldsymbol{K}_* - \boldsymbol{K}_{*M} \left(\boldsymbol{K}_M^{-1} - \boldsymbol{B}^{-1} \right) \boldsymbol{K}_{M*} \\ \boldsymbol{B} &= \boldsymbol{K}_M + \boldsymbol{K}_{MN} \left(\operatorname{diag}(\boldsymbol{K}_N - \boldsymbol{Q}_N) + \sigma_n^2 \mathrm{I} \right)^{-1} \boldsymbol{K}_{NM}, \end{aligned} \tag{3.36}$$

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



(a) Full GP **(b)** SPGP

Figure 3.4: The black crosses signify data sampled from a noisy sine function. Figure 3.4a shows a full GP trained on the complete data. Figure 3.4b shows an SPGP with pseudo inputs located at the dart positions. Since the pseudo function values are marginalized, only their x-coordinate is meaningful. Three pseudo inputs are enough to approximate the full GP with reasonable accuracy.

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 selecting pseudo inputs by optimization of the SPGP marginal likelihood, Titsias proposed a variational approach [Tito9] which optimizes a lower bound of the marginal likelihood of the original Gaussian process. This formulation has the property of minimizing the Kullback-Leibler divergence between the variational distribution and the exact posterior distribution of the latent function values of the full GP. Since this strategy of selecting hyperparameters leads to batter convergence and more robust results in practice, this variational SPGP is used to model transition dynamics within this thesis.

In order to solve the control problem of the bicycle benchmark, the next step after modeling 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 using 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].

In order to avoid these complexities, this thesis chooses a non-parametric approach. In contrast to using the expected long-term reward to find optimal choices for parameters of a policy, the *Particle Swarm Optimization-Policy* described by Hein et al. in [Hei+16] directly chooses optimal actions. When presented with a decision, PSO-P optimizes the *action-value-function* $\hat{\mathbf{V}}$ given by

$$\hat{\mathbf{V}}_{s_0}: \begin{cases} \mathcal{A}^T \to \mathbb{R} \\ (\boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}}) \mapsto \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t \, \mathbf{R}(\boldsymbol{s_t}) \,\middle|\, f, \boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}} \right] \end{cases} \tag{3.37}$$

where f denotes a model of the transition function, $\boldsymbol{s_0}$ denotes a state and γ is the discount factor. After finding an optimal vector of actions for all decisions in the time

horizon, PSO-P applies the first of these actions to the system. Avoiding a parametric representation allows PSO-P to find actions which, according to the models, are optimal for the specific situation the agent currently is in, instead of having to find a set of parameters which is good for all possible states. This, however, comes with the disadvantage of having to solve an optimization problem for every decision of the controller, which increases computational cost.

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

$$\pi_{\text{PSO-P}} : \begin{cases} \mathcal{S} \to \mathcal{A} \\ s \mapsto a_0^*, \text{ where } a^* \in \underset{a \in \mathcal{A}^T}{\operatorname{argmax}} \hat{\mathbf{V}}_s(a). \end{cases} \tag{3.38}$$

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 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.

This section first introduces *Particle Swarm Optimization*, a gradient-free heuristic used in PSO-P to tackle the non-convex optimization problem of finding the optimal action sequence. After the definition, different choices of parameters are discussed.

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 [Pre07] 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 $f\colon\ \mathcal{X} \to \mathbb{R}$, the problem is to find an $\boldsymbol{x}^* \in \mathcal{X}$ such that

$$\boldsymbol{x}^* \in \operatorname*{argmax}_{x \in \mathcal{X}} f(x). \tag{3.39}$$

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

$$\mathcal{X} \coloneqq \left\{ \boldsymbol{x} \in \mathbb{R}^d \;\middle|\; x_i^{\min} \leq x_i \leq x_i^{\max} \right\}. \tag{3.40}$$

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 particles are attracted to positions with good performance.

Definition 14 (PSO Instance)
An instance of the *particle swarm optimization (PSO)* scheme on the objective function $f\colon \mathbb{R}^d \to \mathbb{R}$ 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

$$\mathbf{x}\colon \mathcal{P} \times \mathcal{T} \to \mathbb{R}^d, \tag{3.41}$$

$$\mathbf{v}\colon \mathcal{P} \times \mathcal{T} \to \mathbb{R}^d, \tag{3.42}$$

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

$$\mathcal{N}\colon \mathcal{P} \to 2^{\mathcal{P}}. \tag{3.43}$$

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

$$\mathbf{x}_i(0), \mathbf{v}_i(0) \sim \boldsymbol{x}^{\min} + \mathbb{U}(0,1) \cdot \left(\boldsymbol{x}^{\max} - \boldsymbol{x}^{\min}\right), \tag{3.44}$$

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

$$\mathbf{x}_i(t+1) \coloneqq \mathbf{x}_i(t) + \mathbf{v}_i(t). \tag{3.45}$$

This operation can move particles out of the set of feasible points. Engelbrecht discusses multiple possible boundary conditions such as circular algebra or reflection [Eng06]. 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

$$\mathbf{y}_i(t) \coloneqq \mathbf{x}_i \left(\min \operatorname*{argmax}_{t' \in [t]} f(\mathbf{x}_i(t')) \right). \tag{3.46}$$

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

$$\begin{aligned} \hat{\mathbf{y}}_i(t) &\coloneqq \mathbf{y}_{n^*}(t), \text{ where} \\ n^* &\in \operatorname*{argmax}_{n \in \mathcal{N}(i)} f(\mathbf{y}_n(t)). \end{aligned} \tag{3.47}$$

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

$$\mathbf{v}_i(t+1) \coloneqq \underbrace{\omega \cdot \mathbf{v}_i(t)}_{\text{inertia}} + \underbrace{\gamma_c \cdot r_c \cdot (\mathbf{y}_i(t) - \mathbf{x}_i(t))}_{\text{cognitive component}} + \underbrace{\gamma_s \cdot r_s \cdot (\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t))}_{\text{social component}}. \tag{3.48}$$

The real constants ω, γ_c and γ_s weight 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 before the final iteration t_{final} and is given by

$$\boldsymbol{x}^* \in \operatorname*{argmax}_{p \in \mathcal{P}} f(\mathbf{y}_p(t_{\text{final}})). \tag{3.49}$$

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 collapses 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 some scenarios which are not ideal [Engo6].

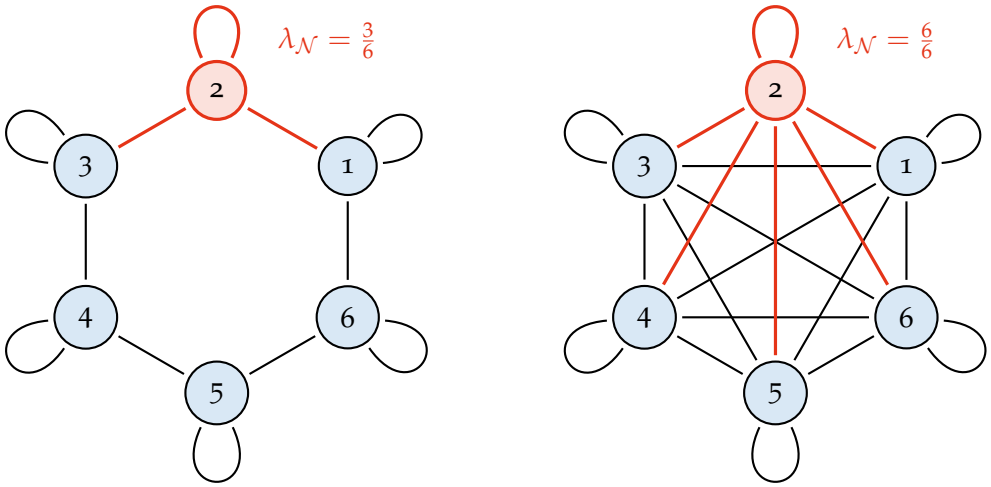
Since PSO cannot judge whether it has converged to the optimum, standard termination criteria for optimization have to be employed. Common problem-dependent choices are a minimum step size between successive best solutions, minimum improvement of the objective function between successive best solutions 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, good 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 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



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

Figure 3.5: Neighbourhood topologies define the social interaction between particles in a swarm. Figure 3.5a shows a ring topology with six particles, where every particle is connected to its neighbours and itself. Since every particle is connected to exactly three other particles, the average degree of connectivity $\lambda_{\mathcal{N}}$ is one half. The special case of the ring topology where every particle is connected to every other particle is called the star topology and is shown in figure 3.5b.

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. In this thesis, PSO is used to optimize in the space of all future actions in the time horizon. Given a time horizon of 15 and two actions per time step, this space is 30-dimensional. To ensure good coverage, 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 prematurely 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 best solution found by the swarm.

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 (3.48) 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 weighting 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 zero and one to allow the swarm to converge. Eberhart and Shi suggest setting ω to 0.7291 and both γ_c and γ_s to 1.49618 [ES00; Cle99].

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.

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

In the experiments of this thesis, the convergence behaviour of PSO was not problematic. 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 optimizing 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 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

Incorporating Uncertainty in Model-Based Reinforcement Learning

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 as they 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.

Modeling the transition dynamics allows the prediction of a state $\boldsymbol{s}_{t+1}$ given a concrete pair of state and action $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ 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 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 (3.37) 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 nonlinear transition model to accumulate the uncertainties of multiple predictions. Additionally, the reward function must be formulated in such a way that it is 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 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 process. 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 Models

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) at every time step, 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 evaluate 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 safety concern. It is therefore not possible to apply an on-line learning scheme to 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 by PSO-P.

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 [VD95] with NumPy [WCV11] and using the classical Runge-Kutta scheme [Kut01].

In their experiments, Randløv and Alstrøm [RA98] 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. The experiments in this thesis are based on the same time discretization of 0.01 seconds but for computational reasons only allow the actor to choose an action every ten time steps, keeping it constant for the time steps in between. This yields a controller frequency of 10 Hz. The choice of time discretization is the only free parameter in the transition dynamics for the bicycle system described in chapter 2. The resulting transition dynamics used for the interaction of the controller and the system are called $\mathbf{f}_{\text{bicycle}}$ below.

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 on a controller which can successfully balance the bicycle. Instead, this thesis chooses an uninformed controller which applies random actions to the system.

Algorithms 4.1 to 4.3 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. 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 about the distribution of starting states and also increases the mean lengths of the sampled trajectories when compared to more random starting states. As shown in figures 4.1 and 4.2a, an average trajectory in the data set is quite short, since random actions are not suitable to balance the bicycle.

