Iterative Synthesis of Extremal Fields for Near-Optimal Feedback Control of Robotic Systems

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Christoph Zelch aus Darmstadt
Tag der Einreichung: 7. Februar 2024, Tag der Prüfung: 18. März 2024

1. Gutachten: Prof. Dr. Oskar von Stryk
2. Gutachten: Prof. Bruce A Conway, Ph.D. (University of Illinois Urbana-Champaign)
Darmstadt, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

Technische Universität
Darmstadt

SIM

Iterative Synthesis of Extremal Fields for Near-Optimal Feedback Control of Robotic Systems

Accepted doctoral thesis by Christoph Zelch

Date of submission: 7. Februar 2024
Date of thesis defense: 18. März 2024

Darmstadt, Technische Universität Darmstadt

Abstract

Optimal control of robots, vehicles, or industrial plants is essential, as it can provide much better, e.g., faster or more energy efficient, operation of these systems than hand-crafted control policies. Optimal control theory and (numerical) methods allow the computation of control sequences for high-dimensional dynamic systems by mathematically defining high-level goals. It is based on mathematical nonlinear dynamics models of such systems, which are often available in high quality for robots and vehicles, typically based on first principles of physics (white-box approaches). However, if the computed sequence of optimal actions is applied to a real robot, the system's states will eventually deviate from the precomputed trajectory due to inevitable model inaccuracies or unforeseen perturbations. This motivates the search for a nonlinear feedback controller that provides optimal control values not only on an optimal path but in real-time for arbitrary system states, which allows the controlled system to proceed optimally, even in case of disturbances.

Explicit formulations of optimal feedback controllers only exist for certain systems, e.g., with linear dynamics and quadratic cost functions, but not for general robots with nonlinear system dynamics. In contrast to white-box approaches based on explicit mathematical models of system dynamics, machine learning approaches based on data-driven black-box models can learn optimal feedback control policies for more general optimal control problems with nonlinear systems. However, they crucially depend on the training scenarios to collect large amounts of data and cannot generalize well beyond these, while white-box approaches are often also useful in scenarios that have not been encountered before.

The main motivation for this thesis is to investigate the combination of white-box optimal control approaches and black-box machine learning to benefit from the advantages of both concepts. The focus is on the *extremal field* approach, where a near-optimal feedback control policy is learned from a set of optimal reference trajectories, the extremal field. It uses the advantages of machine learning approaches and, at the same time, leverages the capabilities of available numerical optimal control solvers that allow the incorporation of knowledge about the problem structure and the consideration of nonlinear constraints.

In this work, the reference trajectories are computed iteratively from carefully selected start states to use the information provided by previously computed trajectories and the current feedback control policy approximation.

Because of the curse of dimensionality, it is challenging to cover high-dimensional joint spaces with sufficient training data, which makes it necessary to focus on small subspaces relevant to a specific task. To address the problem of simultaneously sufficient and efficient coverage of a relevant part of the joint space, three complementing start state selection strategies for the computation of the extremal field are developed. They utilize information from the optimal control solver, from already computed optimal trajectories and uncertainty information provided by the current approximation of the feedback policy. Further, a switch-over to a proportional-integral (PI) controller in the vicinity of a goal state is proposed to stabilize the system around this state without the need for large amounts of training data in this area.

The interpolation between the optimal trajectories to fit the feedback control policy is an essential part of the extremal field approach. It imposes specific requirements on the approximation methods formulated in this work. Two ubiquitous function approximation methods, Gaussian processes and artificial neural networks, are compared and analyzed regarding their suitability for the approximation of optimal feedback control policies with respect to these requirements.

The quality of the feedback control approximation in the extremal field approach can be degraded if data from multiple different solution clusters is merged since the approximation method may directly interpolate between different solutions and, thus, blur their structures. Current trajectory clustering approaches capable of addressing this problem are often learning-based or use pointwise Euclidean distances between two trajectories. A rule-based trajectory clustering approach is developed, which is based on the extraction of characteristic features from motion trajectories' graphs to create a compressed trajectory representation. This representation can be used in an existing string kernel-based distance measure.

The proposed methods are evaluated on different robot models with nonlinear dynamics in simulation (including a detailed nonlinear dynamics model of an industrial robot arm) and physical experiments (Furuta pendulum arm).

Zusammenfassung

Die optimale Steuerung von Robotern, Fahrzeugen oder industriellen Anlagen ist von entscheidender Bedeutung, da sie einen besseren, z.B. energieeffizienteren oder schnelleren, Betrieb dieser Systeme erlaubt, als mit manuell erstellten Kontrollstrategien möglich ist. Die Theorie und numerischen Methoden der optimalen Steuerungen erlauben die Berechnung von Steuersignalen für hochdimensionale dynamische Systeme auf Grundlage mathematisch definierter, allgemeiner Zielvorgaben. Sie basiert auf mathematischen Modellen der nichtlinearen Systemdynamiken, welche in vielen Fällen in hoher Qualität für Roboter und Fahrzeuge verfügbar sind und üblicherweise auf den Gesetzen der Technischen Mechanik, insbesondere der Mehrkörperdynamik, beruhen (*White-Box*-Verfahren). Werden die berechneten optimalen Steuersignale jedoch auf realen Systemen ausgeführt, so führen Modell-Ungenauigkeiten oder unvorhergesehene Störungen von außen über kurz oder lang zu Abweichungen von der vorausberechneten Zustandstrajektorie. Dies motiviert die Suche nach einem optimalen Regler, der optimale Steuerungen nicht nur auf einem vorausberechneten Pfad oder einer Trajektorie des Systemzustands, sondern in Echtzeit für beliebige Systemzustände berechnet, sodass das geregelte System sich auch bei Störungen optimal verhält.

Explizite Formulierungen optimaler Regler existieren nur für bestimmte Systeme, wie solchen mit linearer Dynamik und quadratischer Gütefunktion, aber nicht für beliebige Roboter mit typischerweise nichtlinearen Systemdynamiken. Im Gegensatz zu White-Box-Ansätzen, die auf expliziten mathematischen Modellen der Systemdynamik beruhen, basieren Verfahren des maschinellen Lernens auf datengestützten *Black-Box*-Modellen und können optimale Regler für allgemeinere Probleme mit nichtlinearen Systemdynamiken bestimmen. Allerdings hängen diese stark von den verwendeten Trainings-Szenarien ab, in welchen große Mengen an Trainingsdaten gesammelt werden. Sie können über diese Szenarien hinaus jedoch nicht gut verallgemeinern. White-Box-Ansätze hingegen sind oft auch in neuen Szenarien sehr gut anwendbar.

Die Hauptmotivation dieser Dissertation ist die Untersuchung der Kombination von White-Box-Verfahren basierend auf der Theorie und Numerik optimaler Steuerungen und Black-Box-Verfahren des maschinellen Lernens, um von den Vorteilen beider Verfahren zu profitieren. Der Schwerpunkt liegt dabei auf dem *Extremalfeld*-Ansatz, bei welchem ein annähernd optimaler Regler aus einer Reihe optimaler Referenztrajektorien gelernt wird. Es nutzt die Vorteile maschineller Lernverfahren und gleichzeitig die Fähigkeit numerischer Optimalsteuerungslöser, Vorwissen über die Problemstruktur bei der Berechnung der optimalen Lösung zu berücksichtigen und nichtlineare Nebenbedingungen zu beachten. In dieser Arbeit werden die Referenztrajektorien iterativ von sorgfältig ausgewählten Startzuständen berechnet, um bei der Wahl eines Startzustands Informationen von bereits berechneten Trajektorien und dem aktuellen Modell der optimalen Regelung berücksichtigen zu können.

Wegen des Fluchs der Dimensionalität (*curse of dimensionality*) ist es sehr schwierig, die hochdimensionalen Zustandsräume der Gelenkwinkel eines Roboters mit Trainingsdaten vollständig abzudecken, woraus die Notwendigkeit erwächst, sich auf kleinere Teilräume zu konzentrieren, welche für eine bestimmte Aufgabe relevant sind. Um das Problem, einen problemrelevanten Bereich des Gelenkwinkel-Raumes ausreichend und gleichzeitig effizient mit Daten abzudecken, zu adressieren, werden drei sich ergänzende Strategien zur Auswahl neuer Startzustände entwickelt. Diese verwenden Informationen von den numerischen Optimalsteuerungslösern, von bereits berechneten Trajektorien und Unsicherheits-Schätzungen, die zur aktuellen Approximation des Reglers verfügbar sind. Ferner wird der Ansatz vorgestellt, den gelernten Regler auf einen Proportional-Integral (PI)-Regler umzuschalten, sobald der Systemzustand in die Nähe des Zielzustands kommt, um das dynamische System um den Zielzustand zu stabilisieren, ohne diesen Bereich mit vielen Trainingsdaten abdecken zu müssen.

Die Interpolation zwischen den optimalen Trajektorien zur Anpassung an die Regler ist ein wesentlicher Bestandteil des Extremalfeld-Ansatzes. Sie stellt spezifische Anforderungen an die Approximationsverfahren, welche in dieser Arbeit formuliert werden. Zwei gängige Methoden zur Funktionsapproximation, Gauß-Prozesse und künstliche neuronale Netze, werden hinsichtlich ihrer Eignung für die Approximation von optimalen Reglern auf Grundlage dieser Anforderungen verglichen und analysiert.

Die Qualität der Approximation des optimalen Reglers im Extremalfeld-Ansatz kann sich verschlechtern, wenn Daten aus mehreren unterschiedlichen Lösungsclustern zusammengeführt werden, da die Approximationsmethode zwischen diesen verschiedenen Lösungen mit unterschiedlichen Strukturen direkt interpolieren und diese dadurch verwischen kann. Derzeitige Ansätze zum Clustern von Trajektorien, die zur Lösung dieses Problems verwendet werden können, basieren oft auf Lernverfahren oder verwenden punktweise euklidische Abstände zwischen zwei Trajektorien. Es wird ein regel-basierter Ansatz zum Clustern von Trajektorien entwickelt, der darauf beruht, charakteristische Merkmale aus den Graphen der Bewegungs-Trajektorien zu extrahieren, um daraus eine komprimierte Repräsentation der Trajektorie zu erstellen, welche dann in einem Distanzmaß für Zeichenketten verwendet werden kann.

Die vorgeschlagenen Methoden werden an verschiedenen Robotermodellen mit nichtlinearer Dynamik in Simulationen (einschließlich eines detaillierten nichtlinearen Dynamikmodells eines Industrieroboterarms) und physikalischen Experimenten (Furuta-Pendelarm) evaluiert.

Acknowledgment

This thesis has been written during my time as a research assistant at the Simulation, Systems Optimization and Robotics Group (SIM) at the Department of Computer Science of the Technical University of Darmstadt.

The path to the present thesis was long and I would like to take this opportunity to thank everyone who accompanied and supported me along this way and made all the valuable experiences, insights, and impressions possible.

I deeply thank Prof. Dr. Oskar von Stryk for the opportunity to work in his team and do scientific work under his supervision. I would like to thank him for his scientific guidance and advice over the years, for his extremely helpful and constructive feedback on my work, and for his continuous support. I am very grateful for his patience and encouragement. Furthermore, I wish to express my gratitude to Prof. Dr. Jan Peters and acknowledge his supervision and guidance. I would like to thank him for his repeated valuable advice that helped me progress and for the impetus he provided at multiple occasions. Finally, I am very grateful to Prof. Bruce A Conway, Ph.D., for agreeing to become my second referee. I thank him for his interest in my work, his friendly and encouraging feedback, his valuable suggestions during the final phase of my research, and his detailed comments on the submitted version of the thesis, which helped me improve the present work.