Figure 4.2b 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

Algorithm 4.1 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 4.2 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 4.3 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
```

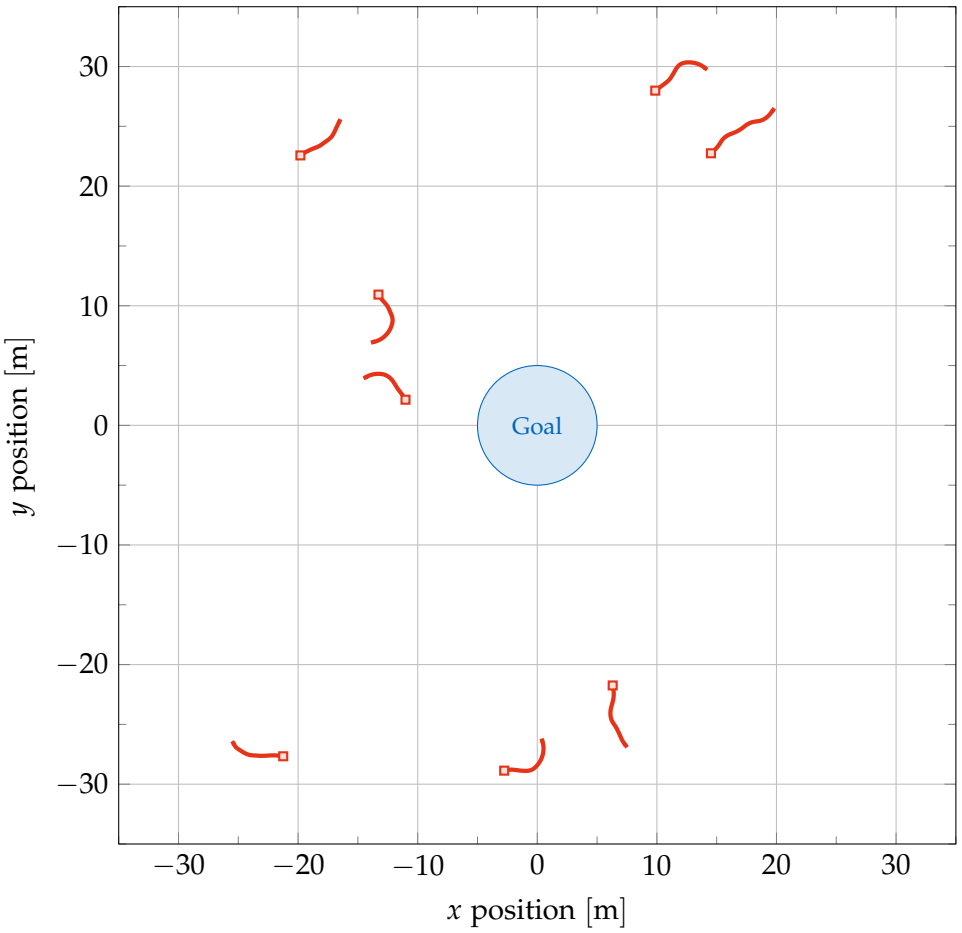
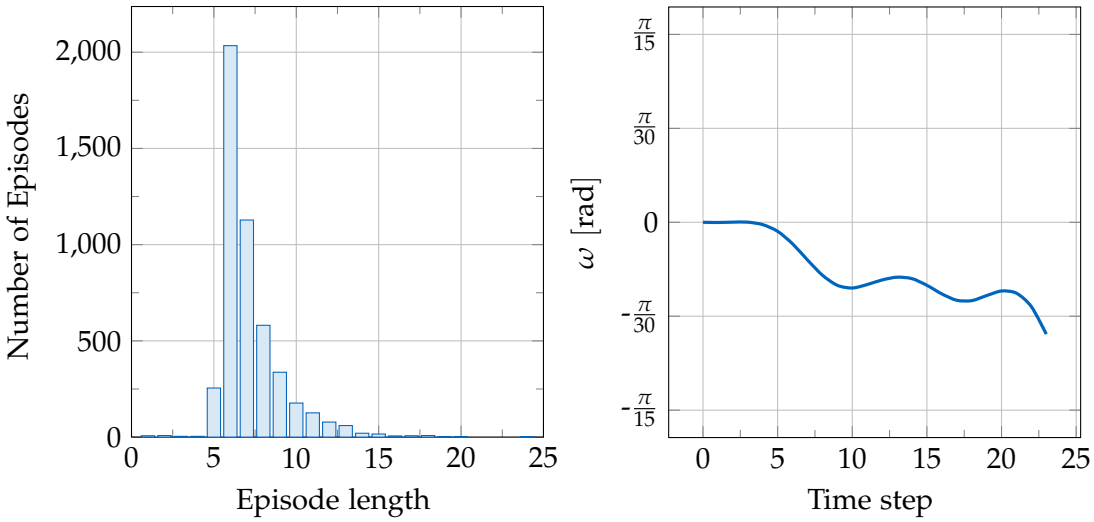


Figure 4.1: The x and y coordinates of the front tyre in representative episodes in the training set with marks at the starting states. Since random actions cannot successfully balance the bicycle, episodes are fairly short and end with the bicycle falling over.



(a) Histogram of episode lengths

(b) The variable ω for one episode

Figure 4.2: Figure 4.2a shows the distribution of episode lengths in a typical training set. Most trajectories are between five and ten time steps long, with a clear peak at six steps. Figure 4.2b shows the development of the angle ω for an exceptionally long episode. Even for long episodes, the bicycle quickly starts leaning towards one side and does not recover. The last value of ω is in the valid range of values, since the last transition which results in a terminal state is not part of the training set.

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 4.3. While those random samples add more coverage of the state space 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 normal 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.

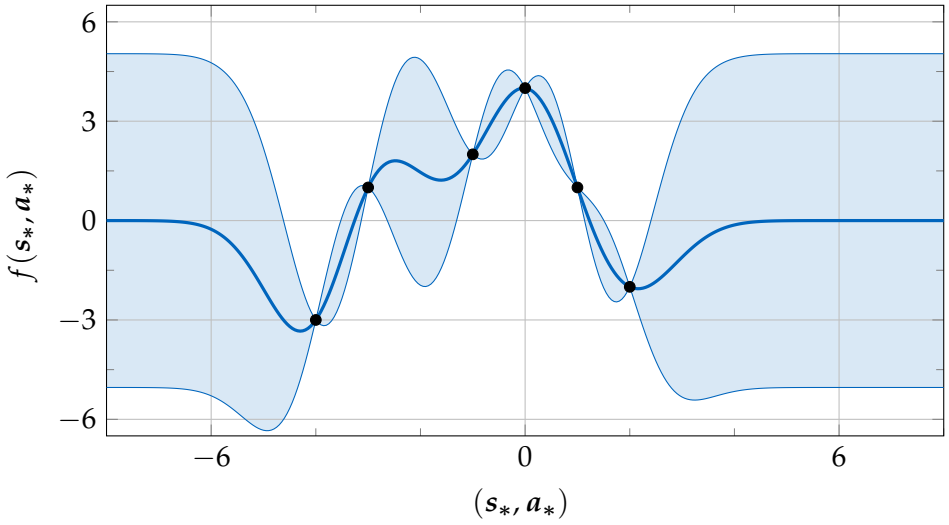


Figure 4.3: Since observations are completely certain, the transition models have no uncertainty at the black observations. The further away from observations a test point is, the higher the uncertainty becomes. Since Gaussian processes are non-degenerate, uncertainties do not converge to zero away from the observed data. Instead, for values with absolute value larger than six, the transition model falls back to the prior.

4.1.2 Gaussian Process Models

The Gaussian process models for the transition dynamics are trained using data sets of the form $\mathcal{D} = \{(s_i, a_i, s_i') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mid i \in [N]\}$ of pairs of states and actions and their corresponding following state. The following states $s_i' = \mathbf{f}_{\text{bicycle}}(s_i, a_i)$ are obtained form the transition dynamics. 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 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 close 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 more 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 the terminal conditions for trajectories. 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 a common choice in Gaussian process regression when no special knowledge about the shape of the transition function is available. Experiments with other kernels and combinations of kernels showed no difference in terms of mean squared error on a test set.

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

$$\Delta s_{i,d} := \mathbf{f}_{\text{bicycle}}(\boldsymbol{s_i}, \boldsymbol{a_i})_d - s_{i,d} = s'_{i,d} - s_{i,d}, \tag{4.1}$$

where $i \in [N]$ and $d \in [D]$. This can be advantageous since differences tend to vary less than the original function. 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 for the partial model for the d-th dimension $\mathrm{p}(f_d(\boldsymbol{s}_*, \boldsymbol{a}_*))$ is given by

$$\begin{aligned} \mathbb{E}[f_d(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] &= s_{*,d} + \mathbb{E}[\Delta s_{*,d} \mid \boldsymbol{s}_*, \boldsymbol{a}_*], \\ \mathrm{var}[f_d(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] &= \mathrm{var}[\Delta s_{*,d} \mid \boldsymbol{s}_*, \boldsymbol{a}_*], \end{aligned} \tag{4.2}$$

respectively, since the prior state is for now 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

$$\begin{aligned} \mathrm{p}(f(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*) &= \mathcal{N}\big(f(\boldsymbol{s}_*, \boldsymbol{a}_*) \,\big|\, \boldsymbol{\mu}_f, \boldsymbol{\Sigma}_f\big), \text{ where} \\ \boldsymbol{\mu}_f &= \begin{pmatrix} \mathbb{E}[f_1(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] \\ \vdots \\ \mathbb{E}[f_D(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*] \end{pmatrix} \\ \boldsymbol{\Sigma}_f &= \mathrm{diag}\left(\mathrm{var}[f_1(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*], \ldots, \mathrm{var}[f_D(\boldsymbol{s}_*, \boldsymbol{a}_*) \mid \boldsymbol{s}_*, \boldsymbol{a}_*]\right). \end{aligned} \tag{4.3}$$

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

$$\begin{aligned} \varphi(x, y) &\coloneqq \mathrm{atan2}(y, x), \\ r(x, y) &\coloneqq \sqrt{x^2 + y^2}, \end{aligned} \tag{4.4}$$

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 which is given by a vector

$$\boldsymbol{s} = (\theta, \dot{\theta}, \omega, \dot{\omega}, \varphi, r, \psi) \in \mathbb{R}^7 \tag{4.5}$$

is transformed to the *augmented state*

$$\hat{\boldsymbol{s}} = (\theta, \dot{\theta}, \omega, \dot{\omega}, \sin\varphi, \cos\varphi, r, \sin\psi, \cos\psi) \in \mathbb{R}^9 \tag{4.6}$$

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 is 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 between 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

The transition model trained on a predefined data set allows the prediction of a successive state distribution $p(s_1)$ given a deterministic pair of a state and an action

(s_0, a_0). To evaluate the action value function $\hat{\mathbf{V}}$, two extensions need to be made. Firstly, beyond specifying the goal, chapter 2 does not define a concrete reward function. This section introduces a variant of the reward function used by Randløv and Alstrøm in [RA98].

And secondly, for a time horizon T longer than one step into the future, the predictive state distributions $p(s_1)$ up to $p(s_T)$ are required. Since the GP dynamics model returns a Gaussian predictive distribution for all states beyond the starting state to account for the model uncertainty, all states beyond the starting state are no longer deterministic. In order to mimic a classic non-Bayesian model without a measure of uncertainty, the approach presented in this section discards this information and considers the maximum-a-posteriori estimation to be the deterministic prediction of the transition model. Having established the deterministic mode of evaluating the action value function, this section finally introduces the evaluation setup used in this thesis and discusses the results of applying this technique to the bicycle system.

4.2.1 Bicycle Reward Function

Solving the bicycle benchmark is a composite problem. An agent has to both learn to balance a bicycle and drive to the goal. Instead of having to solve the two tasks one after the other, they both have to be solved simultaneously, constantly switching between them. While an agent is in control of the bicycle's balance, it can try to drive towards the goal. If any action applied to the system leads to the danger of falling over however, the agent has to quickly change its focus towards preventing this.

Without expert knowledge available, the controller must learn this distinction autonomously, given the reward function. The most basic and uninformed reward function possible assigns positive reward for reaching the goal, a punishment (in the form of negative reward) for falling over and weights all other states equally somewhere between the two extremes. While this can be enough to teach the short-term task of avoiding to fall down, the agent has no incentive of driving towards the goal. For most situations, the goal cannot be reached within the time-horizon of one PSO-P instance. In this setting, PSO-P would optimize towards a trajectory for which the chance of falling down is minimal. This trajectory is a circle with large radius [RA98].

To give the controller a motivation of reaching the goal, it has to receive some hint about the correct direction to drive. Encoding this information in the reward function goes against the assumption of the complete absence of expert knowledge. If it is too detailed, it introduces the risk of significantly simplifying the learning problem or

pushing the agent towards a policy which is only locally optimal. This reduction of the hard problem of finding the goal to a series of easier problems of driving in the correct direction and then going straight is called *shaping* [SB98; RA98].

The hint towards the goal encoded in the reward function should be a term which represents information which is local in the sense that its value can change considerably within the time horizon. The most simple term to consider is a punishment based on the current distance to the goal. This formulation is problematic however, since the agent should not care about the actual distance rather than the change of distance with respect to the previous state, which cannot be expressed in the reward. Using the distance itself, an increase in reward would express movement in the correct direction. However, for any non-linear punishment, the amount of increase is dependent on the current position in the input space and can lead to numerical problems if it gets too small. If it were linear, the punishment might at some point be larger than the punishment for falling. At this point, the agent's correct choice would be to fall down as quickly as possible.

In order to avoid these problems, the reward function used in this thesis is based on the current angle between the frame of the bicycle and the direction towards the goal. Since the goal's position is at the origin of the coordinate system, this angle can be calculated as the difference of the current rotation of the bicycle ψ and the angular component of its polar coordinates in space φ. It is in the agent's interest to minimize this angle, since if it zero, the agent is heading towards the goal on the shortest route possible. As the bicycle moves at a constant speed, a successful trajectory which locally minimizes this angle is also a globally optimal trajectory, since it results in the shortest path possible.

Instead of a linear reward based on the difference, this thesis chooses a quadratic saturating reward function as proposed by Deisenroth et al. in [DFR15]. The quadratic saturating reward function is a Gaussian density function normalized to a value of one at its maximum. It behaves locally quadratic around this maximum and shows exponential drop off towards zero for large deviations from the mean. Figure 4.4 compares the saturating reward function to a linear one. The Gaussian reward is more forgiving for small errors around zero but punishes large deviations more, which is a good behaviour for the bicycle benchmark since it is often good enough for the bicycle to only approximately head towards the goal. If the bicycle has reached the goal, the agent receives a constant reward of two. This is double the reward which can be obtained for any state which is not in the goal. Similarly, the reward for any state where the bicycle has fallen over is zero, which is less than for any non-terminal state in the benchmark.

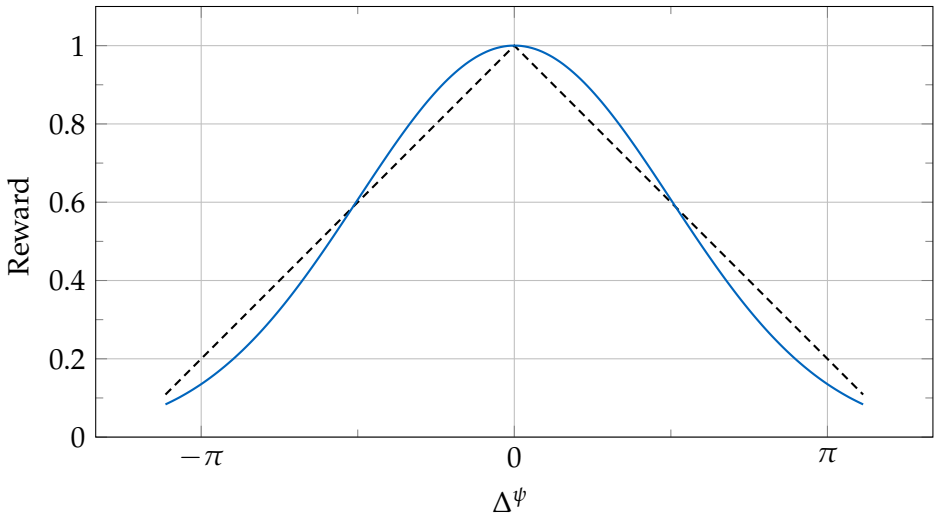


Figure 4.4: The blue quadratic saturating reward function is a Gaussian probability density which has been renormalized to a maximum value of one. When compared to the dashed piecewise linear reward function, the saturating reward is more forgiving for small errors in the angle difference Δ^{Ψ}, but punishes large deviations more.

Definition 15 (Bicycle Reward Function)
Given the set $\mathcal{S}^+$ of possible states of the bicycle benchmark and the sets $\mathcal{T}_{\text{goal}}$ and $\mathcal{T}_{\text{fallen}}$ of all terminal states where the bicycle has reached the goal or fallen respectively, the *bicycle reward function* is defined as

$$\mathbf{R}_{\text{bicycle}} : \begin{cases} \mathcal{S}^+ \to \mathbb{R} \\ \quad s \mapsto \begin{cases} 2 & \text{if } s \in \mathcal{T}_{\text{goal}} \\ 0 & \text{if } s \in \mathcal{T}_{\text{fallen}} \\ \sqrt{2\pi\sigma^2_{\text{angle}}}\,\mathcal{N}(\Delta^{\psi}_s \mid 0, \sigma^2_{\text{angle}}) & \text{otherwise} \end{cases} \end{cases} \tag{4.7}$$

where $\Delta^{\psi}_s = |\psi_s - \varphi_s| \in (-\pi, \pi]$ denotes the difference between the bicycle's heading and the goal given a state s and σ^2_{angle} is a positive real constant. The Gaussian density function $\mathcal{N}$ is normalized to a maximum value of 1.

The agent does not receive negative reward for falling down. Instead, the reward is zero. The agent is still punished when falling down though, since the episode has to end and the agent is not able to collect additional reward by driving towards the goal at later time steps. The width σ^2_{angle} of the saturating reward was chosen to be $(\pi/2)^2$, a

compromise between an emphasized peak at zero and sufficiently large derivatives at the extreme values $-\pi$ and π.

4.2.2 Long-Term predictions

The models for the bicycle transition dynamics are Gaussian processes with the assumption that their underlying space is $\mathbb{R}^7$, a complete vector space. The set $\mathcal{S}$ of valid non-terminal states in the bicycle benchmark is an axis-parallel cuboid however, whose axis-wise boundaries are defined by the vectors $\boldsymbol{s}^{\min}$ and $\boldsymbol{s}^{\max}$. The minimal and maximal values for every variable can be found in table 2.1. The transition models transform the euclidean coordinates x and y to the polar coordinates φ and r. Their intervals of valid values are given by the intervals $(-\pi, \pi]$ and $[0, 100]$ respectively.

A state of the bicycle benchmark is terminal if the bicycle has either reached the goal or if it has fallen over. The goal is defined as a circle around the origin with radius five. A bicycle has fallen if the absolute value of the angle ω is larger than $\pi/15$. If the bicycle is inside the goal area but has also fallen down, the state is defined to be successful. Using these constraints, the set of terminal states $\mathcal{T}$ and the set of non-terminal states $\mathcal{S}$ are defined as

$$\mathcal{T}_{\text{goal}} := \left\{ \boldsymbol{s} \in \mathbb{R}^7 \;\middle|\; r_{\boldsymbol{s}} \leq 5 \right\}, \tag{4.8}$$

$$\mathcal{T}_{\text{fallen}} := \left\{ \boldsymbol{s} \in \mathbb{R}^7 \;\middle|\; |\omega_{\boldsymbol{s}}| > \frac{\pi}{15} \right\} \setminus \mathcal{T}_{\text{goal}}, \tag{4.9}$$

$$\mathcal{S} := \left\{ \boldsymbol{s} \in \mathbb{R}^7 \;\middle|\; \forall i \colon \boldsymbol{s}_i \in \left[\boldsymbol{s}_i^{\min}, \boldsymbol{s}_i^{\max} \right] \right\} \setminus \mathcal{T}_{\text{goal}}, \tag{4.10}$$

respectively, where $\mathcal{T} := \mathcal{T}_{\text{goal}} \cup \mathcal{T}_{\text{fallen}}$. The set of all possible states in the bicycle benchmark is given by $\mathcal{S}^+ := \mathcal{S} \cup \mathcal{T}$. Note that $\mathcal{S}^+$ is not equal to $\mathbb{R}^7$. A state where ψ is greater than π or where θ is greater than $\pi/2$ is considered physically impossible. This is unknown to the transition models and it is possible for them to predict impossible states. Since these extreme states are rare, this thesis considers all physically impossible predictions to be predictions of the closest possible value. A prediction of $\pi/6 + \epsilon$ for θ is interpreted as a prediction of $\pi/6$.

Gaussian processes are Bayesian models which predict values by averaging over all plausible latent functions, resulting in a Gaussian belief about the prediction. In order to simulate a deterministic transition model, this first approach for long-term predictions discards this belief and only predicts a single successive state. The most plausible choice for this single successive state is the maximum-a-posteriori estimation given by

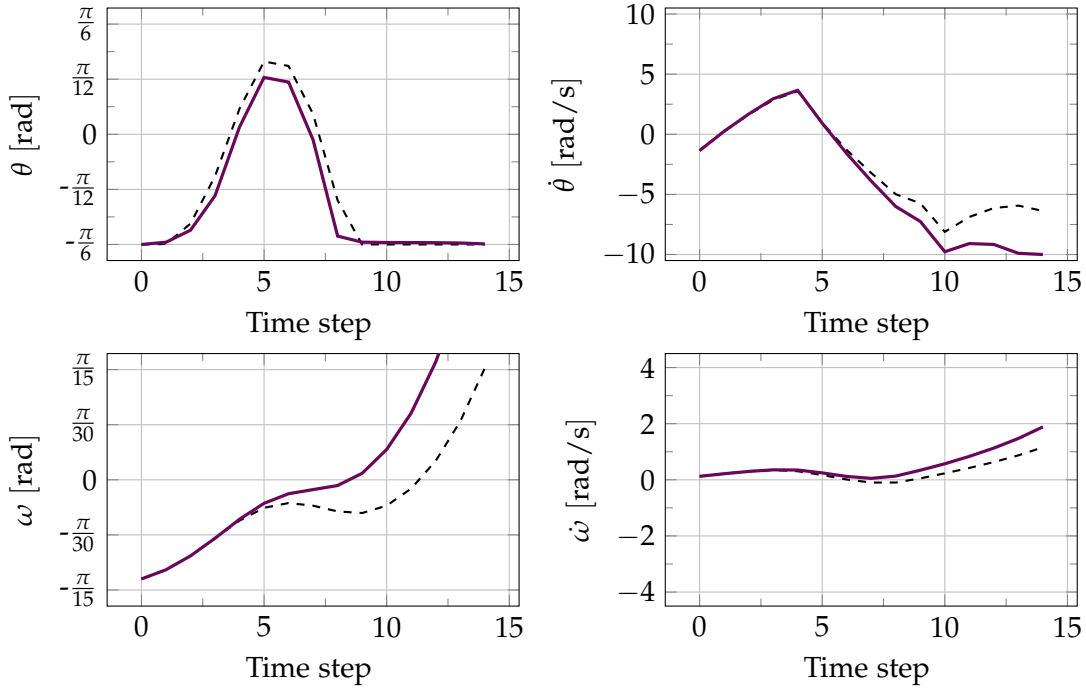


Figure 4.5: Colored long-term predictions for one time horizon using MAP estimates and starting from a deterministic state compared to the dashed simulation. After about half the time horizon, iterating MAP-prediction produces considerable errors.

the maximum of the posterior density function. With a deterministic state s_t and an action a_t, this MAP estimate of the transition model f is given by

$$\begin{aligned} \underset{s_{t+1} \in \mathbb{R}^7}{\operatorname{argmax}} \, p(s_{t+1} \mid s_t, a_t) &= \underset{s_{t+1} \in \mathbb{R}^7}{\operatorname{argmax}} \, f(s_{t+1} \mid s_t, a_t) \\ &= \underset{s_{t+1} \in \mathbb{R}^7}{\operatorname{argmax}} \, \mathcal{N}(s_{t+1} \mid \mu_{t+1}, \Sigma_{t+1}) \\ &= \{\mu_{t+1}\}. \end{aligned} \tag{4.11}$$

A Gaussian distribution's density function has a unique maximum at the mean.

The transition model is trained using observations in $\mathcal{S}$ and is unaware of terminal states. Since states at all time steps are deterministic, terminal states can easily be

modelled as fixed points in the function defined as

$$s_{t+1}^{\text{MAP}} : \begin{cases} \mathcal{S}^+ \times \mathcal{A} \to \mathcal{S}^+ \\ (s_t, a_t) \mapsto \begin{cases} s_t & \text{if } s_t \in \mathcal{T} \\ \mathbb{E}[f(s_t, a_t)] & \text{otherwise,} \end{cases} \end{cases} \tag{4.12}$$

where the expected value of $f(s_t, a_t)$ can directly be computed using lemma 11. Given a time horizon T, a starting state s_0 and a vector of actions $(a_0, \ldots, a_{T-1})$, this function can be used to calculate the intermediate states s_1^{MAP} to s_T^{MAP} using dynamic programming. Figure 4.5 compares these iterated MAP-predictions with the correct evolution of the bicycle system in an example. Only the angles θ and ω and their derivatives are shown in the picture, since they define the dynamics of the bicycle. The position and orientation of the bicycle are derived from those variables and do not play a role in long-term predictions.

With these intermediate states and the reward function $\mathbf{R}_{\text{bicycle}}$ from section 4.2.1, it is possible to evaluate the action value function $\hat{\mathbf{V}}$ defined in equation (3.37) and therefore apply PSO-P to the bicycle system. Using the MAP-estimates for intermediate states, the action value function can be calculated as

$$\begin{aligned} \hat{\mathbf{V}}_{s_0}^{\text{MAP}}(a_0, \ldots, a_{T-1}) &= \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t \, \mathbf{R}_{\text{bicycle}}(s_t) \;\middle|\; f, s_0, a_0, \ldots, a_{T-1} \right] \\ &= \sum_{t=1}^{T} \gamma^t \, \mathbb{E}\left[\mathbf{R}_{\text{bicycle}}(s_t) \mid f, s_0, a_0, \ldots, a_{T-1} \right] \\ &= \sum_{t=1}^{T} \gamma^t \, \mathbf{R}_{\text{bicycle}}(s_t^{\text{MAP}}), \end{aligned} \tag{4.13}$$

where $s_0^{\text{MAP}} := s_0$. Since all states are deterministic, all expected values collapse to applications of the reward function to the single deterministic state.

4.2.3 Evaluation Setup

Algorithm 4.4 describes the setup used to evaluate and compare the different implementations of the action value function used in this thesis. PSO-P directly optimizes the action value function in order choose actions to be applied to the bicycle system. Since the action value function is based on the dynamics models, this policy highly depends on the performance of the transition models. In order to reduce this bias in

Table 4.1: The parameters used for evaluation in this thesis.

Part	Parameter	Description	Values
Simulation	τ	Time Discretization	0.01 s
		Successive Steps	10
Evaluation	I	Number of data sets	46
	S	Number of trajectories	15
		Maximum trajectory length	120
PSO-P	T	Time horizon	15
	σ^2_{angle}	Reward width	$(\pi/2)^2$
GP models	$\boldsymbol{N}$	Number of training points	{60000, 70000}
	$\boldsymbol{M}$	Number of pseudo inputs	{20, 30, 40, 50, 75, 100, 150}
	$\mathcal{K}$	Kernel	RBF

the evaluations, the performance is measured based on multiple different transition models.

The behaviour of the transition models depends on the quality of the underlying data set, the number of input points used for training and the prior choices of the hyperparameters. The evaluation samples multiple data sets and then considers a number of combinations of training points N and pseudo inputs M as shown in table 4.1. The number of pseudo inputs determines the expressiveness of the model. Many pseudo inputs allow the transition models to describe the data with more detail while few pseudo inputs may force the model to generalize more.

Standard choices for the number of pseudo inputs are in the range from a few ten to several hundred inputs [Sne07], but the correct choice depends on the function to be learned and cannot easily be determined. The models in this thesis range from very simple models with 20 pseudo inputs to more powerful models with 150 pseudo inputs. The boundaries were chosen based on empirical studies. At some point below 20 pseudo inputs, the models collapse since they loose their expressive power. For a very high number of pseudo inputs, all techniques discussed in this thesis behave very similarly, since the predictions of the models become almost perfect. Similarly, the number of training points was chosen such that there is enough data to catch all relevant parts of the dynamics but not enough data for the models to be perfect.

These different choices of hyperparameters for the transition models are applied to a number of different data sets sampled according to algorithm 4.3. For every such data

Algorithm 4.4 Bicycle evaluation setup

Let $\hat{\mathbf{V}}$ denote an implementation of the action value function to be evaluated and $\boldsymbol{s}^{\max}$ the vector of maximum values for all state variables. All other constants are described in table 4.1 and algorithm 4.3.