I had the pleasure of working in a great team and would like to thank all my colleagues for the exciting time together at SIM, for the enlightening discussions and the scientific exchange. In particular, I want to thank Dr. Jérôme Kirchhoff for all the scientific advice and discussions, for his competent feedback and his encouragement. I want to express my gratitude to the IAS department for the opportunity to work in their laboratory to use their Furuta pendulum for my experiments. In particular, I would like to thank Dr. Fabio Muratore, who helped me getting started with the Furuta pendulum.

I would like to warmly thank Jeremy for all the discussions about machine learning and robotics, for sharing his experience with me, and his feedback on my ideas and thoughts.

Further, I am extremely grateful to my sister Ines for all her support and help. I would like to thank her for all the invaluable discussions, for her detailed and constructive feedback, and for proofreading large parts of this thesis.

Most of all, I wish to deeply thank my parents for all their enormous support, motivation and their repeated encouragement that helped me continue step by step. Without their constant moral support, I could not have finished this work.

Contents

List of Figures

List of Tables

List of Abbreviations

DDP differential dynamic programming

DTW dynamic time warping

FITC fully independent training conditional

GHJB generalized Hamilton-Jacobi-Bellman

GP Gaussian process

HJB Hamilton-Jacobi-Bellman

LHS Latin hypercube sampling

LQR linear-quadratic regulator

LWPR locally weighted projection regression

ML machine learning

MLP multi-layer perceptron

MPC model predictive control

NMPC nonlinear model predictive control

NMSE normalized mean squared error

NN neural network

OCP optimal control problem

ODE ordinary differential equation

PDE partial differential equation

PI proportional-integral

RL reinforcement learning

1. Introduction

Optimal control of robots, vehicles, or industrial plants offers significant advantages as it allows one to operate them faster, more energy efficient and in some cases even more safely than with hand-crafted control trajectories. It is commonly assumed that regular periodic human locomotion is optimal [119, 171], such that humanoid robots intended to provide human-like behavior should also perform at least near-optimal locomotion. Further, energy-efficient trajectory planning makes many aerospace missions feasible in the first place due to very limited payload for fuel [63].

Sophisticated mathematical models of the nonlinear dynamics behavior of such systems are often available for robots and vehicles describing the evolution of their state. These models are typically based on first principles of physics and formulated as systems of nonlinear ordinary differential equations (ODEs) [69]. Optimal control theory [30, 170, 104, 22] provides a basis for numerical methods that utilize these white-box system dynamics models to compute optimal open-loop state and control trajectories [174, 21], even for large-scale nonlinear robot dynamics models, nonlinear cost functions, and nonlinear state and control constraints.

However, in practical applications, model inaccuracies and external disturbances or small deviations from the measured start state are inevitable on real systems and lead to deviations from the precomputed trajectory. Controlling the system away from the precomputed optimal path requires recomputation of the optimal trajectory, which in most cases takes too long to be performed during the robot's motion execution. The deviations thus must be dealt with in a non-optimal manner, e.g., by real-time trajectory tracking controllers that try to bring the system state back onto the precomputed path. It is desirable to have some feedback controller that provides optimal control (with respect to the current task) even away from some precomputed time-dependent optimal trajectory. Explicit formulations of optimal feedback controllers only exist for specific systems, e.g., with linear dynamics and quadratic cost functions (linear-quadratic regulator (LQR)), but not for systems with nonlinear dynamics or general cost functions. In contrast to white-box approaches, machine learning (ML) (and in particular reinforcement learning (RL)) approaches based on black-box models are able to learn optimal feedback control policies for general optimal control problems with nonlinear systems. However, they crucially depend on the training scenarios and cannot well generalize beyond these. The advantage of white-box models is that they can also be used in scenarios that were previously unknown. This ability is required to train black-box ML-based models.

The main motivation for this thesis is to investigate how white-box and black-box approaches can be brought together to benefit from the advantages of both concepts. Optimal control theory and numerical methods are combined with machine learning to leverage the information provided by model-based numerical trajectory optimization methods that incorporate prior knowledge about the system dynamics and its constraints and compute near-optimal feedback control policies based on general nonlinear optimal control problems for robotic systems.

A central challenge that all existing approaches that aim to calculate a global feedback control policy have to face was coined by Bellman in 1957 as the “curse of dimensionality” [18]. It is based on the fact that the volume of a hypercube in multi-dimensional vector spaces grows exponentially with increasing dimensionality. Consequently, sampling in these spaces to cover the full volume becomes increasingly difficult or, at some point, impossible, as the computational resources of computers are limited and quickly exhausted. This requires careful design of approaches to compute approximate feedback controllers such that they can be scaled to more interesting but high-dimensional problems.

This work is centered around an approach called “extremal field” [64] or “trajectory library” [172] approach to approximate a near-optimal feedback control policy for general nonlinear robotic systems. Information from a set of optimal trajectories computed with numerical optimal control solvers is used to iteratively learn an approximation of the feedback control. This approach allows to utilize valuable information provided by state-of-the-art numerical trajectory optimization methods, which can be highly beneficial for the learning process. Optimal control solvers based on direct collocation, as used in this thesis, can deal with highly nonlinear dynamics and path constraints.

The chapters of this work focus on the different key steps of the extremal field approach, namely the computation of optimal trajectories, the approximation of the feedback controller and the selection of start states for new optimal trajectories. In the large spaces of high-dimensional dynamic systems, it is impracticable to cover the full state space with trajectories (“curse of dimensionality”). The extremal field approach allows to focus on a small subspace that is relevant for one or more tasks and cover this subspace with sufficient density using as few trajectories as possible. The state space coverage is largely determined by the start states of the trajectories; a good strategy to select start states is thus essential to improve the scalability of the approach to problems in higher dimensions. Feedback control policies are highly nonlinear and challenging to fit since training data is unevenly distributed in space due to the sampling from trajectories. This requires careful selection and tuning of the function approximation method used to approximate the feedback control policies. The numerically computed optimal state and control trajectories provide the training data for the feedback control approximation. Consequently, they are a crucial part of the extremal field approach, as the quality of the available training data strongly affects the success of the function approximation. Merkt et al. [117] consider "learning as a way of compressing and generalizing across optimal solution samples" in their work. They note that "discontinuity and multimodality can greatly impact the quality of prediction obtained using function approximation as regressors smooth across the boundaries between clusters or modalities." In this context, multimodality occurs when "multiple equally optimal solutions to a problem exist." Consequently, multimodality potentially decreases the accuracy of the approximated control policy for some optimal control problems (OCPs). It is thus worth to take multimodality into account and analyze its effect on the approximated control policy.

While the extremal field approach can be applied in many different areas, this thesis focuses on the application to robotic systems. The difficulties that arise from those systems are high nonlinearity in the system dynamics and tight real-time requirements. The nonlinear dynamics arise from forces and moments acting along a rigid kinematic chain of usually rotary joints that move the limbs in most modern robots. Fast and dynamic movements of robots are necessary, e.g., for periodic transport motions of industrial robots, stable walking of legged robots, performing tasks in reasonable speed, or immediate reactions to changing environments. They require the feedback control values to be available within a few milliseconds.

1.1. Contribution

The curse of dimensionality makes it necessary to carefully select start states for the computation of optimal trajectories from which the training samples for the feedback control approximation are taken to provide good coverage of the system's state space. The selection of start states has been identified as "main difficulty" by Zhong et al. [217]. However, only very few works (for example, [7, 91]) attempt to consider iterative approaches to generate reasonable new start states beyond sampling strategies. To address the problem of state space coverage, this work proposes a combination of three new start state selection methods for trajectories and random sampling around the nominal start state to guide exploration of the relevant state space. The combination of these approaches uses information provided by numerical optimal control solvers and by the already computed state trajectories. The method uses the uncertainty information provided by the near-optimal control policy approximation computed so far to select the next start state from a set of candidates.

The area around the goal state (or final state) is also crucial for successfully performed motions and must be well approximated. However, high-dimensional state spaces are difficult to cover densely. In contrast to agents in reinforcement learning approaches, solutions of optimal control solvers do not explore the vicinity of the goal state but approach it directly. In this thesis, it is proposed to combine the learned feedback control with a stabilizing proportional-integral (PI) controller around the goal state when this goal state is fixed. This removes the necessity to ensure good coverage of the area around the goal state, which would otherwise require careful and dense data collection in this part of the state space. However, the PI controller is suboptimal with respect to the objective since it does not consider the cost function. For this reason, this controller is used only for stabilization and is applied only in a small area around the goal state.

In the iterative extremal field approach considered in this thesis, an approximation of the global optimal feedback control policy is learned. Thus, it depends on the function approximation method chosen to represent the policy. Similar approaches that learn a feedback controller typically rely on a particular method without further discussing how well this approximation approach suits their problem and how their results change when that approach is replaced. In this work, several requirements that the fitting of near-optimal feedback control policies imposes on the approximation methods are formulated. The suitability and applicability of two widely used function approximation methods, Gaussian processes (GPs) and neural networks (NNs), are analyzed and compared regarding these requirements. While there are many general comparisons of GPs and NNs, the author of this thesis is not aware of any comparison that respects the specific requirements of feedback control policy approximations and focuses on these problems. The comparison also includes experiments on a real-world Quanser Furuta pendulum [150]. Furthermore, some extensions of GPs and NNs are considered to fulfill the formulated requirements better, and insights into the selection of favorable hyperparameters are presented.

As noted by Merkt et al. [117], multimodality potentially decreases the accuracy of the approximated control policy for some OCPs. One way to deal with it is to cluster all trajectories computed to provide data for the control policy learner. The trajectories that do not belong to one selected cluster are removed and not considered for training. This eliminates the multimodality in the training data. A novel rule-based distance measure for clustering motion trajectories is developed in this thesis to separate the solutions provided by the optimal control solver into clusters. The novelty of the approach is the extraction of feature data from the trajectory graphs to build a compressed representation of the trajectories in the form of a feature sequence. These sequences can then be used in a string kernel method by Elzinga [52, 54, 53] to compute the distances between the motion trajectories. Advantages of the presented feature-based approach compared to the widely used dynamic time warping (DTW) are demonstrated. Merkt et al. [117] show that multimodality impacts the warm-start strategies using learned feedback policies based on NNs. In this thesis, the effect of using multimodal training data on the performance of a feedback policy based on GPs is further investigated when it is used directly to control a real system.

1.2. Outline

In the following, the structure of this thesis will be outlined. Chapter 2 presents the mathematical formulation of the optimal control problem considered in this thesis and introduces the notation and terminology used. It concludes with an extensive overview of the relevant state of research and related work divided into several subsections to structure it with respect to the different approaches and applications.

The approach of learning a near-optimal feedback controller from a set of optimal trajectories considered in this thesis consists of four main steps, presented in Chapter 3. This chapter also introduces the first of three novel start state selection methods based on estimates of adjoint variables provided by the optimal control solver. The benefit of this selection method compared to naive random sampling is analyzed. Three of the four main steps are more complex and will be examined in more detail in the following chapters.

In Chapter 4, two complementing start state selection methods based on the state sensitivity and the simulation error are proposed and used together with the adjoint-based approach from Chapter 3 and random sampling around the start state. This chapter presents a continuation and further development of the main approach from Chapter 3. In addition, a switch-over to a stabilizing PI controller when the system approaches this goal state is proposed for problems with a fixed goal state.

The focus of Chapter 5 is on the representation of the near-optimal feedback control policy. It contains a systematic comparison of the two most important function approximation methods NNs and GPs regarding their suitability to fit feedback control policies. Experiments on a real-world Furuta pendulum are presented.

Chapter 6 is centered around the problem of multimodality that may occur during the optimal trajectory computation step. This problem can be solved with trajectory clustering. As a consequence, this chapter deals with the computation of distances between trajectories that are required for clustering and analyzes the effect of the separation of clusters on learning an accurate feedback control policy.

The results of the previous chapters are reviewed in Chapter 7; the findings and contributions of the given work are summarized. Beyond this, this chapter addresses open problems and further work related to the results achieved in this thesis.

2. Optimal Control and Optimal Feedback Control

Optimal control problems (OCPs) are formulated in different variations in the literature. In the next section, the mathematical notations and the formulation of the OCP considered in this thesis is given. It is followed by a brief review of the most important numerical methods for trajectory optimization. The chapter concludes with an extensive review of the state-of-the-art methods for the computation of near-optimal feedback controllers for dynamic systems.

2.1. Problem Statement

According to Siciliano and Khatib [163], robotic mechanisms are defined as “systems of rigid bodies connected by joints”, which can be passive (unactuated) or actuated. Kinematic joints restrict the relative motion between two bodies, the most commonly used type of joints is a hinge (revolute joint) that allows a rotational motion around a single axis. In this thesis, all robotic systems consist of revolute joints only. Given the physical description of a robotic system, the position and orientation of all its bodies in space (relative to some reference coordinate system) can be derived from the current configuration of all joints. The joint configuration q and its time derivative $\dot{q}$ are aggregated in the state vector

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \in \mathbb{R}^{n_x} \tag{2.1}$$

with $2n_q = n_x$. The control commands, often given as torques, are represented by $u \in \mathbb{R}^{n_u}$ with $n_u \leq n_q$. If the inequality is strict, the robotic system is called *underactuated*. The system dynamics give the behavior of a robotic system in time, depending on its current state and the control applied over time. It can be described by an ordinary differential equation (ODE)

$$\dot{x}(t) = f(x(t), u(t), t), \tag{2.2}$$

where $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \to \mathbb{R}^{n_x}$. The dependency of x and u on time is mostly omitted in the following for brevity. In this work, only autonomous models that are not explicitly depending on time are considered, which reduces the ODE to $\dot{x} = f(x, u)$. The function f may or may not incorporate detailed motor models, environment information etc. The torque that can be applied to a joint is usually constrained, which implies upper and lower bounds on the control vector. Physical limitations of the joint position, together with artificial bounds on the allowed joint velocity, are also reflected in the upper and lower bounds

$$x_{\text{min}} \leq x(t) \leq x_{\text{max}} \tag{2.3a}$$

$$u_{\text{min}} \leq u(t) \leq u_{\text{max}}. \tag{2.3b}$$

More complex limits on the state or control can be modeled as inequality constraints:

$$g(x(t), u(t)) \leq 0. \tag{2.4}$$

The high-level description of a control task as presented in [30] is given by an objective functional

$$J[u] := \Phi(x(t_f), t_f) + \int_0^{t_f} L(x(t), u(t), t)\, dt \tag{2.5}$$

that consists of a running cost L (the Lagrange term) that accumulates in time and a final cost Φ (the Mayer term) that prices the final state of the controlled system. The terminal time t_f can be fixed to some positive value or free. The initial state of the dynamic system is given as $x(0) = x_0$. The system state at the terminal time $x(t_f)$ may be fixed to some state vector $x(t_f) = x_f$. In the scientific literature, this state is referred to as *final state* [30, 22, 104], *terminal state* [30, 187] or *goal state* [187, 217, 10, 78]; the latter is mainly used in this thesis. The initial and final state x_0 and x_f must satisfy the state bounds (2.3a) and, if existent, the inequality constraint (2.4).

The goal of an optimal control problem is to find a control function $u(t)$ fulfilling (2.3b) that minimizes the objective function (2.5) such that the resulting progression of $x(t)$ in time starting at x_0 determined by (2.2) satisfies the constraints (2.3a), (2.4) and ends at time t_f in an optimally fixed state x_f.

2.1.1. Numerical Methods for Trajectory Optimization

Numerical methods to solve the OCP that is described in the previous section can be divided into dynamic programming, indirect and direct methods [25]. Dynamic programming [18] provides the global solution of an OCP but is strongly subject to the curse of dimensionality. It is not considered further in this section.

Using indirect methods to solve an OCP requires the formulation of its first-order necessary conditions (similar to the Karush-Kuhn-Tucker conditions for constrained nonlinear optimization problems), which are transcribed into a two-point boundary value problem (TPBVP) that needs to be solved numerically [174, 36]. The use of indirect methods requires expert knowledge about optimal control theory and the solution structure of the problem (the so-called *switching structure* incurred by constraints). Further, it requires a good initial guess of the solution's state and adjoint variables (also known as co-states). This makes indirect methods challenging to be applied to real-world problems and may explain why they are today less widely used than direct methods. However, they provide more accurate results than direct methods [174].

Direct methods (aka. direct transcription methods) discretize the time of the OCP to transform it into a nonlinear optimization problem [36]. Adjoint variables are not required for this approach, so finding an initial guess for them is unnecessary. Furthermore, the switching structure of the solution does not need to be known beforehand either. This makes them far more easy to handle than indirect methods. Direct methods, however, depend on an initial guess for the state and control variables. For both direct and indirect methods, tests for sufficient conditions are often omitted and the solution is assumed to be a local minimum [36].

Algorithms that are commonly used in the direct and indirect approaches are collocation and shooting methods as well as heuristic approaches. Shooting and multiple shooting methods exist for the initial value problem (direct approach) as well as for TPBVPs (indirect method) [22]. Multiple shooting is much more stable than single shooting (where small changes in the initial conditions can cause large changes in the final conditions, the "'tail wagging the dog' problem" [22]), such that it is usually preferred. Multiple shooting methods use parameterized controls and a start state to perform explicit numerical integration of the ODE on subintervals of the time interval. The parameterization and free start states are corrected iteratively to reduce violations of the terminal constraints (if existent) and the linkage constraints, which ensure a continuous solution on the full time interval, until all constraints are satisfied [25, 122].

In addition to the parameterization of the control variable, collocation methods also require the parameterization of the state. The state and control variables are typically approximated using piecewise low-order polynomials [173, 22]. The integration of the ODE is done implicitely by ensuring that the differential equation holds at predefined collocation points on the time interval [173]. The parameters that satisfy all boundary conditions of the OCP as well as the OCP path constraints and ODE conditions on a fixed time grid spanned by the collocation points are determined using a constrained nonlinear optimization problem, for which sophisticated solvers exist (e.g., SNOPT [66] or IPOPT [199]). Alternatively, orthogonal collocation uses Legendre or Chebyshev polynomials, it is known as the Gauss-pseudospectral method [36, 20, 55]. Collocation methods can be used for both indirect and direct approaches, they are, however, more commonly used in transcription methods, where they are termed "direct collocation methods" [36, 174].

The use of evolutionary algorithms and metaheuristics to solve OCPs is an alternative to the direct and indirect methods described above. Because of their stochastic nature, these algorithms are less likely to be trapped in local minima. Further, they do not need an initial guess (it is chosen randomly) and they can determine beneficial parametrizations of the state and control variables or optimal switching structures of the solution [36, 37]. Notable disadvantages are that they do not provide optimality guarantees and that path constraints can only be considered using penalty functions, such that their accuracy is lower than collocation or multiple shooting methods [36]. They are thus well suited to be combined with the methods mentioned above [36].

2.1.2. The Hamilton-Jacobi-Bellman Equation and LQR

The cost-to-go at some state $\tilde{x}$ at time $\tilde{t}$ is the sum of Mayer and Lagrange term cost that will be accumulated on an optimal path starting at the current state and time $\tilde{x}(\tilde{t})$.

The value function (aka. optimal return function)

$$V(\tilde{x},\tilde{t}) := \min_{u(t)} J[u] = \min_{u(t)} \left\{ \varphi(x(t_f), t_f) + \int_{\tilde{t}}^{t_f} L(x(t), u(t), t)\, dt \right\} \tag{2.6}$$

provides for a given state the cost of an optimal path from this state to the goal state. Using Taylor expansion, the *Hamilton-Jacobi-Bellman (HJB) equation*

$$-\frac{\partial V(x(t),t)}{\partial t} = \min_{u^*(t)} \left\{ L(x(t), u(t), t) + \frac{\partial V(x(t),t)}{\partial x} f(x(t), u(t), t) \right\} \tag{2.7}$$

can be derived (for more details see Bryson and Ho [30], Chapter 4.2). The HJB equation is a first-order nonlinear partial differential equation (PDE), which is known to be difficult to solve, because its solution, the value function, may be nondifferentiable [33] and because the curse of dimensionality makes the problem “intractable for systems with all but modest dimension” [78].

If the value function is known, the globally optimal control can be easily computed at every state of the system by minimizing the right-hand side of the HJB [49]

$$u^*(x(t),t) = \arg\min_{u^*(t)} \left\{ L(x(t),u(t),t) + \frac{\partial V(x(t),t)}{\partial x} f(x(t),u(t),t) \right\}, \tag{2.8}$$

which is the state-dependent optimal feedback control.

Linear systems with quadratic performance criterion

$$\begin{aligned} J[u] &= \frac{1}{2} x(t_f)^T S x(t_f) + \frac{1}{2} \int_0^{t_f} x^T A x + u^T B u \, dt \\ \dot{x} &= F(t)x(t) + G(t)u(t), \end{aligned} \tag{2.9}$$

where S and A are positive semi-definite matrices and B is a positive definite matrix, are known to have a closed form of the optimal feedback control. Its HJB equation leads to a matrix Riccati equation that can be solved numerically, providing a matrix $P(t)$. The optimal feedback control for the linear-quadratic OCP (2.9) is then given by

$$u^*(t) = -\left[B(t)^{-1} G(t)^T P(t)\right] x(t). \tag{2.10}$$

For more details, the reader is referred to Bryson and Ho [30].

2.1.3. Optimal Feedback Control

The solution of the OCP is a time-dependent control function $u(t)$. A deviation from the computed optimal control path leads to suboptimal performance or even infeasible trajectories. In real-world scenarios, such deviations are practically inevitable, as there is a variety of possible internal and external perturbations. Models of mechanical systems are built using assumptions and simplifications, such as ignoring backlash in the gears of joints [163] or neglecting uncertainties in the state estimation [112], modeling actuators as perfect torque sources not subject to bandwidth limits [72] or assuming the absence of delays [22]. Further, the parameterization or even the exact physical description of higher order dynamic effects like the phenomenon of friction in the joints [4, 26], the wear and tear of mechanical components [93], or the influence of temperature [204] is often unknown and thus neglected in the model. Besides the fact that parts of the environment or some of its properties may be completely unknown, the positions of known obstacles are typically subject to measurement errors and even the exact state of the controlled system is often uncertain due to sensor noise [163]. Examples of unforeseen events are the breaking of a propeller blade for a flying quadrotor [79], the appearance of an obstacle in the planned path of a moving arm [77] or ground robot, pushes during execution of the trajectory [107], or unpredicted changes in the weather conditions during the flight of an airplane [83]. While numerical optimal control methods are readily available to solve OCPs for high-dimensional nonlinear dynamic systems with nonlinear constraints, the problem of finding an optimal feedback control policy for general problems is hard.