```
for i ← 1, I do
    D_i ← SampleMixedBicycleDataSet(max N)
    for all (N, M) ∈ N × M do
        Train transition model f^(N,M) on D_i with kernel K
        for s ← 1, S do
            s_0^(s) ← SampleBicycleStartState() + N(0, diag(10^-4 · s^max))
            Observe trajectory τ_{i,s} from s_0^(s) using PSO-P on f^(N,M) and V̂
Evaluate V̂ using the trajectories {τ_{i,s} | i ∈ [I], s ∈ [S]}
```

set, transition models are trained using the different combinations of the numbers of training points and pseudo inputs. Based on every such model, multiple trajectories are generated using PSO-P and the action value function $\hat{\mathbf{V}}$ to be evaluated. A trajectory is created by sampling a starting state $\boldsymbol{s}_0$ and using PSO-P to choose an action $\boldsymbol{a}_0$ to be applied. Using this pair, the successive state $\boldsymbol{s}_1 = \mathbf{f}_{\text{bicycle}}(\boldsymbol{s}_0, \boldsymbol{a}_0)$ can be calculated using the simulation. This process is iterated until the bicycle reaches the goal, falls over or the trajectory becomes longer than the maximum trajectory length. The time horizon T used in PSO-P is chosen to be long enough to contain about a third of a rotation of the bicycle on a tight curve. This proofed to be a long enough horizon to solve the benchmark while still remaining computationally feasible.

The application of algorithm 4.4 leads to a large set of trajectories based on different data and different models. In order to compare different implementations of the action value function $\hat{\mathbf{V}}$, PSO-P is applied to the same starting states using the same models with the respective functions.

4.2.4 Results using MAP Predictions

Figure 4.6 shows an example of a successful trajectory generated using PSO-P on the maximum-a-posteriori action value function $\hat{\mathbf{V}}^{\text{MAP}}$ derived in equation (4.13). Since the PSO policy does not have to be trained, the first interaction with the system can be successful, based on the quality of the transition models. PSO-P tends to aggressively exploit model bias to choose good actions in the time horizon. Since it does not have any kind of memory, it does not follow a long-term strategy.

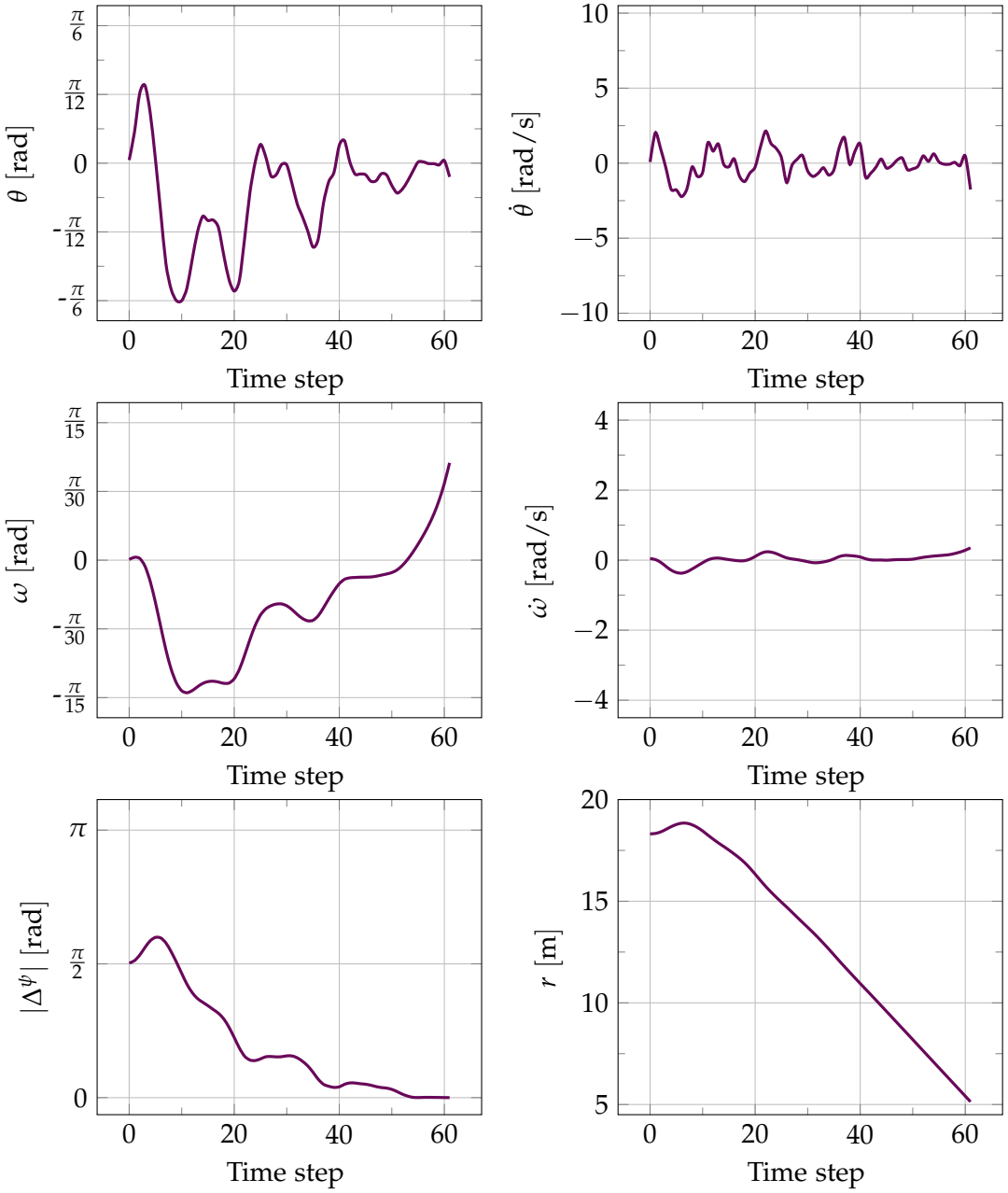


Figure 4.6: A single successful MAP trajectory. The bicycle starts at about 18 meters distance from the goal with an angle of about 90 degrees. PSO-P starts by increasing both the distance and angle towards the goal in the first five time steps. This allows the agent to make a more aggressive turn until time step 30, after which the bicycle is heading towards the goal in a mostly straight line.

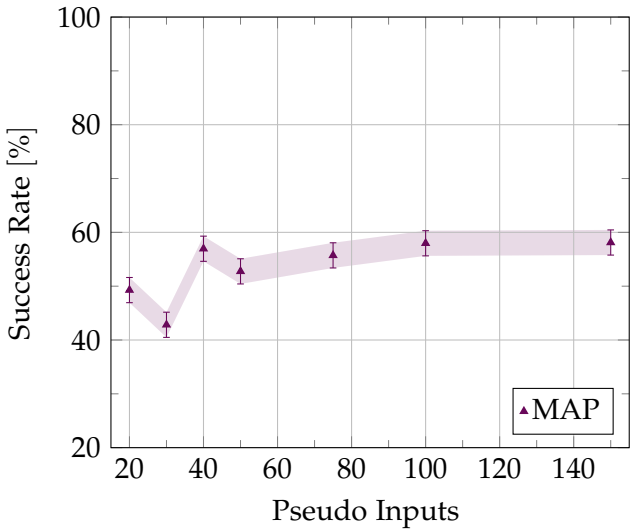


Figure 4.7: The success rate for different numbers of pseudo inputs with the standard error of the mean. The performance of MAP-predictions increases slightly with the number of pseudo inputs but always remains below 60 percent.

The bicycle reward function hints towards the goal by increasing the reward if the bicycle's frame points towards the goal. A typical trajectory created with the PSO policy therefore first describes a curve with a radius as small as possible in order to quickly point in the correct direction. If the bicycle points into the correct direction, PSO-P drives straight towards the goal until it is reached. The policy avoids falling over on the model since falling over reduces the achieved reward. It will however choose actions which are as close to falling over as possible, since the minimum radius of a curve is constrained by the maximum leaning angle ω.

Figure 4.8 shows that this strategy often is not successful. Minimal errors in the predictions of the models when the cyclist is leaning close to the maximum amount lead to the cyclist falling over. Therefore, most trajectories fail while driving a curve. If a trajectory starts at a state which does not require a tight curve, PSO-P can often reach the goal, since it can very effectively solve the balancing problem in non-extreme states and drive straight.

While figures 4.6 and 4.8 are based on a single transition model and give an idea about the performance of PSO-P using MAP-predictions, statistically significant results can be obtained using the multiple iterations of the evaluation algorithm with different data sets and transition models. Figure 4.7 shows the percentage of trajectories which reach the goal plotted against the number of pseudo inputs. The more expressive the model,

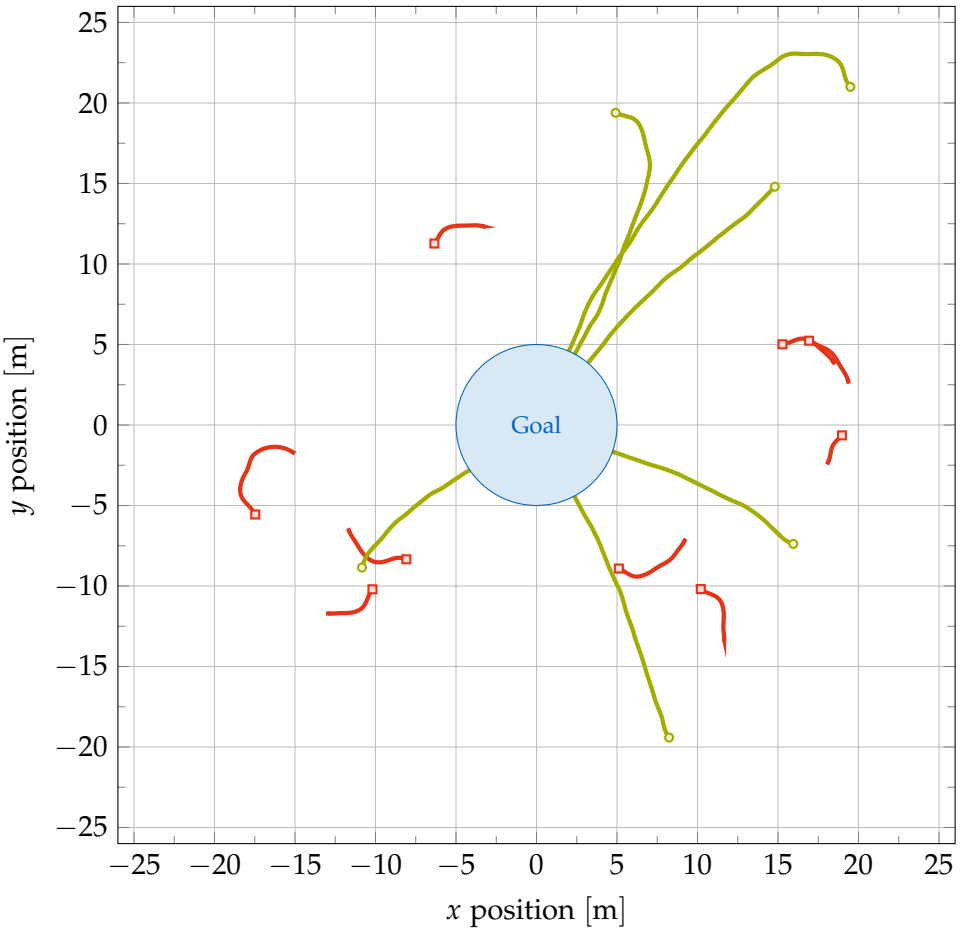


Figure 4.8: The x and y coordinates of the front tyre in representative episodes when using MAP-predictions with marks at the starting states. Successful episodes are colored in green and failed episodes are colored in red. PSO-P tends to be successful for starting states which do not require a curve to reach the goal. For other episodes, the policy usually fails during curved driving. If successful, trajectories usually consist of one curve to orient the bicycle followed by a straight line towards the goal.

the more successful PSO-P with MAP-predictions becomes, but overall, about half of the trajectories trajectories end in failure.

In order to reduce the aggressiveness with which PSO-P exploits the model, the uncertainty measure provided by the Gaussian processes are going to be integrated into the action value function. If the model predicts that with a large probability, the true system dynamics lead to failure, even if the MAP-prediction might still be a valid state, PSO-P should choose a less extreme action. The following section introduces how the reward function can be extended to beliefs about states and uses the uncertainty predicted by the transition model during planning without propagating it through multiple time steps.

4.3 Predictions with One-Step Uncertainties

PSO-P on the basis of MAP-predictions discards all information about model uncertainties by assuming that the mean of the predictive distribution produced by the transition model is the correct successive state. Long-term predictions for states multiple time steps into the future can be obtained by iterative application of the transition model since all intermediate states are deterministic. In the same way, the expected reward for every time step can easily be calculated for arbitrary reward functions since no integration using the state's density function is needed.

The first approach to incorporating uncertainties into planning with PSO-P presented in this thesis is a direct extension of the MAP approach. Instead of discarding them at every time step, the uncertainties returned by the transition model are used to evaluate the expected reward. These uncertainties are still not propagated however, as the algorithm for deriving intermediate states in long-term predictions essentially remains the same compared to the previous technique.

If states are not considered to be deterministic, it is no longer clear when an episode should be considered to be over. For every time step, the predictive state distributions $\mathrm{p}(\boldsymbol{s_1})$ to $\mathrm{p}(\boldsymbol{s_T})$ can have their probability mass spread over both terminal and non-terminal states. After defining how to calculate the intermediate state distributions using one-step uncertainties, this section considers how the probability that the predictive trajectory has already ended can be calculated with respect to the whole trajectory.

4.3.1 Long-Term predictions

Similar to the MAP-predictions defined in section 4.2.2, long-term predictions with one-step uncertainties are based on iterating the transition model using the means of the predicted Gaussian distributions. The intermediate states are no longer assumed to be deterministic however. They are random variables with the distribution returned by the transition model. The function mapping a pair of a state distribution $\boldsymbol{s_t}$ and an action $\boldsymbol{a_t}$ to the successive state distribution $\boldsymbol{s_{t+1}}$ is given by

$$\boldsymbol{s}_{\boldsymbol{t+1}}^{\mathrm{OS}}\colon \begin{cases} \mathcal{P}(\mathcal{S}^+) \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+) \\ \qquad (\boldsymbol{s_t}, \boldsymbol{a_t}) \mapsto f(\mathbb{E}[\boldsymbol{s_t}], \boldsymbol{a_t}), \end{cases} \tag{4.14}$$

Note that there can still be probability mass assigned to impossible states since the Gaussian is symmetric around the mean. Since only the mean is used for predictions, this does not cause a problem in this setting however.

The intermediate states $\boldsymbol{s}_{\boldsymbol{1}}^{\mathrm{OS}}$ to $\boldsymbol{s}_{\boldsymbol{T}}^{\mathrm{OS}}$ can again be calculated using dynamic programming. Terminal states are no longer modelled using fixed points in the transition function. Since the intermediate states are probabilistic, it is no longer clear when the predicted trajectory has ended. Instead, for every state, the probability of having reached the goal can be calculated as

$$\begin{aligned} \mathrm{p}\Big(\boldsymbol{s}_{\boldsymbol{t}}^{\mathrm{OS}} \in \mathcal{T}_{\mathrm{goal}}\Big) &= \mathrm{p}\Big(r_{\boldsymbol{s}_{\boldsymbol{t}}^{\mathrm{OS}}} \leq 5\Big), \\ \mathrm{p}\Big(\boldsymbol{s}_{\boldsymbol{t}}^{\mathrm{OS}} \in \mathcal{T}_{\mathrm{fallen}}\Big) &= \mathrm{p}\Big(|\omega_{\boldsymbol{s}_{\boldsymbol{t}}^{\mathrm{OS}}}| > \frac{\pi}{15}\Big) \end{aligned} \tag{4.15}$$

respectively. Since the intermediate state distributions are Gaussian, the marginal distributions for every variable in the state are (not necessarily independent) univariate Gaussian distributions. Given a state distribution $\boldsymbol{s} \sim \mathcal{N}(\boldsymbol{\mu_s}, \boldsymbol{\Sigma_s})$, the distribution of the i-th state variable is given by

$$\boldsymbol{s_i} \sim \mathcal{N}\Big(\mu_s^{(i)}, \Sigma_s^{(i,i)}\Big). \tag{4.16}$$

Probabilities of the form $\mathrm{p}(\boldsymbol{s_i} \leq k)$ are then given by

$$\mathrm{p}(\boldsymbol{s_i} \leq k) = \Phi\left(\frac{k - \mu_s^{(i)}}{\sqrt{\Sigma_s^{(i,i)}}}\right), \tag{4.17}$$

where Φ denotes the cumulative distribution function of the standard normal distribution.

The probability $\mathrm{p}(\boldsymbol{s}_t^{\mathrm{OS}} \in \mathcal{T}_{\text{fallen}})$ denotes the transition model's belief that given that the bicycle has not fallen down in the past, it will fall down exactly at time step t. Now consider the state $\boldsymbol{s}_{t+1}^{\mathrm{OS}}$ one time step later. Again, the probability $\mathrm{p}(\boldsymbol{s}_{t+1}^{\mathrm{OS}} \in \mathcal{T}_{\text{fallen}})$ denotes the belief that the bicycle will fall down at exactly time step $t+1$. The state for time step $t+1$ was calculated under the assumption that the trajectory has not already ended, since the transition model does not know about terminal states. However, the correct trajectory could have also ended at time step t. In this case, the trajectory should be considered to have been stuck at that terminal state. The probability $\mathrm{p}(\boldsymbol{s}_{t+1}^{\mathrm{OS}} \in \mathcal{T}_{\text{fallen}})$ then looses its significance in this case, since it has been calculated under wrong assumptions.

In order to correctly calculate the probability that the trajectory has already ended before or at time step $t+1$, the probabilities of falling down or reaching the goal at any of the previous time steps have to be considered. Let

$$\begin{aligned} \mathrm{p}(\mathcal{G}_t) &\coloneqq \mathrm{p}\big(\exists i \leq t \colon \boldsymbol{s_i} \in \mathcal{T}_{\text{goal}}\big), \\ \mathrm{p}(\mathcal{F}_t) &\coloneqq \mathrm{p}(\exists i \leq t \colon \boldsymbol{s_i} \in \mathcal{T}_{\text{fallen}}) \end{aligned} \tag{4.18}$$