The main approach pursued in this thesis to approximate a near-optimal feedback control policy is called extremal field approach. It is based on the fact that, in the absence of perturbations, the control commands on a globally optimal trajectory match the optimal feedback control commands along this path. The idea of the extremal field approach is to approximate from multiple optimal trajectories the optimal feedback control in the vicinity of these trajectories.

2.2. Overview of Related Approaches to the Optimal Feedback Control Problem

The variety of approaches is large and spans several major fields of research, such as optimal control theory, reinforcement learning, numerics, optimization, machine learning, robotics and control theory. Moreover, the problem formulations that are investigated in the published works differ notably: deterministic and stochastic, state-discrete and state-continuous, time-discrete and time-continuous, linear, control-affine and nonlinear dynamics, finite-time and discounted infinite-time horizon problems, quadratic and nonlinear cost functions, and different kinds of constraints.

Considering the range of the investigated problems and the amount of work that has been dedicated to the respective fields, the following outline of existing work relevant to the problem of feedback control approximation for robotic systems is therefore by no means exhaustive. This overview is intended to provide an overview of larger classes of methods, excluding the extremal field approach for which the existing work is discussed in more detail in Section 3.1. The aim is to illustrate the variety of approaches, to highlight important works, and to discuss the strengths and weaknesses of the various approaches in relation to the extremal field approach on which this thesis is based.

2.2.1. Approaches based on the Hamilton-Jacobi-Bellman Equation

As outlined in Section 2.1.2, the solution of the HJB equation gives the global feedback control of the corresponding OCP. For this reason, repeated attempts have been made to solve this PDE numerically. A brief survey of methods is given by Horowitz in his doctoral thesis [78].

For general nonlinear problems, a solution to (2.7) is not known. Beard et al. [14] and others extending their work consider a simplified formulation of the OCP with control-affine dynamics and quadratic weighted control penalty. For this problem, the optimal feedback control can be explicitly formulated if the value function $V(x)$ as defined in (2.6) is known [139]. Beard et al. [14] propose to solve the generalized Hamilton-Jacobi-Bellman (GHJB) equation, which is linear and thus easier to solve than the HJB. They show that a solution to the original HJB equation can be found by iteratively computing the solution of the GHJB equation and the respective approximation of the feedback control. Since then, this method has been improved, e.g., by Park and Tsiotras [139] who propose to use wavelets instead of polynomials as basis functions.

To reduce the computational effort of solving the HJB equation in high-dimensional spaces, one can replace fixed grids with Monte Carlo sampled points. Chilan and Conway [33] propose to compute the HJB solution on a quasi-Monte Carlo grid, which is more efficient than simple Monte Carlo sampled grids since it covers a space more uniformly. They use universal kriging to interpolate between the irregular grid points produced by quasi-Monte Carlo sampling.

2.2.2. Power Series Approaches

Several approaches that are based on power-series expansions to solve the HJB equation have been proposed, based on different semi-analytical methods. The underlying idea is "based on considering the system as a perturbation of a linear system, with the control being an extension of the linear control" [17]. For example, the Adomian decomposition method has been proposed by Fakharian et al. [56]. Nik et al. [131] use He's homotopy perturbation method to solve the partial differential equation. Jafari et al. [82] show that many of the approaches are equivalent, meaning that they lead to the same iterative formula. See the references in [82] for more methods on how to solve the HJB equation using power series expansions.

Vadali and Sharma [191] approximate the cost function with a power-series expansion and present a method to determine optimal feedback control laws for nonlinear systems. They demonstrate their method on one to three dimensional systems. However, they note that "[c]onvergence of the series solution is not guaranteed for highly nonlinear systems."

2.2.3. Compensation of Local Disturbances

Bryson and Ho [30] show in Chapter 6 how to compute a neighboring optimal feedback controller around a nominal trajectory. For some perturbations, the system dynamics are linearized along the nominal path. A first order expansion of the system dynamics and a second order expansion of the performance criterion around the nominal path, with respect to the perturbations, is called the accessory minimum problem [28]. This is exactly the linear-quadratic problem for which a feedback control is known (see (2.9) and (2.10) in Subsection 2.2.1) [28, 30, 142]. This leads to a feedback control $\hat{u}(t) = u^*(t) + \delta u(t)$ near the nominal path that is applicable for small perturbations, where $u^*(t)$ is the open-loop control of the nominal trajectory and $\delta u(t)$ optimizes the accessory minimum problem that considers the perturbations. Many of the computational work can be precomputed, such that the local feedback control can be computed very efficiently online [142]. However, this feedback control is only near-optimal in the immediate vicinity of the nominal trajectory, where the first- and second-order Taylor approximation is sufficiently accurate. For highly nonlinear problems, this region can be very small.

Büskens and Maurer [31] use sensitivity analysis to get the sensitivity differentials of an optimal trajectory to be able to approximate in real-time the solution of a perturbed OCP. In the problem they consider, the OCP is parameterized and solved for nominal parameters p_0. For this solution y, the sensitivity differentials $\partial y(t, p_0)/\partial p$ are computed. They can then be used online to linearly approximate the perturbed solution locally using a first order Taylor expansion. All disturbances and perturbations of the optimal trajectory must be represented as perturbations of the nominal parameters. The OCP may also contain inequality constraints and constrained final state. However, this method can only deal with local perturbations. The authors apply their method to control the industrial Manutec r3 robot arm.

Diehl and colleagues [47, 45, 48, 46] develop a real-time approach for nonlinear model predictive control (MPC) that is able to provide a near-optimal control command very quickly after a new state estimation becomes available, which allows a fast reaction to respond to disturbances. They propose to prepare the feedback control computation between two steps before the new system state is known. When the new state becomes available, the control resulting from one optimization iteration is used to control the system. Then, before the next system state is given, the full solution of the optimization problem is computed and used to prepare the computation of the next step.

Jardin and Bryson [84] use a *neighboring optimal control* approach, similar to a linear-quadratic regulator (LQR), to compute a time-varying feedback control around a linearized nominal trajectory. The authors comment in [83] that this method "may produce a suboptimal result in the vicinity of a locally-optimal solution". This is, for example, the case if the problem is significantly nonlinear or perturbations become too large.

LQR can be used to create a local region of stability around a trajectory. Tedrake et al. [184] "sacrifice direct attempts at obtaining optimal feedback policies in favor of [...] stronger guarantees of getting to the goal". For each optimal trajectory, they compute a time-varying LQR feedback stabilization around this trajectory and estimate its region of attraction by construction of a valid Lyapunov function (using sums-of-squares optimization) in a maximal tube ("funnel"). Their approach is to iteratively sample random start states for new trajectories. If this sample point is outside a stabilized region around an existing trajectory, they compute a new optimal trajectory and its respective funnel until they connect to an already computed funnel. With this strategy, they aim at covering the full relevant state space.

2.2.4. Differential Dynamic Programming and Related Approaches

Using a quadratic model of the value function, differential dynamic programming (DDP) is a second-order method that iteratively improves a local trajectory. According to Todorov and Li, DDP is "an ideal blend of the advantages of local and global methods" [188]. It is the basis or inspiration of many other approaches.

DDP has been proposed by Jacobson and Mayne [116, 81]. It is based on dynamic programming "applied within a 'tube' around the current trajectory" [188]. The approach "maintains a local quadratic model of the value function along the current best trajectory [...] as well as a local linear model of the corresponding policy" [6] and iteratively improves this locally optimal trajectory. It has second order convergence [188]. DDP provides a local feedback control policy along the computed optimal trajectory. However, it is unable to incorporate control constraints and requires a quadratic cost function [188].

Tassa and Smart [182] approximate the value function along a trajectory using a library of locally optimal linear controllers. They consider problems with discrete-time dynamics, but continuous states and actions. The small local area around an optimal trajectory computed using DDP in which the feedback control can be approximated is expanded using a library of multiple optimal trajectories. The selection of the local model at a given state is done using the nearest-neighbor approach. The DDP algorithm is modified to compute the quadratic model of the value function using the information from known points in the neighborhood from other trajectories in the library. This approach requires to solve a small system of linear equations. Further, they modify the standard DDP approach to use a receding horizon scheme similar to MPC. They apply their method to d-link swimmers with 14 to 34 dimensions in a viscous liquid on a planar surface that need to reach some target point.

Todorov and Li present modified versions of the DDP approach, termed iterative linear-quadratic regulator design (iLQR) [105] and iterative linear-quadratic Gaussian (iLQG) [188]. They are based on “iterative linearization of the nonlinear dynamics around the current trajectory” [188], which is iteratively improved by solving modified LQR problems along the trajectory. This way, the method provides locally valid feedback gains along the trajectory. iLQG is an improvement of iLQR that is applicable for stochastic nonlinear system models and arbitrary cost functions and can also solve problems with control constraints.

Since iLQG is “insensitive to additive noise” [188], Tassa and Todorov propose with iterative local dynamic programming (iLDP) a local method that is tailored for nonlinear stochastic noisy problems with non-quadratic cost functions [189]. At each time step, states around the nominal trajectory are sampled and the local value is approximated for each sample. The approximation of the trajectory’s value at the current time step with its first and second derivative are approximated by fitting a function approximation to the values at the sampled states.

Howell, Fu and Manchester [79] have taken a similar approach with Direct Policy Optimization (DPO), which aims to solve stochastic optimal control problems and make them robust to perturbations and unmodeled dynamics around a nominal trajectory. They sample a small number of start states around a nominal start state and solve the resulting OCP (with noisy state dynamics) with a direct collocation method. The solution of the stochastic nominal OCP together with a local feedback control policy is then estimated from these trajectories using unscented transform. In contrast to iLDP, which is a rollout-based approach where samples are taken around each time step, DPO uses complete optimal trajectories computed with direct collocation. The start states for the optimal trajcetores are sampled from a normal distribution.

2.2.5. Approaches based on Reinforcement Learning

Searching for a sequence of actions that optimally (with respect to some defined optimality criterion) transfers the considered system from some initial state to a desired final state is the core domain of optimal control theory (OC) and reinforcement learning (RL). These two fields share many concepts and ideas. Nevertheless, there are differences in the assumptions and objectives of these scientific disciplines. For example, optimal control theory typically assumes a known, deterministic, and time-continuous model of the system. In contrast, the models of the reinforcement learning community are often stochastic, time-discrete and not (completely) known in advance. These differences induce different approaches and techniques to find an optimal action sequence. Sutton et al. [180] and Bennett and Parrado-Hernández [19] offer interesting perspectives on the connections and differences between RL and OC.

RL methods depend on training data, which is time-consuming and expensive to obtain [93]. Alternatively, training can be done in a simulated environment, which is much cheaper and safer than data collection on real systems; however, modeling errors cause a “reality gap” between simulation and reality that optimization-based learners usually exploit. The resulting policy must afterward be transferred to the real environment, termed *sim-to-real transfer* [216, 125]. Nevertheless, data-driven black-box approaches like RL typically generalize less well beyond the training scenarios than model-based white-box models.

RL is a huge field of research that has been explored for over 40 years [180]. This thesis can only give a rough overview of a selection of key approaches. Schaal and Atkeson [160], Kober et al. [93], Polydoros and Nalpantidis [147], and Singh et al. [164] have written excellent surveys on which the following review is based.

Value function approaches (also known as *critic-only* methods) are based on the HJB equation (see Equation (2.7)) or on its time-discrete equivalent, the Bellman Principle of Optimality [18] and aim at approximating the value function [160, 93]. This value function contains all information about the optimal policy, which can be obtained by optimizing the obtained cost (or reward in machine learning (ML)) in each step, thus pure value function approaches do not explicitly model the policy [164]. Temporal difference learning estimates a gradient of the value function from the difference between expected and received reward [93, 160]. Value function-based methods often use discretized action spaces, they are not guaranteed to converge to a near-optimal solution [164].

Policy search methods, also called *actor-only* methods, directly maintain and improve a model of the optimal feedback control. Commonly used policy search methods are gradient-based and optimize the parameters of a control policy, which are iteratively improved using the gradient of the cost function (or reward function in RL) with respect to the policy parameters [160]. The main difficulty of this approach is the estimation of this gradient. Approaches that are widely used in robotics use finite-difference estimations based on perturbed roll-outs and likelihood ratio methods [144].

Actor-critic methods approximate both the control policy and the value function and thus combine the advantages of the two approaches described above: the parameterized policy approximation provides a continuous action space and the value function estimation has a low variance compared to gradient estimations in actor-only methods [164]. The critic evaluates the performance of the policy model and guides the update of the actor, featuring "the local convergence properties of policy gradient algorithms while reducing the update variance" [93].

Levine and Koltun [102, 103] describe how to incorporate information from optimal trajectories computed with an iterative LQR solver in a policy search approach by using a technique called importance sampling. This enables the authors to "incorporate guiding samples into the policy search" to pretrain the policy, which is particularly useful when complex behaviors are to be learned.

Marin, Sigaud [115] use a learning classifier system XCFS as parametric policy representation. Training of the near-optimal feedback policy is done in two stages: Firstly, the learning classifier system is trained using near-optimal trajectories. Secondly, the XCSF parameters are improved using a direct policy search approach. The authors evaluate their approach on reaching motions of a robotic arm from different start states to a single target and from a single start state to multiple different targets.

Kim et al. [91] use an imitation learning approach to "combining multiple [optimal] trajectories to obtain a control policy". They do not perturb the start states to get different trajectories but consider parameterized dynamics and sample a set of parameters that "cover the space of system dynamics that are likely to be encountered during real execution". During iterative simulations for the different dynamics parameters, new optimal trajectories are computed as soon as the state leaves the area covered with training data (using the Maximum Mean Discrepancy metric). This is done until for all sampled dynamics parameterizations the state space along executed trajectories is covered with training data. Kim et al. also consider only partially observable dynamics, for which the parameterization is inferred using observations.

Khadke and Geyer [89, 90] consider decomposition of the dynamic system to reduce the curse of dimensionality when computing a feedback control. They decompose the system dynamics of the OCP in decoupled and/or cascaded subsystems with their respective independent sub-policies. For each lower-dimensional sub-problem, they use policy iteration to compute the global control sub-policy. Their approach also provides suboptimality estimates of the decompositions by computing the LQR or "DDP solutions for the original and decomposed systems from a few initial states to estimate" the value functions; their average difference quantifies the suboptimality. A difficulty of their approach is to find in all possible decompositions of the system dynamics one that provides low value errors.

Su et al. [178] use inverse reinforcement learning to incorporate information from numerical OCP solvers in RL. Samples from optimal trajectories computed using the GPOPS solver are used as expert knowledge. A generative adversarial network is trained to discriminate between the (optimal) expert trajectories and the trajectory data generated using the RL approach. The information from the adversarial discriminator network is used to optimize the feedback control policy network used in RL. The RL algorithm is an actor-critic approach with two neural networks (NNs) that approximate the near-optimal feedback control policy and the respective value function.

2.2.6. Model Predictive Control

Another major branch of research that is concerned with the near-optimal control of disturbed dynamical systems is MPC (or receding horizon control). It is an online approach that computes a solution during interaction with the real system. A new optimal trajectory from the current state is recomputed in regular short intervals, such that deviations are dealt with in an near-optimal manner. The challenge of these approaches arises from the requirement of being able to recompute optimal solutions very quickly. This is done by reducing the OCP to a short time horizon, the cost after this short time interval may be estimated by some approximation of the value function. This simplifies the problem that needs to be solved in each iteration. Furthermore, the OCPs considered for MPC typically have a special form (quadratic cost function and linear or linearized system dynamics) such that the special sparsity structure of the resulting optimization problems can be utilized to make computations efficient. Also, the optimization methods used are able to reuse the information provided by the previously found solution. To avoid linearization of nonlinear systems, nonlinear model predictive control (NMPC) builds upon MPC but considers nonlinear models in the receding horizon optimal control problem. NMPC is often but not necessarily formulated for infinite time optimal control problems.

Nonetheless, for a long time, the high computational effort only allowed applications for systems with slow dynamics, such that one of the first industrial applications was the control of chemical plants. Advancements in the optimization procedures [94] and NMPC [215, 74] enabled the use of NMPC in robotics, where fast movements and highly nonlinear dynamics set high demands on the real-time capability of the approach. Examples of NMPC applied to robotic systems are given in [205, 215, 183, 58, 210, 73, 169, 15]. While reducing the optimal control problem to a short horizon is necessary to allow real-time solutions, it introduces a suboptimality into the executed trajectory. The predicted (short horizon) open-loop trajectory starting at t_0 differs from the MPC's closed-loop behavior (since states are used for subsequent controls that are beyond the time horizon of the current step), which itself differs from the optimal solution of the full/infinite time horizon problem. These discrepancies are greater for shorter time horizons [59].

The use of a "terminal penalty function [...] such that a corresponding local control law is a good approximation of the control resulting from the infinite horizon control law [...] can recover the performance of the infinite horizon cost even for short horizons" [59]. NMPCs with a terminal penalty term combined with a terminal region constraint are called quasi-infinite horizon NMPCs. For these methods, a guaranteed stability proof exists, they are inherently robust to sector bounded input uncertainties [59]. Using an estimate of the OCP-related value function (or cost-to-go function) can have a significant influence on the performance of the MPC scheme. Zhong et al. [217] further note that the "more accurately the final cost [. . .], the less the method relies on the sum of the running costs." They estimate the value function of the OCP and use it to estimate the cost-to go in the MPC problem, which enables them to shorten the horizon of the MPC computation. Besides all this work, improvements on trajectory optimization software, fast approaches to compute derivatives and better physics simulations (see, e.g., Tassa et al. [183]) paves the way to more applications of NMPC to control robotic systems.

Song and Scaramuzza [168], [169] combine MPC with reinforcement learning: They use trained deep NNs to provide the hyperparameters for the MPC computations. A task-specific high-level reward function rates the quality of the parameterized MPC controller, policy search is applied to find a policy of optimal parameters for the low-level MPC controller that optimizes its performance with respect to this reward function. They apply their method to steer a drone through a moving gate. Song and Scaramuzza reference further work that connects learning approaches with MPC.

3. Synthesis of Extremal Field: An Iterative Approach

This chapter presents the general framework that is used in this thesis to compute approximations of the optimal feedback control policy for a given problem. It is based on the extremal field approach where data from a set of optimal trajectories from different start states is used to approximate a near-optimal global feedback control policy. The optimal trajectories are computed using the direct collocation method DIRCOL [173], which allows to solve very general OCPs with nonlinear cost functions and nonlinear state and control constraints. Furthermore, it is able to additionally compute estimates of the optimal trajectory's co-states. The policy is approximated using Gaussian processes (GPs) [154], which provide at each point a variance value that may serve as measure of uncertainty. In this chapter, a novel approach is proposed to iteratively select start states for new optimal trajectories that leverages information provided by the numerical optimal control solver and the current control policy approximation. The goal is to collect training data efficiently by using only a small set of optimal trajectories that provides sufficient information to cover a relevant part of the joint space.

The term *extremal field approach* that is used in this work dates back to Kelley [86] who proposed in 1962 to approximate a feedback control using a "field of extremals in the neighborhood of a predetermined extremal serving as a 'nominal' trajectory". The term extremal field refers to families of optimal trajectories [86, 30], as an *extremal* designates an *optimal trajectory*. The name *extremal field* or *field of extremals* has since been adopted by many researchers [30, 83, 64]. However, there have also been other terms for exactly the same approach. In addition to *extremal field*, Ghosh and Conway [64] also use the terms *synthesis of optimal feedback controllers* and *feedback synthesis method*, Pesch et al. [143] use *synthesis of optimal strategies*, Atkeson and colleagues [172, 107, 8] use *trajectory libraries* , Breitner [29] uses *guidance synthesis*. The fact that other terms may be used to refer to the *extremal field* approach must be taken into account when the existing literature is searched.

3.1. Related Work on the Iterative Extremal Field Approach

The discussion of existing work on (near-) optimal feedback control policies in Section 2.2 excludes methods based on the extremal field approach. These are presented in detail in this section.

Machine learning approaches to finding a near-optimal policy usually approximate either the value function as a solution of the Hamilton-Jacobi-Bellman equation to derive the optimal control or directly the optimal feedback control using policy iteration. Both approaches lack the generalization to large-scale nonlinear system dynamics and nonlinear constraints. They usually do not account for the capable model-based numerical trajectory optimization methods. However, some of these provide valuable information that can be highly beneficial to be utilized for the learning process.

Compared to reinforcement learning, the advantage of the extremal field approach used in this thesis is the direct incorporation of model knowledge into the construction of the control policy during training. This is done indirectly using data from optimal trajectories that comply with the known physical model of the system including its dynamics and constraints.

Some of the first approaches that directly use trajectory information to fit an approximation of the feedback control are proposed by Edwards and Goh [51] and Pesch et al. [143]. Edwards and Goh formulate their approach as a parameter selection problem, where a series of optimal control problems is parameterized by the weights of a NN that represents the feedback controller. The optimal trajectories are computed using MISER [70], in their evaluation, they consider a flight control problem. Pesch et al. solve a pursuit-evasion problem, a two-person differential game where a rat escapes from a cat. They also train a NN to approximate the optimal strategy for both opponents. The open-loop trajectories that provide the training data are computed analytically. Some years later, Hardt [76] used a similar approach in his dissertation, controlling a jet engine compressor. He combines a direct optimal control method with an indirect method to get more precise open-loop trajectories.

In all three approaches, the start states of the open-loop trajectories are on regular grids in the state space, which scales poorly for high-dimensional problems and is thus not applicable for robots with many joints. For the three dimensional flight guidance problem in [51], each dimension is subdivided in five grid points, which already results in a total of 125 optimal trajectories to be computed.

In his master thesis, Wiratunga [203] learns a simplified inverse dynamics model for quadrotor helicopters using Gaussian Process Regression. Optimization of parameterized trajectories are used to generate training data for the GP. In this optimization problem, the variance of the GP is maximized to cover areas of insufficient training data with new trajectories. The thesis compares different sparse approximations of GP regression (including Sparse Spectrum GP Regression), and evaluates different methods of hyperparameter optimization. The main difference to the approach considered here is that Wiratunga learns the inverse dynamics instead of the optimal feedback control model. The model of the inverse dynamics is the mapping from a desired acceleration (and the current system state) to the torque required to get this acceleration and does not consider a task-specific objective function. Consequently, the objective function of the optimization problem consists of a term to induce and guide the exploration of the state space and contains no task specific optimization of the movement.