denote the probabilities that at any point before or at t, the trajectory has already ended because of reaching the goal or falling down respectively. In other words, they denote the accumulated belief of the transition model that the trajectory has at some point hit a terminal state. These probabilities can be calculated using dynamic programming. They are given by

$$\begin{aligned} \mathrm{p}(\mathcal{G}_{t+1}) &= \mathrm{p}(\mathcal{G}_t) + (1 - \mathrm{p}(\mathcal{G}_t)) \cdot \mathrm{p}\big(\boldsymbol{s_{t+1}} \in \mathcal{T}_{\text{goal}}\big), \\ \mathrm{p}(\mathcal{F}_{t+1}) &= \mathrm{p}(\mathcal{F}_t) + (1 - \mathrm{p}(\mathcal{F}_t)) \cdot \mathrm{p}(\boldsymbol{s_{t+1}} \in \mathcal{T}_{\text{fallen}}), \end{aligned} \tag{4.19}$$

respectively, where the probabilities $\mathrm{p}(\mathcal{G}_0)$ and $\mathrm{p}(\mathcal{F}_0)$ can be calculated directly.

With these probabilities describing the overall belief that the predictive trajectory has not already ended, the expected reward for every state in the trajectory can be calculated. Consider the state $\boldsymbol{s_t}$ with the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu_t}, \boldsymbol{\Sigma_t})$. Its expected reward is given by

$$\begin{aligned} \mathbb{E}\big[\mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t})\big] &= \int \mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t})\, \mathrm{p}(\boldsymbol{s_t})\, \mathrm{d}\boldsymbol{s_t} \\ &= \int \mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t})\, \mathcal{N}(\boldsymbol{s_t} \mid \boldsymbol{\mu_t}, \boldsymbol{\Sigma_t})\, \mathrm{d}\boldsymbol{s_t}. \end{aligned} \tag{4.20}$$

In section 4.2.1, $\mathbf{R}_{\text{bicycle}}$ was specified using a definition by cases, separating the terminal states. To simplify the integration, it can be rewritten as a single sum using the indicator

function $\mathbb{I}$ in

$$\mathbf{R}_{\text{bicycle}}(\boldsymbol{s}) = \begin{cases} 2 & \cdot \mathbb{I}(\boldsymbol{s} \in \mathcal{T}_{\text{goal}}) \\ +0 & \cdot \mathbb{I}(\boldsymbol{s} \notin \mathcal{T}_{\text{goal}}, \boldsymbol{s} \in \mathcal{T}_{\text{fallen}}) \\ +\sqrt{2\pi\sigma^2_{\text{angle}}}\,\mathcal{N}(\Delta_\psi \mid 0, \sigma^2_{\text{angle}}) & \cdot \mathbb{I}(\boldsymbol{s} \notin \mathcal{T}_{\text{goal}}, \boldsymbol{s} \notin \mathcal{T}_{\text{fallen}}). \end{cases} \tag{4.21}$$

In the integration required for the expected reward, the indicator functions are replaced by their corresponding probabilities according to the law of total probability. The integral is then given by

$$\begin{aligned} \int \mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} &= 2 \cdot \mathrm{p}(\mathcal{G}_t) + 0 \cdot (1 - \mathrm{p}(\mathcal{G}_t))\,\mathrm{p}(\mathcal{F}_t) \\ &\quad + (1 - \mathrm{p}(\mathcal{G}_t))(1 - \mathrm{p}(\mathcal{F}_t)) \\ &\quad \cdot \int \hat{\mathbf{R}}_{\text{bicycle}}(\boldsymbol{s_t})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} \\ &= 2 \cdot \mathrm{p}(\mathcal{G}_t) + (1 - \mathrm{p}(\mathcal{G}_t))(1 - \mathrm{p}(\mathcal{F}_t)) \\ &\quad \cdot \int \hat{\mathbf{R}}_{\text{bicycle}}(\boldsymbol{s_t})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} \end{aligned} \tag{4.22}$$

where $\hat{\mathbf{R}}_{\text{bicycle}}$ denotes the reward term for states which are not terminal and is defined as

$$\hat{\mathbf{R}}_{\text{bicycle}}(\boldsymbol{s}) \coloneqq \sqrt{2\pi\sigma^2_{\text{angle}}}\,\mathcal{N}(\Delta_\psi(\boldsymbol{s}) \mid 0, \sigma^2_{\text{angle}}). \tag{4.23}$$

The angle Δ_ψ between the bicycle's heading and the goal can be calculated as the difference of two entries of the state vector. Equivalently, it is the result of applying a linear transformation to the Gaussian state distribution. Since $\Delta_\psi = \psi - \varphi$ and the variables in the state vector are organized as presented in section 4.1.2, this linear transformation has the form

$$\begin{aligned} \pi &= (0, 0, 0, 0, -1, 0, 1)^\mathsf{T}, \\ \boldsymbol{\Delta_\psi} &= \pi^\mathsf{T} \boldsymbol{s_t}. \end{aligned} \tag{4.24}$$

Linear transformations of Gaussians are also Gaussian [P+08]. The distribution of $\boldsymbol{\Delta_\psi}$ is given by

$$\boldsymbol{\Delta_\psi} \sim \mathcal{N}(\pi^\mathsf{T} \boldsymbol{\mu_t}, \pi^\mathsf{T} \boldsymbol{\Sigma_t} \pi). \tag{4.25}$$

The expected reward for non-terminal states can then be rewritten as the integral of a product of univariate Gaussian distributions. With $c \coloneqq \sqrt{2\pi\sigma^2_{\text{angle}}}$ it is given by

$$\begin{aligned}
\int \hat{\mathbf{R}}_{\text{bicycle}}(\boldsymbol{s_t})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} &= \int c\,\mathcal{N}(\Delta_\psi(\boldsymbol{s_t}) \mid 0, \sigma^2_{\text{angle}})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} \\
&= c\int \mathcal{N}(\Delta_\psi(\boldsymbol{s_t}) \mid 0, \sigma^2_{\text{angle}})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} \\
&= c\int \mathcal{N}(\boldsymbol{\Delta_\psi} \mid 0, \sigma^2_{\text{angle}})\,\mathrm{p}(\boldsymbol{\Delta_\psi})\,\mathrm{d}\boldsymbol{\Delta_\psi} \\
&= c\int \mathcal{N}(\boldsymbol{\Delta_\psi} \mid 0, \sigma^2_{\text{angle}})\,\mathcal{N}(\boldsymbol{\Delta_\psi} \mid \pi^\mathsf{T}\boldsymbol{\mu_t}, \pi^\mathsf{T}\boldsymbol{\Sigma_t}\pi)\,\mathrm{d}\boldsymbol{\Delta_\psi} \\
&= c\,\mathcal{N}(\pi^\mathsf{T}\boldsymbol{\mu_t} \mid 0, \sigma^2_{\text{angle}} + \pi^\mathsf{T}\boldsymbol{\Sigma_t}\pi).
\end{aligned} \tag{4.26}$$

The product of two Gaussian density functions is another unnormalized Gaussian density function. The normalization constant can in turn be written in terms of Gaussian densities and is the result of the integration [P+08].

Together these results give a closed form solution for the expected reward for the state $\boldsymbol{s_t}$ which is given by

$$\begin{aligned}
\mathbb{E}\big[\mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t})\big] &= 2\cdot\mathrm{p}(\mathcal{G}_t) + (1-\mathrm{p}(\mathcal{G}_t))(1-\mathrm{p}(\mathcal{F}_t))\int \hat{\mathbf{R}}_{\text{bicycle}}(\boldsymbol{s_t})\,\mathrm{p}(\boldsymbol{s_t})\,\mathrm{d}\boldsymbol{s_t} \\
&= 2\cdot\mathrm{p}(\mathcal{G}_t) + (1-\mathrm{p}(\mathcal{G}_t))(1-\mathrm{p}(\mathcal{F}_t)) \\
&\qquad \cdot\sqrt{2\pi\sigma^2_{\text{angle}}}\,\mathcal{N}(\pi^\mathsf{T}\boldsymbol{\mu_t} \mid 0, \sigma^2_{\text{angle}} + \pi^\mathsf{T}\boldsymbol{\Sigma_t}\pi).
\end{aligned} \tag{4.27}$$

Note that for states with very low uncertainty, that is states for which $\pi^\mathsf{T}\boldsymbol{\Sigma_t}\pi$ goes to zero, the expected value converges towards the original reward function. Similarly, both $\mathrm{p}(\mathcal{G}_t)$ and $\mathrm{p}(\mathcal{F}_t)$ are close zero or one for all time steps, converging to the binary behaviour of deterministic states. Using the one-step uncertainties, the action value function can therefore be calculated as

$$\begin{aligned}
\hat{\mathbf{V}}^{\text{OS}}_{\boldsymbol{s_0}}(\boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}}) &= \mathbb{E}\left[\sum_{t=1}^{T}\gamma^t\,\mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t}) \,\middle|\, f, \boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}}\right] \\
&= \sum_{t=1}^{T}\gamma^t\,\mathbb{E}\big[\mathbf{R}_{\text{bicycle}}(\boldsymbol{s_t}) \,\big|\, f, \boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{T-1}}\big] \\
&= \sum_{t=1}^{T}\gamma^t\Big(2\cdot\mathrm{p}(\mathcal{G}^{\text{OS}}_t) + (1-\mathrm{p}(\mathcal{G}^{\text{OS}}_t))(1-\mathrm{p}(\mathcal{F}^{\text{OS}}_t)) \\
&\qquad \sqrt{2\pi\sigma^2_{\text{angle}}}\,\mathcal{N}(\pi^\mathsf{T}\boldsymbol{\mu}^{\text{OS}}_{\boldsymbol{t}} \mid 0, \sigma^2_{\text{angle}} + \pi^\mathsf{T}\boldsymbol{\Sigma}^{\text{OS}}_{\boldsymbol{t}}\pi)\Big).
\end{aligned} \tag{4.28}$$

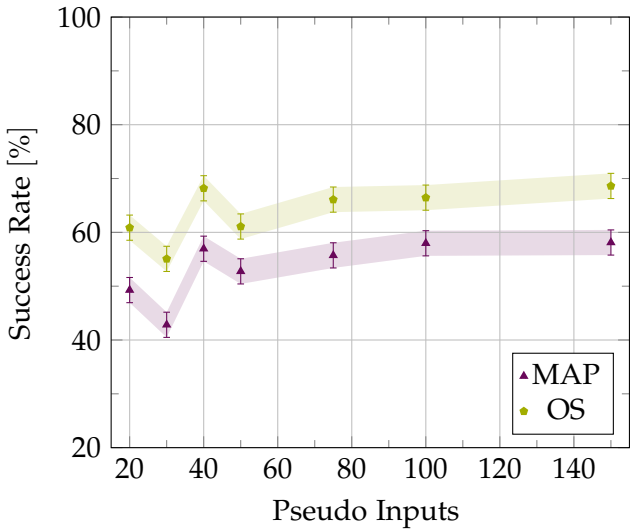


Figure 4.9: The success rate for different numbers of pseudo inputs with the standard error of the mean. Considering one step uncertainties increases the success rate by about 10 percent for all numbers of pseudo inputs.

4.3.2 Results using One-Step Uncertainties

Figure 4.11 shows a number of trajectories generated with PSO-P applied to the one-step action value function $\hat{\mathbf{V}}^{\mathrm{OS}}$. The trajectories generated look very similar to the MAP-trajectories shown in figure 4.8. Both figures were created using the same transition model and starting states. Comparing the two figures, more trajectories were successful using the approach with one-step uncertainties. Figure 4.9 shows that this is a statistically significant result.

For all numbers of pseudo inputs, and therefore for all different levels of model expressiveness, considering one-step uncertainties in planning is beneficial. PSO-P aggressively exploits model bias and creates trajectories where the bicycle is nearly falling over. Since the predictive distribution is Gaussian and therefore symmetric, half of the plausible successive states are more extreme than the predictive mean. The expected reward reflects that these more extreme realizations might cause the bicycle to fall down and therefore the trajectory to end. This incentivizes PSO-P to choose actions where most possible posterior states for every time step are not failure states.

The newly introduced uncertainties do not prompt PSO-P to establish a noticeable safety margin towards the maximum value of ω however. Figure 4.10 shows why this

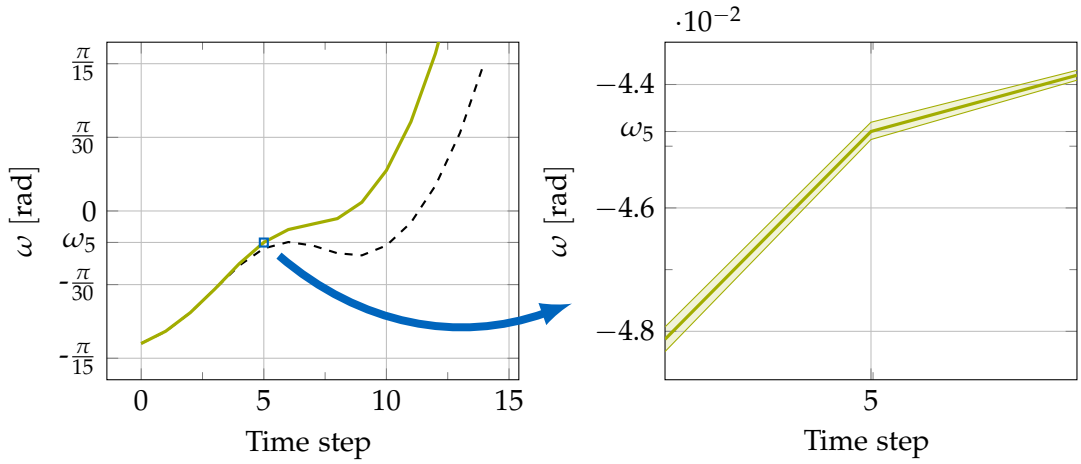


Figure 4.10: A visualization of the one step uncertainties. Long term predictions using one step uncertainties do not accumulate uncertainties. Instead, they are given by the predictive variance of the transition model for one specific step. Since the transition models are very accurate for single steps, these uncertainties can only be seen in a plot when zoomed in considerably. The safety margin induced by these uncertainties therefore is also very small.

is the case: Since the transition model is very confident about predicting one single step into the future, the predictive uncertainties (and therefore the safety margin) are very small. This is to be expected however, since a model which is supposed to be able to predict multiple steps into the future must show very good performance for a single step in order to not quickly deviate from the correct trajectory.

Small errors in the single predictions accumulate and after some time steps close to the correct trajectory, predicted trajectories tend to diverge from the truth rapidly. While for every time step the transition model only makes small mistakes compared to the correct transition function, iterating it means that later invocations create predictions based on wrong assumptions about the prior state. The intermediate prior states have up to now been assumed to be deterministic. The next section introduces how this accumulation of errors can be modelled by propagating the uncertainty about the current state through the Gaussian process model to produce a more correct posterior distribution.

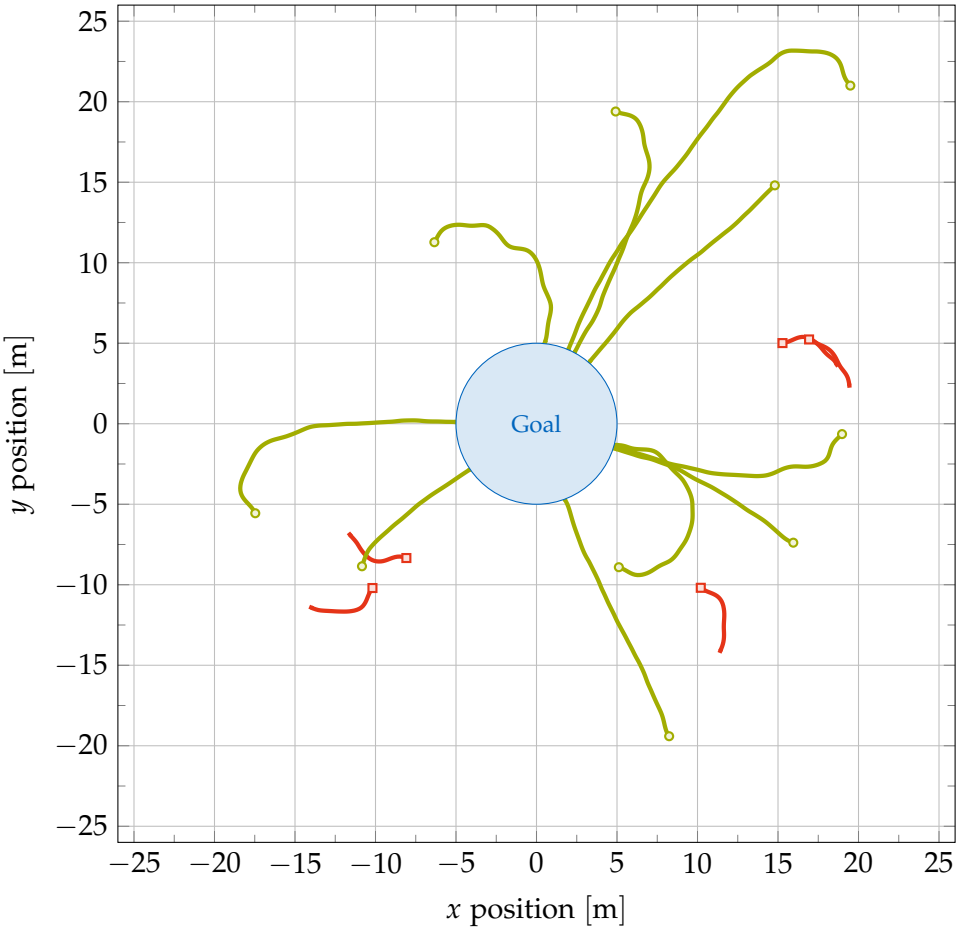


Figure 4.11: The x and y coordinates of the front tyre in representative episodes when using one step uncertainties with marks at the starting states. Successful episodes are colored in green and failed episodes are colored in red. When compared to MAP trajectories in figure 4.8, PSO-P generates very similar results. However, considering one step uncertainties has a higher success rate when curved driving is needed.

4.4 Predictions with Multi-Step Uncertainties

Considering the one-step uncertainties produced by the transition model for planning using PSO-P shows improved results when compared to only using MAP-predictions. Figure 4.10 however shows that these uncertainties are very small since the transition models used for the bicycle benchmark are very confident about their predictions. Over the course of multiple time steps, small errors nevertheless accumulate and cause large errors in the predicted trajectories which are not reflected well in the one-step uncertainties.