Beaudin and Lin [15] apply MPC on a quadruped robot to provide real-time control and combine this with a trajectory optimization approach for long-term path planning. In an offline step, optimal trajectories are generated using direct collocation for a highly simplified robot model: The problem formulation considers only the four-dimensional state that describes the torso position with state and control bounds and collision avoidance. A NN that is used to predict the velocities required to proceed to the goal state is trained on the data from these trajectories. This network is then used online to compute the reference trajectory for the MPC approach that uses a more detailed physical model to solve the whole-body control problem and provide the motor torques to follow this reference trajectory. Since the previously learned policy that provides the desired velocities ensures that the long-term goal is approached, the MPC approach uses only a short time horizon, which reduces the complexity of the MPC.

Tsiotras and Sanz Díaz [190] compute a near-optimal feedback control for a specific collision avoidance maneuver for cars. Their car model includes wheel dynamics and a tire friction model. They use a GP to interpolate data from numerically computed optimal trajectories. In their problem, they vary only a single value (initial speed) to compute different trajectories and are therefore able to use a grid of initial states.

Merkt et al. [117] learn a feedback controller from optimal trajectories to warm-start optimal control solvers. The use of good initial guesses to warm-start the solver reduce the number of iterations required to find an optimal solution and increases the success rate of the solver. The authors sample trajectories from a fixed grid of start states and use NNs to interpolate between the optimal trajectories. In their work, they explicitly consider multimodality (multiple optimal solutions to the same problem) and discontinuity (similar problems with very different solutions).

Indirect methods for trajectory optimization provide very accurate results which makes them attractive for space mission planning. However, they require a good initial guess of the adjoint variables. Singh and Junkins [165] propose a variation of the extremal field approach and use information from a set of neighboring optimal trajectories to train GPs to provide an approximation of the adjoint variables. This enables them to recompute optimal trajectories during a space mission very quickly, as the indirect method converges after only a few iterations when initialized with the learned adjoint variable approximation. They perturb the terminal conditions of the adjoint variables and integrate backwards to create optimal trajectory bundles for training. The authors employ an extremal field approach, but with the special feature that the goal is the approximation of the adjoint variables and not, as usual, that of the control.

Zhong et al. [217] train a NN using trajectory data to approximate the optimal value function. In their work, they consider very general but discrete-time problems. They conclude that fitting the value function using trajectory data is difficult, since it is sensitive to many parameters and “a good fit is commonly elusive”. They further note that the “main difficulties in obtaining data arose in defining useful initial states.” This problem is considered in Chapter 3 and 4 in this thesis.

Mordatch and Todorov [120] continue the work of Zhong et al. They also use a discrete-time formulation of the OCP, but rely on a direct method (instead of iLQG) to compute the optimal trajectories; they use NNs for function approximation. In their work, they combine the solution of a collection of OCPs for different start states and the fitting of a feedback control policy to the solution of these OCPs into a single optimization problem. The resulting optimization problem is solved using the *alternating direction method of multipliers*. They thus jointly optimize a set of trajectories from different start states and the parameters of the corresponding feedback control policy approximation. The start states are selected once at the beginning using random sampling and kept fixed.

Aircraft and aerospace vehicles are often exposed to strong perturbations, such as winds. Jardin and Bryson [83] consider minimum-time paths of aircraft on a spherical earth model in the presence of strong winds. They propose to create families of optimal paths to some destination using backward integration of the differential equations of state and control (the latter is derived using necessary conditions for the solution of the optimal control problem). The optimal solution for intermediate start states is then found by interpolation between these trajectories using Delaunay triangulation.

Ghosh and Conway [64, 63, 65] consider minimum-time orbit insertion and (aeroassisted) orbit transfer problems as well as an orbital pursuit-evasion game. They use universal kriging (which is basically GP regression [35]) to approximate the feedback control policy from a set of optimal trajectories from different start states. To get useful initial guesses for the numerical trajectory optimization, they apply a preprocessing step, where unconstrained guesses are generated with a trajectory optimization method based on the particle swarm metaheuristic [148, 37] that does not require an initial guess. The start states are sampled around the nominal start state using Latin hypercube sampling (LHS), which scales well to higher dimensions. In contrast to the approach presented here, their method of start state selection is not iterative. Extending the extremal field approach, Chilan and Conway [34] consider significant disturbances using the example of a high speed aerospace vehicle with wind disturbances. They use real-time measurements of the disturbances as additional inputs to the learned feedback control model, which notably improves the performance compared to a feedback control policy that is ignorant of the measured perturbations.

Schierman et al. [161] investigate how a vehicle flight control for re-entry can adapt to critical failures of control effectors. They create a database of optimal trajectories by varying the trajectory start states as well as critical parameters of the optimal control problem. The latter enables them to simulate critical failures of control effectors that alter lift and drag characteristics. The trajectories in the database are represented as polynomials; polynomial neural networks (PNNs) are then used to map initial states to the coefficients of these polynomials. The PNNs are used during flight to provide the coefficients that constitute the trajectory for the current state and parameterization. The critical parameters are inferred from sensor data. To make their solution robust to sensor noise and disturbances, SChierman et al. correct the coefficients provided by the PNN models online by applying a least-squares approach that minimizes the error and distance to the nominal trajectory. The trajectory defined by the corrected coefficients is then used by the adaptive guidance and the inner-loop feedback controller.

Kim et al. [91] focus on uncertain dynamics $f_\alpha(x_i, u_i)$, where the uncertainty parameter α may also be only partially observable. They train a regression function that learns a mapping of the product of state and parameter space to the control space. Their approach is applicable to problems with only partially observable dynamics: Get a posterior distribution of the hidden properties based on online observations and use this instead of the parameter. For training, they use data from optimal trajectories. Several local regressors are learned and a distance-based criterion (maximum mean distance) is used to select the appropriate local regressor for each query point. An iterative process is used to add additional optimal trajectories if required: The parameter space is sampled and for each sample, the trajectory from the given start state is simulated in each iteration. If for some query point no local regressor can be found, a new trajectory is optimized starting from this query point. This is repeated until the trajectories for all sampled parameters can be simulated without computing new trajectories.

Atkeson and his colleagues [7, 9, 121, 172, 10, 106, 8, 107] have been working on the approximation of global feedback control policies using locally optimal trajectories over a period of almost 20 years. The methodology employed by the authors is focused on DDP (cf. Section 2.2.4). However, their work is still strongly related to the iterative extremal field approach presented in this chapter. They consider deterministic problems where the state and control are continuous but time-discrete. Their main tool to compute locally optimal trajectories is DDP. This method gives them a quadratic model of the value function and a linear model of the policy along this trajectory. In some works [172, 106], they compute trajectories by first generating trajectories using another approach like sequential quadratic programming methods [22] or the path search method A^* and refine the result with DDP. They ensure consistency between the computed trajectories by recomputing them frequently using information from neighboring trajectories. New start states, from which optimal trajectories are computed, are selected randomly, but with several acceptance criteria based on the estimated trajectory cost to reject unnecessary or incorrect data [10]. Furthermore, the approach presented by Atkeson and colleagues [10, 8] is one of very few that select start states iteratively to consider information collected so far. For global approximation of the value function and control policy, they do not use a trained function approximation, but a nearest-neighbor approach. Efficient look-ups are done using kd-trees. While parameterized function approximations like GPs or locally weighted projection regression (LWPR) may be more accurate, this approach makes it easier to correct model errors from data collected online. One needs to just improve the already computed optimal trajectories using the corrected system model, there is no subsequent retraining of the global parameterized value function and policy model necessary. The problems considered are one- to four-link pendulum swing-ups and balance and periodic walking motions of a humanoid five-link robot.

Approaches for selecting (optimal) trajectories to guide data collection in order to learn a control policy have been proposed in slightly different contexts, such as parameterized skill learning. While this thesis focuses on learning a feedback control policy from multiple trajectories, parameterized skill learning aims at learning the policy for multiple similar (parameterized) tasks using multiple feedback policies. These approaches need to iteratively select new tasks for which the expensive training of a policy improves the performance in a wide range of related problems. Baranes and Oudeyer [11] focus on the constrained selection of new training tasks for learning parameterized policies. Their

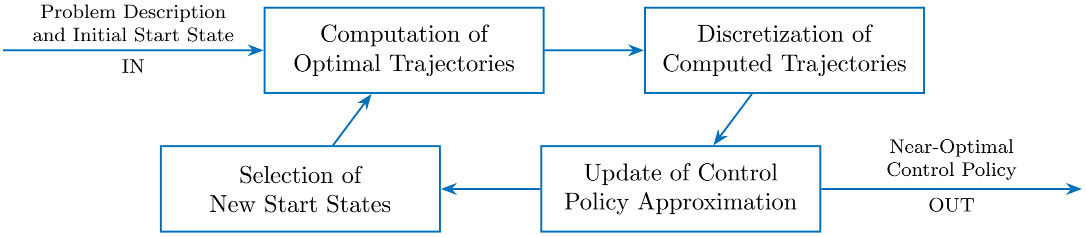


Figure 3.1.: Flowchart of the four main steps of the iterative extremal field approach.

approach uses a measure of interest based on competence progress to explore the task space. However, Baranes and Oudeyer consider inverse dynamics models and not optimal feedback controls. Da Silva et al. [40, 39] apply a Bayesian approach and maintain a model of the skill performance using a GP. They select new tasks that maximize the expected improvement in overall skill performance. Since they solve families of control tasks instead of a policy for a single task, they can reuse unsuccessful policies that appeared during training as training samples for a different goal. Queißer et al. [151, 152] use the "current estimate of the iteratively trained skill" to initialize the optimization of new tasks.

It should be noted that this chapter addresses the identification of trajectory start states such that optimal trajectories starting from these states provide useful data to improve the learned control policy. This is a different type of problem than finding good initial guesses for warm-starting trajectory optimization as done, e.g., in [101, 113] or [117].

3.2. Successive Optimal Trajectory Generation and Approximation of a Near-Optimal Policy

To learn a near-optimal feedback policy for a given optimal control problem (see Section 2.1), information collected from a set of optimal trajectories for different start states in the state space is used. These trajectories are generated with the solver DIRCOL [173] that implements a direct collocation method to solve optimal control problems.

The start states are not determined at the beginning, instead, an iterative approach is used that allows to continually supervise the improvement of the policy, select new start states based on progress achieved until then, and add only relevant data. The optimal control policy is approximated by a GP, retrained in each iteration with the data points on the optimal trajectories. The main steps of the algorithm are depicted in Figure 3.1 and

Algorithm 1: Pseudocode of the Iterative Extremal Field Approach

```
Input: problem description problem, start states startpts
Output: policy pi

for fixed number of iterations do
    optTrajs ← getOptimalTrajs(problem, startpts, pi)
    pointList ← discretizeAndSelect(optTrajs)
    pi ← trainGPs(pi, pointList)
    startpts ← getNewStartpoints(optTrajs, pi)
end
```

additionally outlined as pseudo-code in Algorithm 1. They are motivated and described in more detail in the following subsections.

3.2.1. Computation of Optimal Trajectories

DIRCOL is a direct collocation method written by von Stryk [174, 173] that uses SNOPT [66, 67] to solve optimal control problems. It allows to solve OCPs with highly nonlinear system dynamics, nonlinear path constraints on the state and control variables and constrained terminal states.

It requires a rough initial guess of the state and control trajectories to start the numerical optimization; however, this guess does not need to be feasible with respect to the ODE and other constraints. Direct collocation methods are, in general, more robust to poor initial guesses than indirect methods [21, 149]. Nevertheless, a reasonably good user-provided initial guess may make the difference between the success and failure of the iterative optimization method.

Three different approaches are potentially used to get initial guesses:

(i) Linear interpolation between the start and goal state (or the expected final state if the joint state at the end of the trajectory is not predetermined), with the control constantly zero: This is the simplest and most unbiased initial guess. In the first iteration, this is the only initial guess used since it requires no additional information.

(ii) An already computed trajectory that starts or passes as close as possible to the current start state: The optimal trajectories from close but different start states often (but not always) have a similar shape. Using an already computed solution that is close to the current start state (in joint space) as initial guess leads in most cases to convergence of the optimal control solver and helps getting consistent solutions. A similar approach is used by Atkeson and Stephens [10] to get a globally consistent value function estimate. However, this approach carries the risk of being biased by the neighboring solution and finding only a local minimum instead of the global optimum. If the distance to the closest data point is too large, intermediate start states are added from which the corresponding optimal trajectories are computed one after another, before the OCP for the original start state is determined. Like that, there is an initial guess from an optimal trajectory in close proximity for each optimal control problem.

(iii) Simulation of a trajectory using the current approximation of the optimal feedback control: To be applicable, this approach requires a sufficiently accurate approximation in the relevant region, it is thus very error-prone in the first iterations. A similar approach has been used by Queißer et al. [151].

Depending on the problem, the approaches above are more or less useful and are enabled or disabled as required, apart from approach (i), which is used in any case. For each start position, the solver is run with all enabled initial guesses to avoid using suboptimal solutions and the best valid solution (the one with lowest total cost) is kept. DIRCOL is started with 11 equidistant grid points; at most ten grid refinements are allowed. Further, DIRCOL's automatic scaling of variables and functions is enabled. If the solution process fails, a computationally expensive homotopy approach is applied where a series of optimal control problems are solved: The final time is increased stepwise from a small fraction to the original value, using each time the solution of the previous run as an initial guess.

In the presented approach, the numerical optimal control solver DIRCOL can be replaced by any other trajectory optimization method that is able to provide an estimate of the adjoint variables, which is required for the selection of new start states (see Section 3.2.3). Examples of existing software are given in Appendix A.2. In this thesis, DIRCOL has been found to be a very fast and reliable solver and is therefore used as the primary tool to solve optimal control problems.

3.2.2. Discretization of Trajectories for Training Data Extraction

The GP learns the control values evaluated at discrete data points, which makes it necessary to extract state-control value pairs $(x(t_i), u(t_i))$ from each computed trajectory and add them to the data used to train and evaluate the policy. Each trajectory provides a set of at most N data points

$$\mathcal{S} = \{(x(t_i), u(t_i))\}_{i=0,\dots,N} \text{ with } t_i = \frac{i \cdot t_f}{N-1} \tag{3.1}$$

that are used to improve the learner. Various experiments with different numbers have shown that the formula $N = \min\{130, t_f/t_{\text{min}}\}$ provides good results. It extracts 130 data points from each trajectory and reduces this number for short trajectories, as a minimum distance in time t_{min} between the discretization points is enforced.

GPs can not cope with redundant input data, which has been analyzed in [42]. A straightforward approach is to remove all points that are closer than some minimum distance d_{min} to any previously added state in $\mathcal{S}$. The remaining data in $\mathcal{S}$ is used as training data for the learner. In Section 4.1.2, an improved procedure for selecting the data points used is presented.

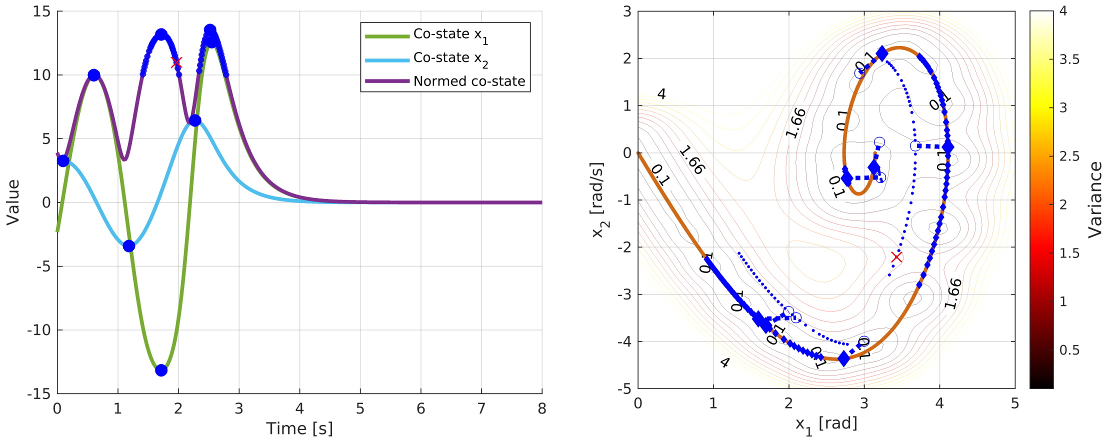
3.2.3. Selection of Start States for New Optimal Trajectories

The selection of new start states is highly relevant for the “exploration” of unknown areas of the state space and determines how the algorithm progresses. It is important to acquire more information in regions where a deviation from the optimal trajectory is relevant, i.e., where suboptimal actions cause a significant change in the total cost of the executed path.

For some optimal trajectory $x(t)$, the relation between the co-states $\lambda(t)$ (which are also called adjoint variables) and the value function is described by

$$\lambda(t) = \frac{\partial V(x(t), t)}{\partial x(t)}$$

(see [30]). Large (absolute) co-state values indicate where some deviation from the trajectory in state has a considerable impact on the value function; Bryson and Ho [30] thus describe them as “influence functions on [the cost] J of variations in $x(t)$”. This motivates the use of the trajectory’s co-states $\lambda(t)$ as an indicator for sensitive regions where having more neighboring trajectories would be beneficial. DIRCOL allows to estimate the Lagrange multiplier function of a computed approximation of an optimal trajectory; these multiplier functions are represented as linear splines. In the following, a start state selection strategy is described that uses co-state information to select new initial states for optimal trajectories. This strategy is referred to as *adjoint-based*.



(a) Trajectory over time

(b) State space view

Figure 3.2.: This example based on the weakly actuated pendulum problem (cf. Sec. 3.3.1) illustrates how start states are selected based on co-state values and the variance of the GP. The figure shows extrema of the co-state values (large filled marker), large normed co-state values (smaller marker on trajectory) and the corresponding candidate start states (unfilled markers and dots). The selected new start state at the state with the largest variance of the current policy approximation (illustrated as a contour) is marked with an “X”.

For a new optimal trajectory, new candidate start states are added to a list of existing candidate start states. These new candidates are selected as follows: Firstly, a list of timestamps $\{t_i\}_i$ is compiled from the list of times at which a local extremum occurs in any of the n_x co-states functions $\lambda_j(t)$ with $j \in \{1, \ldots, n_x\}$, and the list of times at which normed co-state value is large. A second list consists of nodes of the optimal trajectory's splines at which the normed value over all co-state is outside the 67 %-quantile, this list is thinned out so that there is a minimum time interval between its elements. See Figure 3.2a for a two-dimensional example. Secondly, the new candidate start states $\tilde{x}(t_i)$ are set to be in some distance to the points $x(t_i)$ on a trajectory in the direction of the co-state vector $\lambda(t_i)$. The Euclidean distance between $\tilde{x}(t_i)$ and $x(t_i)$ depends on the normed co-state value and is in the interval $[b_l, b_u]$. The mapping from trajectory points $x(t_i)$ to new candidate start states $\tilde{x}(t_i)$ is

$$\tilde{x}_i\left(x(t_i), \lambda(t_i)\right) = x(t_i) + \kappa\left(\|\lambda(t_i)\|\right)\lambda(t_i) \tag{3.2}$$

$$\text{with } \kappa : \lambda \mapsto \frac{1}{\lambda}\left(b_l + (b_u - b_l)e^{-\nu\lambda}\right).$$

Since large absolute co-state values indicate more sensitive regions, the distances between respective points are smaller. The exponential decrease favors start states close to the trajectory. The real-valued parameter $\nu > 0$ as well as the interval $[b_l, b_u]$ need to be tuned problem-specifically. For each new optimal trajectory, the list of candidate start states is supplemented by new start states computed using (3.2). If a start value for a new optimal trajectory is needed, the variance of learned policy is evaluated for all these candidates and the one with the highest variance of the current policy approximation is selected. This is illustrated in Figure 3.2b.

An initial non-empty set of start values has to be provided with the problem formulation. In this work, the initial set consists of a single start state; the optimal trajectory from this start state is sometimes referred to as the reference trajectory. It should be noted that the optimal control problems resulting from the initial start state(s) should not be too difficult to compute because, contrary to the following iterations where already computed trajectories and the approximation of the feedback control can be used, no problem-specific information to generate initial guesses is available.

If the function f that determines the ODE does not explicitly depend on the j-th state variable $x_j(t)$, then $\lambda_j(t)$ is constant. Consequently, the presented start state selection method will not provide any candidate start states for $\lambda_j(t)$, as it has no extrema. Nevertheless, since all entries of the state vector are employed to select new start states and the right-hand side of the differential equation f normally depends on at least one entry of the state vector, at least one co-state will contribute new candidates.

3.2.4. Learning Near-Optimal Feedback Control Policies from Optimal Trajectories

In this work, GPs are used to model the near-optimal feedback control policy approximation. GPs have been successfully used in [96, 44, 64] to approximate the control policy or the value function of optimal control problems. GPs are so-called non-parametric function approximators [154], which means that there is no need for a predetermined assumption on the class of functions appropriate to fit the given data. Further, GPs need not much data (compared to, for example, neural network functions) to achieve satisfactory approximations. The main advantage of using a GP model to approximate the near-optimal policy is that it provides for each joint state a variance, which serves as measure of uncertainty.

Although GPs are known to be non-parametric, this does not mean that the model is free from any assumptions on the function to be learned, but the class of functions that can be modeled with a single GP model is larger than that of parametric models. The assumptions on the learned function are mostly determined by the choice of the covariance function, since "it encodes our assumptions about the function which we wish to learn" [154]. There exists a large variety of well known covariance functions; see, for example, Chapter 4 in the book by Rasmussen and Williams [154]. The high order of differentiability that is inherent to many GP kernels, like the squared exponential or radial basis function covariance function, makes them unsuitable for modeling the feedback control policy, which may exhibit many different local properties. A neural network kernel function (also referred to as multi-layer perceptron (MLP) kernel) is more appropriate, as it is a nonstationary covariance function, which allows to fit functions with both steep slopes and smooth plateaus. In the context of large-scale terrain modeling, this kernel has also been identified by Vasudevan et al. [194] as very suitable to model "fast-changing/large discontinuities in data" that appear in the sensor data representing scans of large-scale terrain.