In order to arrive at more reliable estimates of the true uncertainty about predictions several steps into the future, the uncertainty about the state $\boldsymbol{s_t}$ has to be taken into account when predicting the distribution of the state $\boldsymbol{s_{t+1}}$. Since the transition dynamics $\mathbf{f}_{\text{bicycle}}$ are nonlinear, calculating the propagation of the prior uncertainty through the Gaussian process directly is not analytically tractable and the resulting distribution usually is not a Gaussian. This section introduces an approximation of this propagation via linearization of the transition model around the predictive mean as presented in [KF09] and [DFR15]. This approximation can be used to create long-term predictions with accumulated uncertainties.

4.4.1 Propagation of Uncertainties using Linearization

A transition model f described in section 4.1.2 consists of a group of Gaussian processes f_d, one for every dimension d in a bicycle state. Instead of predicting the successive state $\boldsymbol{s_{t+1}}$ directly given a pair of deterministic state $\boldsymbol{s_t}$ and action $\boldsymbol{a_t}$, the GPs are used to predict $\boldsymbol{\Delta_t} = \boldsymbol{s_{t+1}} - \boldsymbol{s_t}$, the difference between the two states. The model f_d is used to predict the d-th entry of this vector.

Using MAP-predictions, $\boldsymbol{\Delta_t}$ is assumed to be one deterministic value and using the one-step uncertainties, the complete Gaussian distribution $\boldsymbol{\Delta_t} \sim \mathcal{N}(\boldsymbol{\mu_\Delta}, \boldsymbol{\Sigma_\Delta})$ is the basis of the distribution of the successive state. Since $\boldsymbol{s_t}$ is assumed deterministic in these cases, the distribution of $\boldsymbol{s}^{\text{OS}}_{\boldsymbol{t+1}}$ is given by

$$\begin{aligned} \boldsymbol{s}^{\text{OS}}_{\boldsymbol{t+1}} &= \boldsymbol{s_t} + \boldsymbol{\Delta_t}, \\ \boldsymbol{s}^{\text{OS}}_{\boldsymbol{t+1}} &\sim \mathcal{N}(\boldsymbol{s_t} + \boldsymbol{\mu_\Delta}, \boldsymbol{\Sigma_\Delta}). \end{aligned} \tag{4.29}$$

In both cases, the state was again assumed to be deterministic for the timestep $t+1$ and located at $\mathbb{E}[\boldsymbol{s_{t+1}}]$ in order to derive the distribution of the next state $\boldsymbol{s_{t+2}}$, preventing the accumulation of uncertainties.

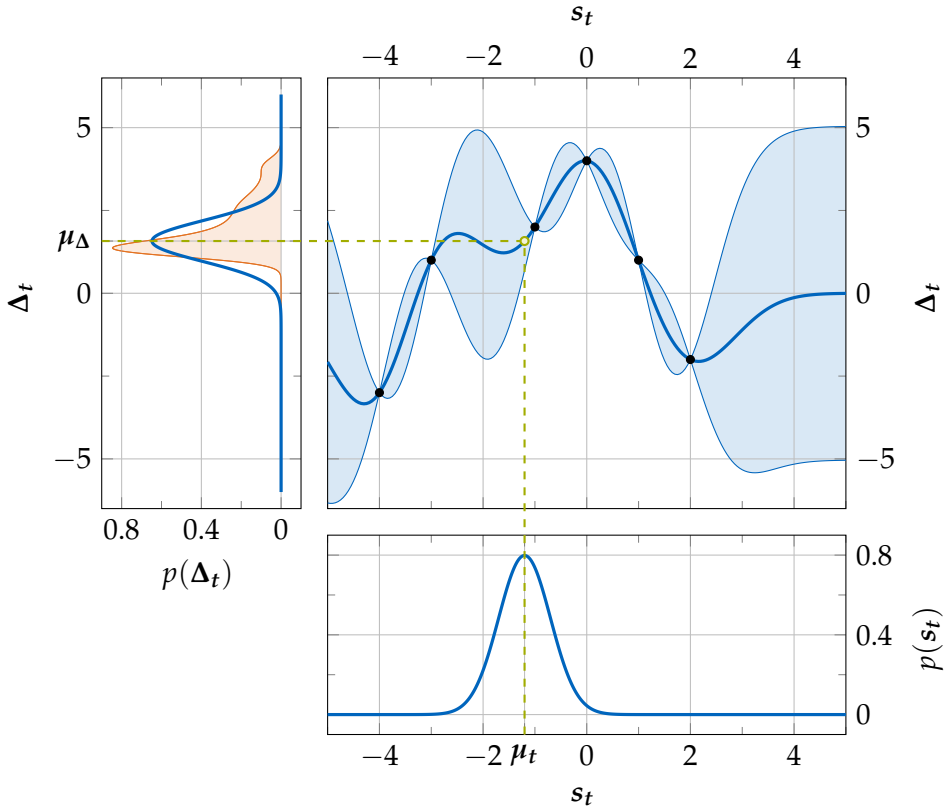


Figure 4.12: This figure shows the propagation of uncertainties through a Gaussian process transition model (upper right panel). The belief about the prior state s_t is assumed to be Gaussian (lower right panel). The orange shaded distribution of state changes Δ_t (upper left panel) is not Gaussian in general and cannot be calculated analytically. A Gaussian approximation can be obtained by linearizing the transition model around the prior mean μ_t.

Algorithm 4.5 Computing the successive state distribution

Let $s_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ denote a Gaussian state distribution and $a_t \in \mathcal{A}$ a deterministic action.

1: Compute the augmented state distribution $\mathrm{p}(\hat{s}_t)$
2: Approximate the predictive GP distribution $\mathrm{p}(\Delta_t) = f(\hat{s}_t, a_t)$
3: Approximate the successive state distribution $\mathrm{p}(s_{t+1})$ as the sum of s_t and Δ_t

Now assume the state $\boldsymbol{s}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ is already uncertain. In order to propagate this uncertainty through the transition model, the distribution $\mathrm{p}(\boldsymbol{\Delta}_t)$ is computed by solving the integral

$$\mathrm{p}(\boldsymbol{\Delta}_t) = \iint \mathrm{p}(f(\boldsymbol{s}_t, \boldsymbol{a}_t) \mid \boldsymbol{s}_t)\, \mathrm{p}(\boldsymbol{s}_t)\, \mathrm{d}f\, \mathrm{d}\boldsymbol{s}_t, \tag{4.30}$$

that is, by marginalizing both the belief over the current state $\boldsymbol{s}_t$ and all plausible transition models f. Figure 4.12 illustrates that computing the exact predictive distribution is analytically intractable, since the GP models are highly nonlinear.

Instead, the distribution is approximated by a Gaussian. Assume that the distribution $\boldsymbol{\Delta}_t \sim \mathcal{N}(\boldsymbol{\mu}_\Delta, \boldsymbol{\Sigma}_\Delta)$ is known. Since $\boldsymbol{s}_{t+1}$ is the sum of $\boldsymbol{s}_t$ and $\boldsymbol{\Delta}_t$, a Gaussian approximation of the posterior distribution is given by

$$\begin{aligned} &\boldsymbol{s}_{t+1} \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}), \text{ where} \\ &\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\mu}_\Delta \\ &\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_\Delta + \operatorname{cov}[\boldsymbol{s}_t, \boldsymbol{\Delta}_t] + \operatorname{cov}[\boldsymbol{\Delta}_t, \boldsymbol{s}_t], \end{aligned} \tag{4.31}$$

where $\operatorname{cov}[\boldsymbol{s}_t, \boldsymbol{\Delta}_t]$ denotes the covariance between the prior state and the state change. Algorithm 4.5 shows the different steps necessary to compute this predictive distribution.

The first step is to compute the augmented state distribution $\mathrm{p}(\hat{\boldsymbol{s}}_t) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{\Sigma}}_t)$. Instead of using the state $\boldsymbol{s}_t$ directly, the Gaussian process models are presented with the augmented state $\hat{\boldsymbol{s}}_t$ when used for predictions. This distinction is made since in general, the Gaussian processes may have been trained on data which has been preprocessed, for example using dimensionality reduction techniques. In this thesis, augmenting a state means replacing the angles φ and ψ by their sine and cosine values as shown in equation (4.6). Equivalently, the angles have to be replaced in the joint distribution of the state. Let $\boldsymbol{\alpha} \sim \mathcal{N}(\mu, \sigma^2)$ denote an angle with a Gaussian distribution. Using Euler's identity, it can be shown that the expected values of its sine and cosine are given by

$$\begin{aligned} \mathbb{E}[\sin \boldsymbol{\alpha}] &= \exp\left(-\frac{\sigma^2}{2}\right) \sin \mu, \\ \mathbb{E}[\cos \boldsymbol{\alpha}] &= \exp\left(-\frac{\sigma^2}{2}\right) \cos \mu, \end{aligned} \tag{4.32}$$

respectively [Dei10]. The variances of the two random variables can be calculated as

$$\begin{aligned} \operatorname{var}[\sin \boldsymbol{\alpha}] &= \frac{1}{2} - \frac{1}{2} \exp\left(-2\sigma^2\right) \cos(2\mu) - \exp(-\sigma^2) \sin^2 \mu, \\ \operatorname{var}[\cos \boldsymbol{\alpha}] &= \frac{1}{2} + \frac{1}{2} \exp\left(-2\sigma^2\right) \cos(2\mu) - \exp(-\sigma^2) \cos^2 \mu. \end{aligned} \tag{4.33}$$

The mean $\hat{\boldsymbol{\mu}}_t$ and covariance matrix $\hat{\boldsymbol{\Sigma}}_t$ of the augmented are calculated by inserting these values into the original mean $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$ instead of the respective angles. Since the sine and cosine functions are nonlinear, the covariances of the transformed angles with all other variables are assumed to be zero.

The second step in the algorithm is to calculate $\mathrm{p}(\boldsymbol{\Delta}_t)$ from the augmented state distribution. Since it cannot be calculated directly, the distribution is approximated with a Gaussian. This approximation is obtained by linearizing the predictive mean function of the transition model around the mean of the prior distribution. The mean function μ_f of the transition model f is therefore approximated by its first-order Taylor expansion f^{lin} around $\hat{\boldsymbol{\mu}}$ which is given by

$$\mu_f(\hat{\boldsymbol{s}}_t, \boldsymbol{a}_t) \approx f^{\text{lin}}(\hat{\boldsymbol{s}}_t, \boldsymbol{a}_t) = \mu_f(\hat{\boldsymbol{\mu}}, \boldsymbol{a}_t) + \frac{\partial \mu_f(\hat{\boldsymbol{\mu}}, \boldsymbol{a}_t)}{\partial \hat{\boldsymbol{\mu}}}(\hat{\boldsymbol{s}}_t - \hat{\boldsymbol{\mu}}_t). \tag{4.34}$$

Using this representation, $\boldsymbol{\Delta}_t$ is given as the affine transformation of $\hat{\boldsymbol{s}}_t$ defined by f^{lin}, which is a Gaussian $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\Delta}}, \boldsymbol{\Sigma}_{\boldsymbol{\Delta}})$. This transformation is illustrated in figure 4.12.

The predictive mean $\boldsymbol{\mu}_{\boldsymbol{\Delta}}$ is obtained by evaluating the linearization at the prior mean and is given by

$$\boldsymbol{\mu}_{\boldsymbol{\Delta}} = f^{\text{lin}}(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t) = \mu_f(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t) = \mathbb{E}[f(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)]. \tag{4.35}$$

The d-th component of this vector is calculated using the predictive mean of the Gaussian process f_d in the transition model. Using lemma 11, it can be computed as

$$\boldsymbol{\mu}_{\boldsymbol{\Delta}}^d = \mu_f^d(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t) = \mathbb{E}[f_d(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)] = \mathcal{K}_d(\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{X}})\boldsymbol{\beta}_d, \tag{4.36}$$

where $\mathcal{K}_d$ denotes the kernel function of f_d, $\boldsymbol{\beta}_d$ is a precomputed vector and $\hat{\boldsymbol{X}}$ is the augmented set of training points (or pseudo-inputs).

Using the different Gaussian process models, the derivative of the predictive mean function at $\hat{\boldsymbol{\mu}}$ can be calculated explicitly. It is a matrix

$$V \coloneqq \frac{\partial}{\partial \hat{\boldsymbol{\mu}}} \mu_f(\hat{\boldsymbol{\mu}}, \boldsymbol{a}_t) \tag{4.37}$$

whose d-th line is determined by f_d. It is given by

$$V_d = \frac{\partial}{\partial \hat{\boldsymbol{\mu}}_t} \mu_f^d(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t) = \frac{\partial\, \mathcal{K}_d(\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{X}})}{\partial \hat{\boldsymbol{\mu}}_t} \boldsymbol{\beta}_d \tag{4.38}$$

and depends on the derivative of the kernel function with respect to the augmented mean. The Gaussian processes used in this thesis only make the prior assumption that

the transition dynamics of the bicycle benchmark are a smooth function and are trained using a squared exponential kernel. For the squared exponential kernel as shown in definition 10, this derivation can be calculated as

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\mu}} \mathcal{K}_{\mathrm{SE}}(\boldsymbol{\mu}, \boldsymbol{x}) &= \frac{\partial}{\partial \boldsymbol{\mu}} \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{\mu}-\boldsymbol{x})^\mathsf{T} \boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu}-\boldsymbol{x})\right) \\ &= \mathcal{K}_{\mathrm{SE}}(\boldsymbol{\mu}, \boldsymbol{x}) \frac{\partial}{\partial \boldsymbol{\mu}}\left(-\frac{1}{2}(\boldsymbol{\mu}-\boldsymbol{x})^\mathsf{T} \boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu}-\boldsymbol{x})\right) \\ &= -\mathcal{K}_{\mathrm{SE}}(\boldsymbol{\mu}, \boldsymbol{x}) \boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu}-\boldsymbol{x}).\end{aligned} \tag{4.39}$$

The predictive variance $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}}$ consists of two components, the propagated uncertainty about the prior state and the newly added model uncertainties for this transition. Since the transition model is linearized, the propagation of the prior uncertainties is equivalent to a linear transformation of the prior state distribution using the predictive mean's derivative V. Since the transition model is only evaluated for the prior mean $\hat{\boldsymbol{\mu}}_t$ and not for all other possible states, the true model uncertainty is unknown. Instead, it is assumed to be constant and equal to the uncertainty for the prior mean. The two components are independent given the prior distribution and therefore, the predictive variance is given by

$$\begin{aligned}\boldsymbol{\Sigma}_{\boldsymbol{\Delta}} &= \operatorname{var}[V \hat{\boldsymbol{s}}_t] + \operatorname{var}[f(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)] \\ &= V \hat{\boldsymbol{\Sigma}}_t V^\mathsf{T} \; + \operatorname{diag}\left(\operatorname{var}[f_1(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)], \ldots, \operatorname{var}[f_D(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)]\right),\end{aligned} \tag{4.40}$$

since $\hat{\boldsymbol{s}}_t$ is Gaussian [P+08]. The covariance of the augmented state and the state change is given by

$$\operatorname{cov}[\boldsymbol{s}_t, \boldsymbol{\Delta}_t] = \operatorname{cov}[\boldsymbol{s}_t, V \boldsymbol{s}_t] = \operatorname{cov}[\boldsymbol{s}_t, \boldsymbol{s}_t] V^\mathsf{T} = \hat{\boldsymbol{\Sigma}}_t V^\mathsf{T}, \tag{4.41}$$

using the linearity of expectations. Equivalently, it holds that $\operatorname{cov}[\boldsymbol{\Delta}_t, \boldsymbol{s}_t] = V \hat{\boldsymbol{\Sigma}}_t$.

Using the linearization of the predictive mean function, all components necessary to approximate the posterior distribution $\mathrm{p}(\boldsymbol{s}_{t+1})$ can be calculated. The belief about the successive state is a Gaussian distribution characterized by

$$\begin{aligned}&\boldsymbol{s}_{t+1} \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}), \text{ where} \\ &\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\mu}_{\boldsymbol{\Delta}} \\ &\qquad = \boldsymbol{\mu}_t + \mathbb{E}[f(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)] \\ &\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_{\boldsymbol{\Delta}} + \operatorname{cov}[\boldsymbol{s}_t, \boldsymbol{\Delta}_t] + \operatorname{cov}[\boldsymbol{\Delta}_t, \boldsymbol{s}_t] \\ &\qquad = \boldsymbol{\Sigma}_t + V \hat{\boldsymbol{\Sigma}}_t V^\mathsf{T} + \operatorname{var}[f(\hat{\boldsymbol{\mu}}_t, \boldsymbol{a}_t)] + \hat{\boldsymbol{\Sigma}}_t V^\mathsf{T} + V \hat{\boldsymbol{\Sigma}}_t.\end{aligned} \tag{4.42}$$

4.4.2 Long-Term predictions

Using the linearization presented in section 4.4.1 to propagate state distributions through the transition model, it is possible to create long-term predictions with accumulated uncertainties. While the starting state $\boldsymbol{s_0}$ is always deterministic since it is an observation of the real system, the intermediate states $\boldsymbol{s_1}$ to $\boldsymbol{s_T}$ are random variables. In the case of the bicycle benchmark, the transition dynamics $\mathbf{f}_{\text{bicycle}}$ are completely deterministic, so all uncertainties in the predictions stem from the transition model's confidence about its predictions.

Using linearization, the intermediate states' distributions $\mathrm{p}(\boldsymbol{s_1})$ to $\mathrm{p}(\boldsymbol{s_T})$ are multivariate Gaussians. Given a pair of a state distribution $\boldsymbol{s_t}$ and an action $\boldsymbol{a_t}$, the distribution of the successive state is given by

$$\boldsymbol{s}_{\boldsymbol{t+1}}^{\text{MS}} : \begin{cases} \mathcal{P}(\mathcal{S}^+) \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+) \\ \qquad (\boldsymbol{s_t}, \boldsymbol{a_t}) \mapsto f(\boldsymbol{s_t}, \boldsymbol{a_t}), \end{cases} \tag{4.43}$$

With multi-step uncertainties it is no longer necessary to fall back to deterministic intermediate states for prediction.

Figure 4.13 shows an example for long-term predictions using multi-step uncertainties. As with the other approaches presented in this thesis, all intermediate states $\boldsymbol{s}_{\boldsymbol{1}}^{\text{MS}}$ to $\boldsymbol{s}_{\boldsymbol{T}}^{\text{MS}}$ can be calculated using dynamic programming. In contrast to the one-step uncertainties presented in figure 4.10, the state distributions obtained using linearization quite accurately estimate the predictive error. The uncertainties reach the same order of magnitude as the dynamic range of their variable and become visible in the plot after a few time steps. After about seven time steps, the uncertainties about ω grow and two standard deviations cover most of the value range. At this point, the model's predictions can be interpreted as almost complete uncertainty about the correct state. At the same time however, the predictive mean is no longer close to the correct trajectory. The model correctly identifies the decline in accuracy of its predictions.

Large uncertainties about ω lead to probability mass in the tail of the Gaussian being placed outside of the valid range of values. For every such time step, the accumulated probability $\mathrm{p}(\mathcal{F}_t^{\text{MS}})$ of already having fallen down at some time before or at time step t increases. This probability is calculated via dynamic programming using the rules presented in section 4.3.1. While falling down does not cause a direct punishment (since the associated reward is zero), a higher probability of an ended trajectory lowers the amount of reward which can be earned for the later time steps in the prediction.

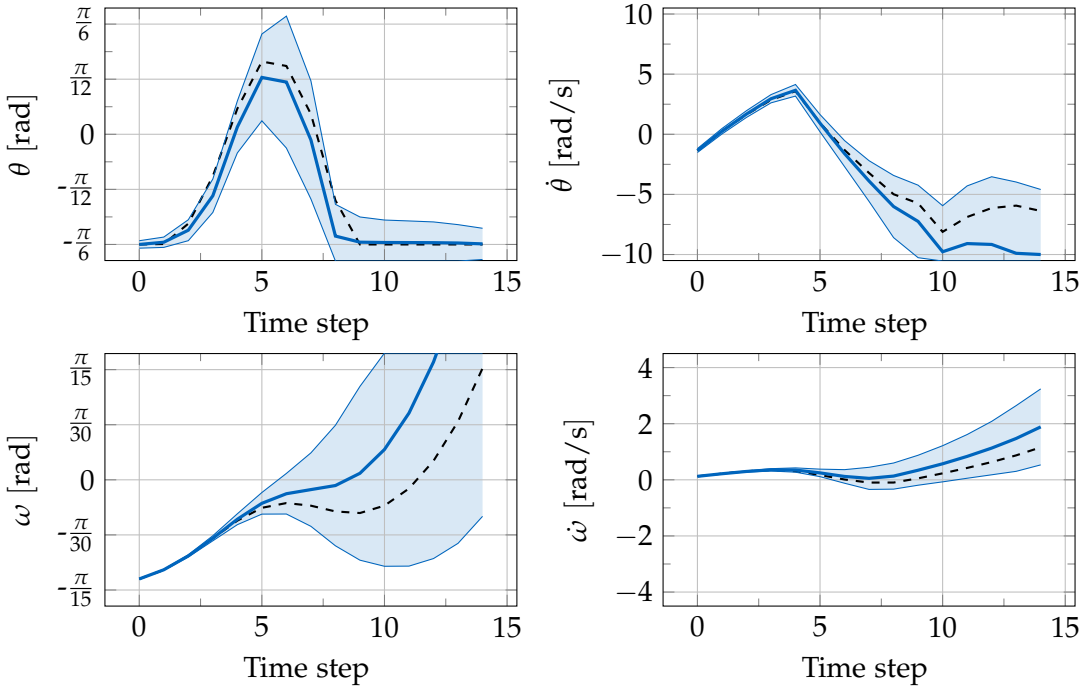


Figure 4.13: Colored long-term predictions for one time horizon using multi step uncertainties and starting from a deterministic state compared to the dashed simulation. Considering multi-step uncertainties yields the shaded measure of confidence, which in this case correctly identifies that the predictive error increases after about half the time horizon.

Given the intermediate state distributions $\boldsymbol{s}_t^{\text{MS}} \sim \mathcal{N}(\boldsymbol{\mu}_t^{\text{MS}}, \boldsymbol{\Sigma}_t^{\text{MS}})$, the action value function using multi-step uncertainties is calculated analogously to the one-step action value function presented in equation (4.28), since only the way of calculating the state distributions has changed. Having calculated the terminal probabilities $\mathrm{p}(\mathcal{G}_t^{\text{MS}})$ and $\mathrm{p}(\mathcal{F}_t^{\text{MS}})$ it is given by

$$\begin{aligned}\hat{\mathbf{V}}_{\boldsymbol{s}_0}^{\text{MS}}(\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}) &= \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t \, \mathbf{R}_{\text{bicycle}}(\boldsymbol{s}_t) \,\middle|\, f, \boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}\right] \\ &= \sum_{t=1}^{T} \gamma^t \Big(2 \cdot \mathrm{p}(\mathcal{G}_t^{\text{MS}}) + (1 - \mathrm{p}(\mathcal{G}_t^{\text{MS}}))(1 - \mathrm{p}(\mathcal{F}_t^{\text{MS}})) \\ &\qquad \sqrt{2\pi\sigma_{\text{angle}}^2}\, \mathcal{N}(\pi^{\mathsf{T}} \boldsymbol{\mu}_t^{\text{MS}} \mid 0, \sigma_{\text{angle}}^2 + \pi^{\mathsf{T}} \boldsymbol{\Sigma}_t^{\text{MS}} \pi) \Big).\end{aligned} \tag{4.44}$$

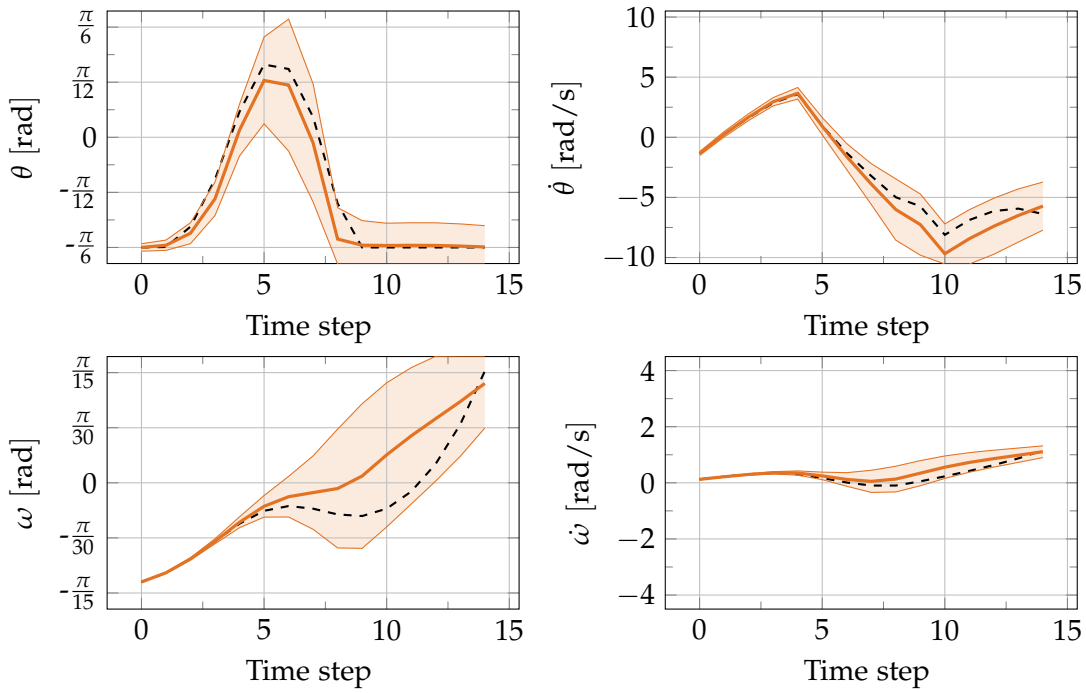


Figure 4.14: Colored long-term predictions for one time horizon using multi step uncertainties with truncation and starting from a deterministic state compared to the dashed simulation. While the amount of probability mass spread over terminal states is negligible, the predictions using the truncation of Gaussians is almost equal to using only linearization as shown in figure 4.13. However, for the last time steps, truncating the terminal regions of ω yields more correct predictions.

4.4.3 Posterior States using the Truncation of Gaussians

Using the intermediate state distributions $p(\boldsymbol{s}_t^{\text{MS}})$ directly when predicting successive states is a good approximation in most cases but is not correct when the terminal probabilities $p(\boldsymbol{s}_t \in \mathcal{T}_{\text{fallen}})$ or $p(\boldsymbol{s}_t \in \mathcal{T}_{\text{goal}})$ are large. For any state $\boldsymbol{s}_t$ in the set of terminal states $\mathcal{T}$, the successive state $\boldsymbol{s}_{t+1}$ is always defined to be equal to $\boldsymbol{s}_t$, regardless of the action performed by the agent. The transition model f is however trained using only non-terminal state transitions, that is, state transitions where neither the prior nor the posterior state is terminal. This implies the assumption that the transition model only operates on non-terminal states, similar to transition dynamics $\mathbf{f}$ in general only being defined on $\mathcal{S}$, the set of non terminal states. The attractor property of terminal states has to be modelled outside of the transition models.

In the action value function, this is reflected by the accumulative probabilities $p(\mathcal{G})$ and $p(\mathcal{F})$. Any probability mass which falls to their respective terminal states is assumed to stay there for the remaining time steps, since their respective trajectories end at that point. When predicting successive states using the function $\boldsymbol{s}_{t+1}^{\text{MS}}$, this information is not considered however. The prior Gaussian distribution used for linearization can have considerable probability mass attributed to terminal states. Consulting the transition model however implies the condition that the trajectory has not yet ended. This condition is encoded in the successive state function $\boldsymbol{s}_{t+1}^{\text{TG}}$ given by

$$\boldsymbol{s}_{t+1}^{\text{TG}} \colon \begin{cases} \mathcal{P}(\mathcal{S}^+) \times \mathcal{A} \to \mathcal{P}(\mathcal{S}^+) \\ \qquad (\boldsymbol{s}_t, \boldsymbol{a}_t) \mapsto f(\boldsymbol{s}_t, \boldsymbol{a}_t \mid \boldsymbol{s}_t \notin \mathcal{T}). \end{cases} \tag{4.45}$$

Given a Gaussian state distribution $\boldsymbol{s}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, the probability distribution

$$p(\boldsymbol{s}_t \mid \boldsymbol{s}_t \notin \mathcal{T}) \tag{4.46}$$

can in general not be calculated analytically and is approximated with another Gaussian using moment matching. In the bicycle benchmark, the density function of this distribution can be obtained by truncating the original Gaussian distribution using the hyperplanes defined by equation (4.15) and renormalizing the result. Algorithms to calculate the moments of the resulting distribution are described in [Her05] and [Tou09]. The action value function $\hat{\mathbf{V}}_{s_0}^{\text{TG}}$ is equivalent to the action value function without the truncation of Gaussians $\hat{\mathbf{V}}_{s_0}^{\text{MS}}$.

Figure 4.14 shows predictions using multi-step uncertainties and the truncation of the state distribution Gaussian after every time step. While the predictions with small uncertainties are almost identical to the predictions in figure 4.13, they differ once the uncertainties are large enough to distribute mass over terminal states. After every time step, the mean of the prediction for ω tends more towards the middle of the value range and thus towards the most uninformative prediction with a mean at zero and the uncertainty spread over the complete range of possible values.

For every time step, the probability mass spread over terminal states represents steps from a non-terminal state at time step t (ensured by the conditional probability in the function $\boldsymbol{s}_{t+1}^{\text{TG}}$) to a terminal state at $t+1$. The probabilities $p(\boldsymbol{s}_{t+1}^{\text{TG}} \in \mathcal{T}_{\text{goal}})$ and $p(\boldsymbol{s}_{t+1}^{\text{TG}} \in \mathcal{T}_{\text{fallen}})$ therefore correctly represent the probability of reaching the goal or falling exactly at time step $t+1$ respectively. In contrast, the probabilities $p(\boldsymbol{s}_{t+1}^{\text{MS}} \in \mathcal{T}_{\text{goal}})$ and $p(\boldsymbol{s}_{t+1}^{\text{MS}} \in \mathcal{T}_{\text{fallen}})$ also contain instances where $\boldsymbol{s}_t^{\text{MS}}$ already was in the respective sets. The multi-step uncertainties without the truncation of Gaussians therefore overestimate the terminal probabilities $p(\mathcal{G}_t)$ and $p(\mathcal{F}_t)$.

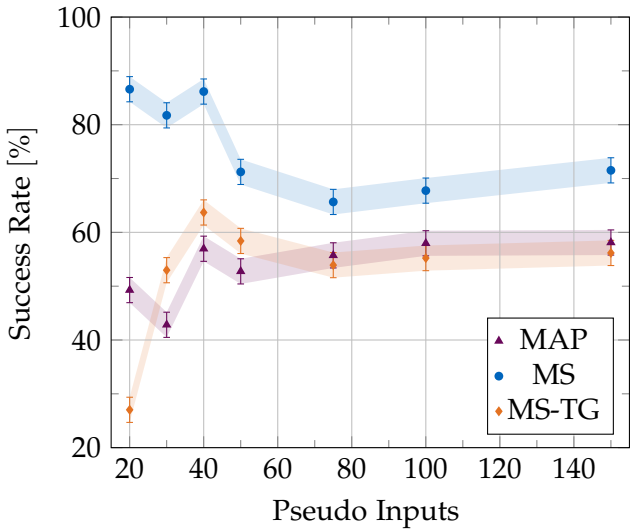


Figure 4.15: The success rate for different numbers of pseudo inputs with the standard error of the mean. Using multi step uncertainties without truncation considerably improves the performance for the less expressive models and always yields an increase of the success rate of at least 10 percent when compared to MAP-predictions. The performance of multi step uncertainties with truncation is comparable to MAP-predictions.

4.4.4 Results using Multi-Step Uncertainties

Figures 4.17 and 4.18 show trajectories generated by applying PSO-P to the one-step value functions $\hat{\mathbf{V}}_{s_0}^{\mathrm{MS}}$ and $\hat{\mathbf{V}}_{s_0}^{\mathrm{TG}}$ respectively. Both the models and starting states are the same as the ones used in figures 4.8 and 4.11, the corresponding figures for the maximum-a-posteriori and the one-step uncertainties approaches. While the trajectories look very similar to each other, they are different to the trajectories generated without informative uncertainties.

PSO-P is able to balance the bicycle longer for almost every starting state and is more successful when using multi-step uncertainties without the truncation of Gaussians. If PSO-P is successful using MAP-predictions or one-step uncertainties, the trajectories consist of a single curve to orient the bicycle directly followed by driving straight towards the goal. Using multi-step uncertainties, PSO-P avoids driving in a straight line. Instead, the bicycle's movement describes a wave, where the policy is always leaning slightly into one direction. This effect is also visible in figure 4.16, which shows all state variables over time for one successful trajectory. The policy alternates between steering heavily to the left or to the right which results in the wavy movement.

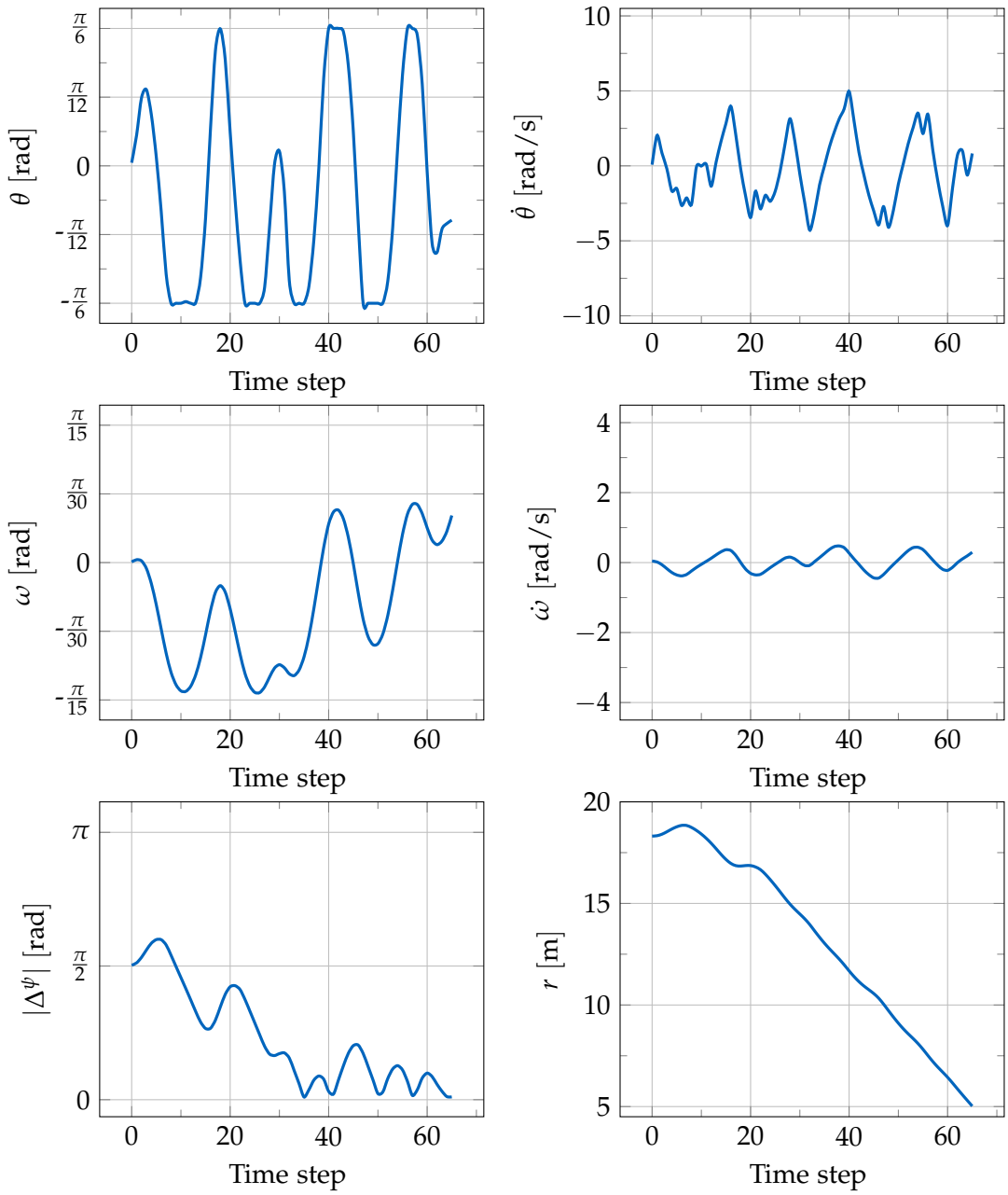


Figure 4.16: A single successful trajectory using multi step uncertainties. Equivalently to the successful MAP trajectory in figure 4.6, the bicycle starts at about 18 meters distance from the goal with an angle of about 90 degrees. Similar to the first trajectory, PSO-P is first concerned with reducing the angle towards the goal in the first thirty time steps and then keeps driving towards the goal. Using multi step uncertainties, instead of driving straight, the bicycle's trajectory describes a wave, with the controller always keeping the bicycle leaning towards one side.

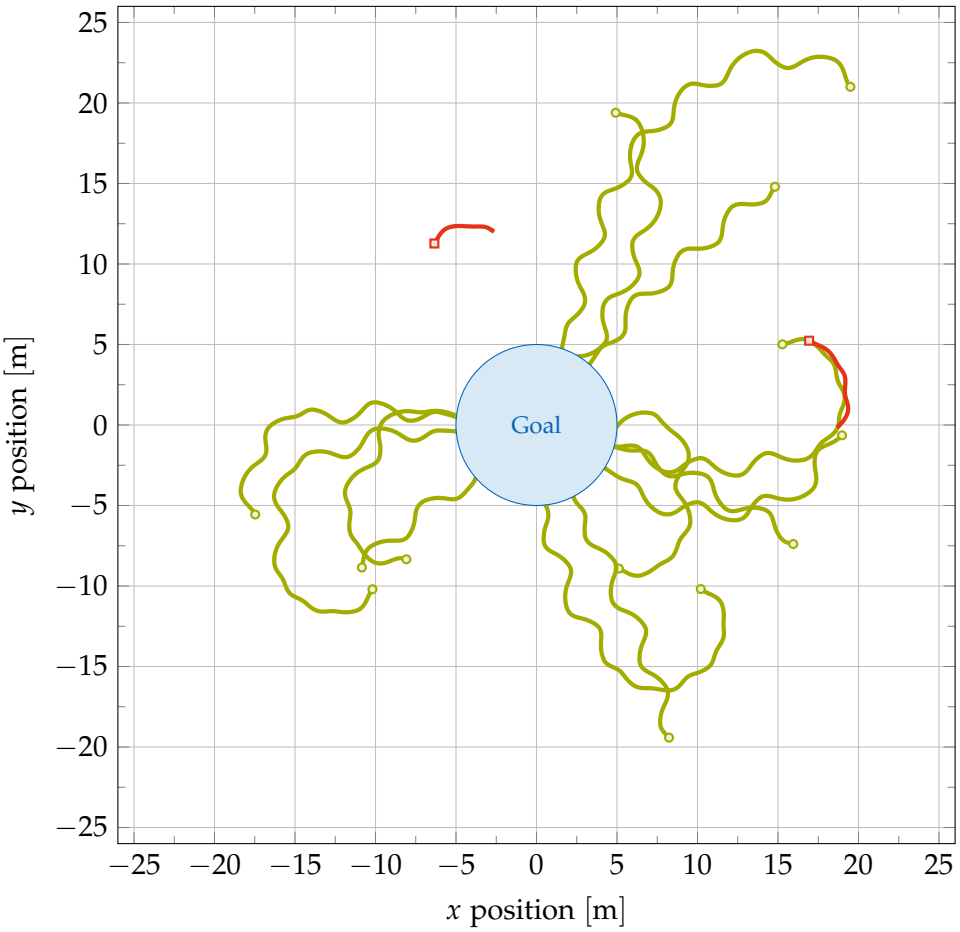


Figure 4.17: The x and y coordinates of the front tyre in representative episodes when using multi step uncertainties without truncation with marks at the starting states. Successful episodes are colored in green and failed episodes are colored in red. Incorporating multi step uncertainties can increase the success rate considerably. However, instead of driving in relatively straight lines as in figures 4.8 and 4.11, the PSO-Policy favours wavy trajectories when considering multi step uncertainties.

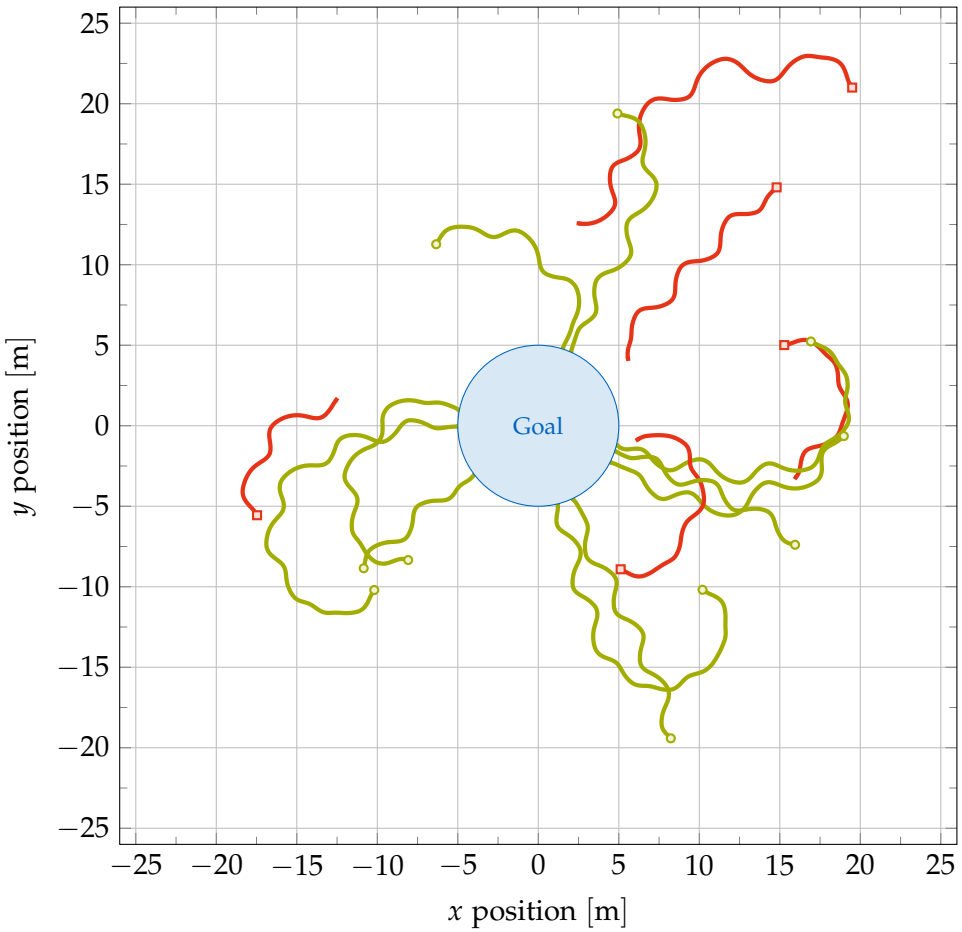


Figure 4.18: The x and y coordinates of the front tyre in representative episodes when using multi step uncertainties with truncation with marks at the starting states. Successful episodes are colored in green and failed episodes are colored in red. Trajectories generated when using the truncation of Gaussians look similar to the trajectories without truncation as showed in figure 4.17, also describing waves.

When using multi-step uncertainties, PSO-P must avoid action sequences which lead to very uncertain predictions, since for high uncertainties, the probabilities for reaching a terminal state grow. A possible explanation for the wavy movement is that driving straight is hard to predict for the transition model for two reasons. Firstly, the training data described in section 4.1.1 does not contain many samples of driving straight. The training trajectories start with a bicycle in an upright position going straight, but applying random actions cannot balance the bicycle. Since the bicycle quickly begins to fall, most transitions observed in the trajectories do not show a balanced bicycle. Similarly, randomly sampled transitions do not show an upright bicycle with high probability. Because of this, the transition models are more confident about the parts of the system where the bicycle is leaning slightly and therefore, the long-term uncertainties are smaller in this case. And secondly, a balanced bicycle is unstable and if left alone, the bicycle will start to fall down. If the bicycle is exactly upright, it is hard for the models to predict if the bicycle will fall towards the left or the right, which leads to a bimodal distribution. Since all approximations used in this thesis are unimodal, this leads to a higher uncertainty compared to states where the bicycle is leaning slightly, since in this case, the natural development of the system is easier to predict.

Not driving in straight lines means that PSO-P takes longer to reach the goal for successful trajectories compared to the other approaches. However, the statistical evaluation in figure 4.15 shows that in most cases, using multi-step uncertainties allows PSO-P to reach the goal significantly more often. The policy trades the higher reward of a quicker solution against a safer route which is more defensive and which the transition models can more easily predict. For models with a higher numer of pseudo-inputs, using multi-step uncertainties with the truncation of Gaussians becomes less successful than the other approaches however. A possible explanation for this is the fact that the truncation of Gaussians has the tendency to create uninformed predictions for later time steps, independent of the actions proposed by PSO-P. It therefore becomes harder for the optimization process to identify good action sequences. While for weaker models, this can be beneficial because is avoids the exploitation of model bias, it might prevent the exploitation of model knowledge for better models.

4.5 Discussion of the Approaches

In table 4.2, the results of the evaluation of the four approaches presented in this thesis are compared. These approaches are using maximum-a-posteriori predictions only (*MAP*) as described in section 4.2.2, using one-step uncertainties (*OS*) as described

Table 4.2: Comparison of the results of the evaluation.

Metric	MAP	OS	MS	MS-TG
Number of Trajectories	9660	9660	9660	9660
Success Rate	53.4 %	63.8 %	**75.8 %**	52.5 %
Failure Rate	46.6 %	36.2 %	**24.2 %**	47.5 %
Mean Mean-Reward	0.87	1.01	**1.17**	0.91
Median Mean-Reward	0.85	1.00	**1.24**	0.96
Mean Time to Goal	**59.9**	62.0	66.5	68.1
Median Time to Goal	**60**	**60**	63	63

in section 4.3.1 and using multi-step uncertainties, both without (*MS*) and with the truncation of Gaussians (*MS-TG*), both of which are detailed in section 4.4. All evaluations were performed according to algorithm 4.4 and table 4.1. To create the respective 9660 trajectories, the same data sets, models and starting states were used for all techniques.

Considering the model uncertainties provided by the Gaussian process transition models during planning with PSO-P can improve the generated trajectories. Both the knowledge about one-step uncertainties and multi-step uncertainties allows the policy to account for errors in the transition model and prevent the exploitation of model bias to some extent. In the bicycle benchmark, exploiting model bias mostly means planning trajectories for which the model assumes that the resulting value of ω will be equal to its maximum value as precisely as possible. In this setting, even small mistakes of the models cause the bicycle to fall down and the trajectory to fail.

Table 4.2 shows that multi-step uncertainties improve the overall success rate of PSO-P in the bicycle benchmark from about half the trajectories to about 75 percent. However, the results when using the truncation of Gaussians are comparable with only using MAP-predictions. In figure 4.19, the success rates of models with different complexities are compared. While both the one-step and multi-step approaches without the truncation of Gaussians consistently perform better than the classic alternative, the last technique only shows some improvement for less expressive models.

This trend continues when comparing the mean rewards earned throughout a trajectory. Given a series of observed (and therefore deterministic) states beginning with the

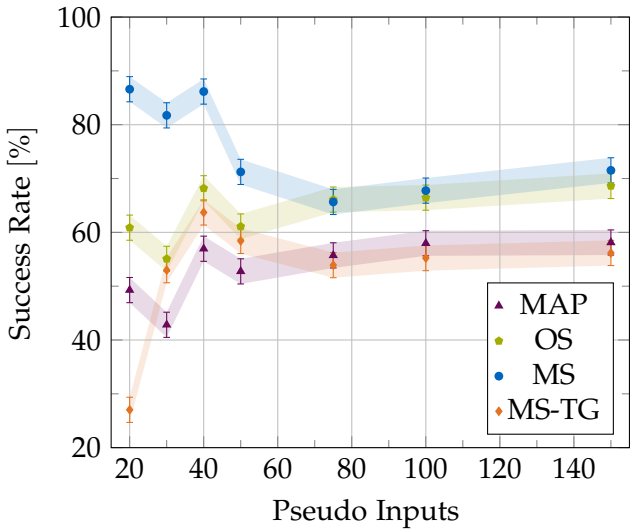


Figure 4.19: The success rate for different numbers of pseudo inputs with the standard error of the mean. Considering uncertainties during long-term planning can be beneficial. Multi step uncertainties without truncation perform much better with less expressive models when compared to one step uncertainties. For a larger number of pseudo inputs, the two approaches both perform about 10 percent better than MAP-predictions.

starting state s_0, the mean reward earned throughout a trajectory is defined as

$$\overline{\mathbf{R}}_{\text{bicycle}}(s_0, \ldots, s_T) := \frac{1}{T} \sum_{t=1}^{T} \mathbf{R}_{\text{bicycle}}(s_t). \tag{4.47}$$

Note that T is a constant defined in table 4.1 and can be larger than the length of the trajectory. In this case, the terminal state at the end of the trajectory is repeated infinitely as defined in equation (3.5). Figure 4.20 shows the mean mean reward of all trajectories created with the different classes of models. Given a set τ of trajectories, the mean mean reward is defined as

$$\overline{\overline{\mathbf{R}}}_{\text{bicycle}}(\tau) := \frac{1}{|\tau|} \sum_{i=1}^{|\tau|} \overline{\mathbf{R}}_{\text{bicycle}}(\tau_i). \tag{4.48}$$

Especially for less expressive models which are forced to generalize more, the techniques which consider model uncertainties perform better. While both OS and MS earn significantly more reward, MS-TG only slightly outperforms the baseline.

The increase in performance of MS comes with a price however. When comparing the mean number of time steps required to reach the goal in successful trajectories, it

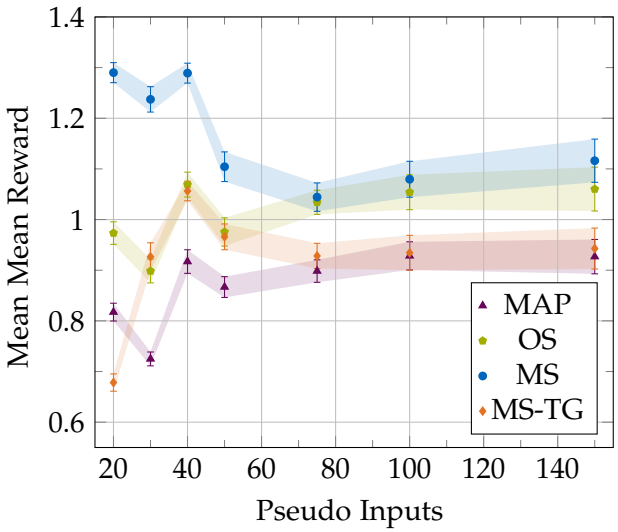


Figure 4.20: The mean mean reward for different numbers of pseudo inputs with the standard error of the prediction. The results of the mean mean reward metric are comparable to the success rate. While multi step uncertainties with truncation performs about as good as MAP-predictions, both one step and multi step uncertainties without truncation perform considerably better. Considering uncertainties is more beneficial for less expressive models.

can be seen that MAP is able to reach the goal more quickly on average than either OS or MS. Using MAP, PSO-P plans trajectories which are as short and aggressive as possible, therefore minimizing the required time. With MS, the bicycle drives much more defensive but takes more time to reach the goal, as can be seen in figure 4.17. The mean time to goal presented in table 4.2 are somewhat biased though, as PSO-P with MAP tends to fail more often for longer or more complex trajectories which require long curves. The median time to goal suggests that for similar situations, the time difference between the two approaches could be smaller.

4.6 Summary

This chapter applied Gaussian processes and PSO-P as presented in chapter 3 to the bicycle benchmark problem. It described the sampling of data sets and the design of Gaussian process models for the transition function $\mathbf{f}_{\text{bicycle}}$. Based on these models, it presented the classical solution of using deterministic maximum-a-posterior predictions to create long-term predictions. As a baseline for comparison, PSO-P was applied to

the action-value-function $\hat{\mathbf{V}}$ using a reward function $\mathbf{R}_{\text{bicycle}}$ formulated using reward shaping.

A first improvement over plain MAP-predictions was given by using the one-step uncertainties yielded by the transition models to approximate the expected reward $\mathbb{E}[\mathbf{R}_{\text{bicycle}}(\boldsymbol{s}_t)]$ of the intermediate states $\boldsymbol{s}_t$ more exactly. If the intermediate states are no longer deterministic, it is no longer clear when a predictive trajectory ends. The solution proposed in this thesis is to calculate accumulative probabilities $\mathrm{p}(\mathcal{G}_t)$ and $\mathrm{p}(\mathcal{F}_t)$ for the different termination conditions for every time step and to weight the expected reward accordingly.

Since one-step uncertainties cannot correctly model the iterated confidence of the transition model, the next approach was to propagate the state uncertainties through the nonlinear transition models using linearization. These accumulated uncertainties reduce model bias since they give an accurate measure of the correctness of long-term trajectories and reduce model bias. Using PSO-P with multi-step uncertainties results in more defensive trajectories which are more often successful.

Chapter 5

Conclusions

This thesis was concerned with incorporating information about uncertainties into reinforcement learning using ideas from Bayesian statistics. In model-based reinforcement learning, a model of the system's transition dynamics extracted from limited observations is used to make approximations about the system's future behaviour. These observations were considered to be gathered beforehand using a simple controller or random exploration instead of allowing the controller to directly interact with the system. This restriction is often imposed in industrial systems due to financial or safety concerns. Instead of relying on a single deterministic model which introduces model bias, a Bayesian framework considers distributions over models obtained by combining broad and simple prior assumptions about the transition function with their likelihood given the observed data. Since the observed data is finite, there are multiple plausible models which have to be considered. Gaussian processes make predictions about successive states by marginalizing the distribution of plausible models and can give estimations about the uncertainty of their predictions in a principled manner. Instead of learning a closed policy representation using these transition models, this thesis used the PSO-Policy, which chooses actions by directly optimizing the expected long-term reward.

The deterministic bicycle system by Randløv and Alstrøm was used as a benchmark problem to illustrate the challenges introduced by considering uncertain predictions. While Gaussian processes directly provide a measure of uncertain predictions given a certain input, there is no analytical solution to propagating already uncertain belief about the current state through the transition model to obtain the belief about the successive state. Besides considering classic deterministic predictions without any uncertainty, this thesis considered using both one-step and multi-step uncertainties. While for one-step uncertainties, long-term predictions are obtained by assuming the state to be deterministic before every application of the transition model, multi-step uncertainties approximate the posterior distribution with a Gaussian by linearizing

the transition model around the mean of the belief about the prior state, thereby propagating uncertainty through the nonlinear transition function.

In the bicycle benchmark, the controller has to balance a bicycle to keep it from falling over and navigate towards a goal region. This introduces the possibility of failure and the concept of terminal states. In contrast to deterministic long-term predictions, where for every time step the predictive state is either terminal or not, uncertain predictions can spread probability mass over both regions of the state space, which represents the belief that the trajectory could both have ended or not. Since the transition models only handle non-terminal states, this thesis proposed a technique which models and propagates this belief separately in order to correctly calculate the expected long-term reward. Additionally, this belief can be used to refine the state distributions by truncating the Gaussian to non-terminal states.

This thesis provided experimental evidence that considering model uncertainties during planning is beneficial when creating trajectories for the bicycle benchmark using PSO-P. Instead of aggressively exploiting the model and therefore falling victim to model bias, multi-step uncertainties incentivize the controller to choose actions where the transition model is confident about its predictions and where small mistakes of the models do not cause the trajectory to fail. This increased the success rate considerably when compared to classical maximum-a-posteriori predictions.

Since this thesis made few assumptions specific to the bicycle problem, this result is promising with respect to other control problems where no or limited expert knowledge is available. Uncertainty information reflecting the confidence of the transition model allow a controller to make more informed decisions during planning and to reduce risk by avoiding unknown parts of the system. On the other hand, the uncertainty in the transition model can also be used to drive exploration in settings where a controller is allowed interaction with the system, thereby reducing the amount of interaction needed.

There are multiple different directions of possible future work based on this thesis. While the uncertainties about the transition model are used as a basis for long-term predictions and provide a more reliable estimation of the expected long-term reward when compared to deterministic predictions, these uncertainties are not directly utilized during the optimization process of PSO-P. Instead, they only indirectly improve the policy by making the objective function more robust. Bayesian optimization schemes [BCF10] could be used on the uncertain reward function to choose actions which result in high expected reward with lower variance. Additionally, the computational cost of choosing actions could be reduced considerably. While PSO has to evaluate the objective function very often, Bayesian optimization schemes assume that this is expensive and

try to find an optimum with as few evaluations as possible. Alternatively, policies in closed form could be learned by maximizing the expected reward of simulated trajectories as proposed by Deisenroth in [Dei10].

The transition models operate under the assumption that the prior state is not terminal. In order to handle terminal states with uncertain long-term predictions, the probabilities of ever having reached a terminal state in the past are calculated via dynamic programming. In order to avoid predictions under wrong assumptions, this thesis proposed to truncate the state distributions to remove the probability mass in terminal states. However, performing this truncation has a detrimental effect on the performance of PSO-P in the bicycle benchmark. This could be specific to the bicycle benchmark or point to an adverse interaction between PSO-P and the truncation given high uncertainties.

In order to ensure analytical tractability, this thesis assumes Gaussian distributions of intermediate states and often approximates unknown distributions. Additionally, the true transition function is assumed to be deterministic or at least unimodal when modelling it using a Gaussian process. However, problems with higher modality are common in reinforcement learning and cannot easily be modelled using the approaches presented in this thesis. Recent work by Depeweg et al. presents an extension of long-term predictions to higher modality by Monte-Carlo-sampling trajectories using Bayesian neural networks [Dep+16], which could also be applied to Gaussian processes.

Appendix A

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).

[Cle99] Maurice Clerc. "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization". In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 3. IEEE, 1999. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=785513 (visited on 06/13/2016).

[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).

[Dep+16] Stefan Depeweg et al. "Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks". In: *arXiv:1605.07127 [cs, stat]* (May 23, 2016). arXiv: 1605.07127. URL: http://arxiv.org/abs/1605.07127 (visited on 06/06/2016).

[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).

[DRF11] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning". In: (2011). 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).

[Gau09] Carl Friedrich Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. sumtibus Frid. Perthes et IH Besser, 1809. URL: https://books.google.de/books?id=VKhu8yPcat8C (visited on 06/06/2016).

[GPy12] GPy. *GPy: A Gaussian process framework in python*. Version 1.0.7. 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).

[Her05] Ralf Herbrich. *On Gaussian expectation propagation*. Technical report, Microsoft Research Cambridge, research. microsoft. com/pubs/74554/EP. pdf, 2005. URL: http://131.107.65.14/pubs/74554/EP.pdf (visited on 05/30/2016).

[Hsu02] Feng-Hsiung Hsu. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002. URL: https://books.google.de/books?id=zV0W4729UqkC (visited on 06/07/2016).

[KF09] Jonathan Ko and Dieter Fox. "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models". In: *Autonomous Robots* 27.1 (2009), pp. 75–90. URL: http://link.springer.com/article/10.1007/s10514-009-9119-x (visited on 05/27/2016).

[Kol+10] J. Zico Kolter et al. "A Probabilistic Approach to Mixed Open-loop and Closed-loop Control, with Application to Extreme Autonomous Driving". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 839–845. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5509562 (visited on 06/07/2016).

[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: 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:1602.02523* (2016). 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 3rd Edition: The Art of Scientific Computing*. 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 (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.

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

[Sch+07] Anton Maximilian Schaefer et al. "A neural reinforcement learning approach to gas turbine control". In: *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*. IEEE, 2007, pp. 1691–1696. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4371212 (visited on 06/13/2016).

[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).

[Sil+16] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 28, 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961. URL: http://www.nature.com.eaccess.ub.tum.de/nature/journal/v529/n7587/full/nature16961.html (visited on 06/07/2016).

[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 (visited on 02/01/2016).

[SRB10] Jean-Luc R. Stevens, Philipp Rudiger, and James A. Bednar. "HoloViews: Building Complex Visualizations Easily for Reproducible Science". In: (2010). URL: http://homepages.inf.ed.ac.uk/jbednar/papers/stevens.scipy15_draft.pdf (visited on 06/06/2016).

[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).

[Tou09] Marc Toussaint. "Technical Note: Computing moments of a truncated Gaussian for EP in high-dimensions". In: (2009). URL: https://pdfs.semanticscholar.org/b88d/9fd24040ac50743bdaebb0a52af6d098fee1.pdf (visited on 05/30/2016).

[VD95] Guido Van Rossum and Fred L. Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995. URL: http://ft-sipil.unila.ac.id/dbooks/Python%20Reference%20Manual.pdf (visited on 05/27/2016).

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

Appendix B

Lists of Figures, Tables and Algorithms

List of Figures

List of Tables

List of Algorithms