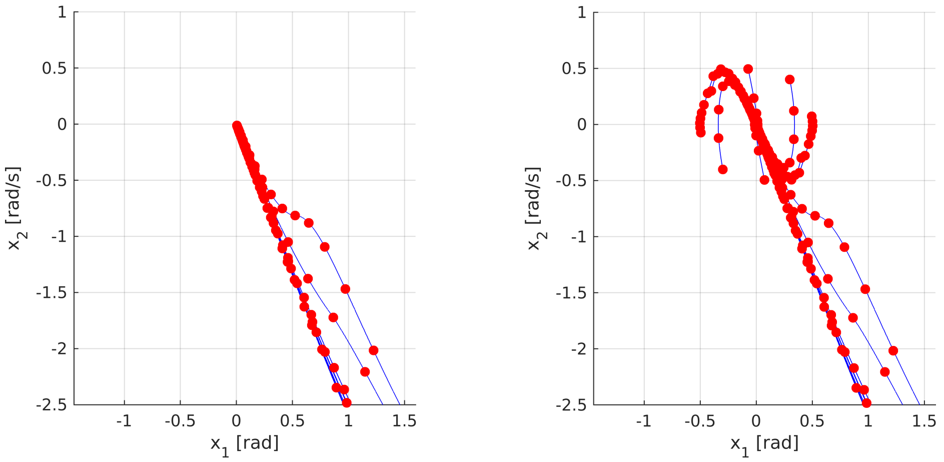
The GP needs to be retrained each time data points of a new optimal trajectory are added. Training requires the inversion of the kernel matrix, usually realized with a Cholesky decomposition, whose complexity of $\mathcal{O}(n^3)$ (for n training samples) dominates the complexity of the overall training algorithm [154]. In consequence, a large number of data points slows down the training and also the evaluation of the GP. To attenuate this, it is possible to use some sparse pseudo-input model as, e.g., introduced in [167] to reduce the computational cost for both operations. However, the use of a sparse approximation may degrade the quality of the learned near-optimal policy.

It must be noted that a policy implemented by a GP is likely to violate the box constraints of the control. It is possible to construct a learned policy that intrinsically complies with box constraints by using some bijective mapping $\mathbb{R} \to (-1, 1)$ that is wrapped around the Gaussian approximation. The following mapping based on the arcus tangens function is given exemplarily to demonstrate the approach:

$$m(p) = \frac{u_{\text{max}} - u_{\text{min}}}{\pi} \operatorname{atan}(p) + \frac{u_{\text{max}} + u_{\text{min}}}{2} \tag{3.3}$$

The training data for the GP is then transformed to be $\left\{x_i, m^{-1}\left(u_i\right)\right\}_i$ and the control approximation is the mapped output of the GP model $u = m(\text{GP}(x))$.

An important disadvantage of this approach is that this transformation works on open intervals, but the control trajectories computed using DIRCOL also include values on the boundary of the feasible region, which would be mapped to infinity. Another disadvantage of this approach is that it introduces considerable nonlinearities into the policy model,



(a) All generated optimal trajectories approach the goal state from the same direction in a small tube.

(b) Additional short trajectories sampled in the proximity of the goal state amend this deficit.

Figure 3.3.: Example showing the sparsely sampled state space around the goal state and the addition of additional trajectories in this area, based on the weakly actuated pendulum problem (cf. Section 3.3.1).

which complicates learning. Finally, this approach does not generalize for nonlinear constraints. These are the main reasons why this approach is not used in this work. Instead, the values outside $\mathcal{U}$ are reset to some valid controls on the boundary of $\mathcal{U}$. Consequently, it is necessary to reset values outside $\mathcal{U}$ to some valid controls on the boundary of $\mathcal{U}$. In case of box constraints, this simplifies to the mapping $u = m(\text{GP}(x))$ of the learned control policy with m defined as

$$m_i(u_i) := \begin{cases} u_i, & \text{if } u_{\text{min},i} \leq u \leq u_{\text{max},i} \\ u_{\text{min},i}, & \text{if } u_i < u_{\text{min},i} \\ u_{\text{max},i}, & \text{if } u_i > u_{\text{max},i} \end{cases} . \tag{3.4}$$

This saturation of the learned control is easy to implement and efficient to compute for box constraints. Its generalization to nonlinear constraints is the projection onto the boundary of the feasible region.

3.2.5. Additional Samples around the Goal State

As can be seen in Figure 3.3a, the sampling of the state space close to the goal state may stay sparse. All generated trajectories approach this state in a narrow tube, leaving a significant area around the goal state unexplored. In the example shown in Figure 3.3, the states are not constrained to the goal state at the end of the trajectory (the convergence to the goal state is accomplished only by the formulation of the objective function). This forces the co-states to become zero at the end point of the trajectory (for details see [30]), leading to small co-state values and hence no new start states around the goal state.

Altogether, while the co-states give valuable information about sensible regions around a large portion of the computed optimal trajectories, they are of little use to promote exploration in the direct proximity of the goal state. The start state selection strategy based on co-state values is thus complemented by adding short trajectories starting at additional (e.g., randomly placed) positions on a ball with small (problem dependent) radius around the goal state. These trajectories are much shorter and accordingly add less data points to the policy learner than the ones starting from points imposed by the method described in Section 3.2.3. In the next chapter in Section 4.1.3, an alternative using a switch-over to a proportional-integral (PI) controller is proposed.

3.3. Evaluation

The viability of the presented iterative approach is evaluated in simulation on models of a weakly actuated pendulum and an industrial robot arm. These two dynamic systems are described in detail in the next subsections. The implementation is done in MATLAB; the GP implementation used is the toolbox *GPmat* by Lawrence and others [3]. The numerical optimal control solver DIRCOL is written in Fortran and uses text files as input and output. A self-written C++ wrapper allows the formulation of optimal control problems, execution of the solver DIRCOL and retrieval of the computed solution in C++ code. MEX files provide access to this wrapper from within MATLAB.

3.3.1. Feedback Control of the Weakly Actuated Pendulum

The weakly actuated pendulum is a single-joint pendulum with system dynamics

$$\ddot{\theta}(t) = \frac{g}{l}\sin\left(\theta(t)\right) + \frac{u(t)}{ml^2} - \mu\frac{\dot{\theta}(t)}{ml^2}, \tag{3.5}$$

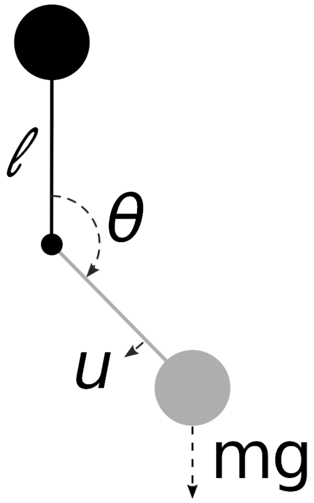


Figure 3.4.: Illustration of the weakly actuated pendulum.

as given in [159, 44, 49]. The link length is $l = 1\,\mathrm{m}$, mass $m = 1\,\mathrm{kg}$, friction coefficient $\mu = 0.05\,\mathrm{kg\,m^2/s}$ and $g = 9.81\,\mathrm{m/s^2}$. The state is defined as $x = \begin{pmatrix} \theta & \dot{\theta} \end{pmatrix}^T$ in $\begin{pmatrix} \mathrm{rad} & \mathrm{rad/s} \end{pmatrix}^T$.

The system is called *weakly actuated* since the applicable torque is constrained to be $u \in [-5, 5]\,\mathrm{N\,m}$. This restriction makes the task non-trivial, as the maximum available torque is not sufficient to bring the pendulum from a hanging position ($\theta = \pi$) to the goal state. Accordingly, a trajectory that solves this task must contain some swing-up [49].

The quadratic cost function of the OCP is

$$\mathcal{J}(x, t_f) = \frac{1}{2}\left(10\bar{x}_1(t_f)^2 + 10\bar{x}_2(t_f)^2\right) + \frac{1}{2}\int_0^{t_f} \bar{x}_1(t)^2 + \bar{x}_2(t)^2 + u(t)^2 + \bar{x}_1(t)u(t)\,dt \tag{3.6}$$

with $\bar{x}(t) := x_f - x(t)$, similar to [43].

The start state is fixed by constraints to be $\theta(0) = \pi$, $\dot{\theta}(0) = 0$, which describes the pendulum at its stable equilibrium state with hanging mass. The final state $x_f = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ is not enforced by constraints but is used in the objective function (3.6) to penalize deviations from this state at the terminal time. The OCP's terminal time is fixed to be $8\,\mathrm{s}$. This is a large value considering that the time-optimal solution starting from $\begin{pmatrix} \pi & 0 \end{pmatrix}^T$ can reach the goal state within $3\,\mathrm{s}$. Still, the cost function favors solutions that quickly approach the goal state and stay there for the remaining time. The reason for the large terminal time is to make it more likely that the optimal control solver can find a solution for more difficult start states. The approach presented in this thesis is applicable to both problems with fixed and free terminal time.

The state of the original problem is unconstrained. In this work, very loose constraints of $x_{\text{max}} = \begin{pmatrix} 4 \cdot 2\pi & 100 \end{pmatrix}^T$, $x_{\text{min}} = -x_{\text{max}}$ are used to improve efficiency of the numerical collocation method to solve the OCP.

To find a near-optimal feedback control policy, Algorithm 1 is applied to the described OCP. After ten iterations, the control approximation is trained with 219 samples from ten trajectories. Eight additional trajectories around the goal state (see Chapter 3.2.5) increase the number of training samples to 308. To give an unbiased view on the start state selection procedure, all computed trajectories are manually checked during execution. A re-computation is initiated if some OCP has not been successfully solved to ensure that the trajectories for all start states are used. All trajectories and samples used for training are depicted in Figure 3.5a as blue lines and markers.

A test set of 200 trajectories, which solve the above OCP, starting from different start states s_i with $0.3 \leq \| \begin{pmatrix} \pi & 0 \end{pmatrix}^T - s_i \|_2 \leq 1.0$ is used to evaluate the learned feedback controller. The trajectories of the system that result from applying the learned feedback control policy are computed starting from each of the 200 start states of the test set. The numerical integration method used in this chapter for evaluation is a self-written classical fourth-order Runge-Kutta scheme with a fixed step size of 1×10^{-3} s. The total cost of the trajectory is integrated using the trapezoidal integration rule, as the requirement for accuracy is not as high as for the system state. In the evaluation of the results for the weakly actuated pendulum, the goal is considered as reached if the Euclidean distance of the state $x(t)$ to the goal state x_f is less than 10^{-3} and a near-optimal trajectory is said to be found if its approximated cost is at most $110\,\%$ of the trajectory cost computed by DIRCOL for the same start state.

A selection of covariance (or kernel) functions is used for the evaluation to determine a GP kernel capable of modeling the approximate feedback control. Radial basis function (RBF) kernels, Matérn-$3/2$ and Matérn-$5/2$ kernels are standard kernel functions that are often used. They are, however, both stationary models, which means that they are "invariant to translations in the input space" [154], but this does not hold for the feedback control. The multi-layer perceptron (MLP) kernel, also called neural network kernel [154] or arcsin kernel, is nonstationary. It results from a neural network with a single hidden layer where the number of hidden nodes tends to infinity [202]. The definition of several

GP kernel	✳ near optimal	✳ goal reached	✳ failed	distance to goal		cost ratio	
				mean	var	mean	var
MLP	168	32	0	$6.46 \cdot 10^{-4}$	$1.61 \cdot 10^{-11}$	1.0375	0.0050
MLP (FITC)	7	139	54	$1.40 \cdot 10^{-3}$	$1.90 \cdot 10^{-5}$	1.5721	0.0467
MLP (DTCVAR)	173	22	5	$3.14 \cdot 10^{-4}$	$3.78 \cdot 10^{-7}$	1.0553	0.0094
Matérn-3/2	159	39	2	$1.53 \cdot 10^{-4}$	$3.78 \cdot 10^{-8}$	1.0834	0.0198

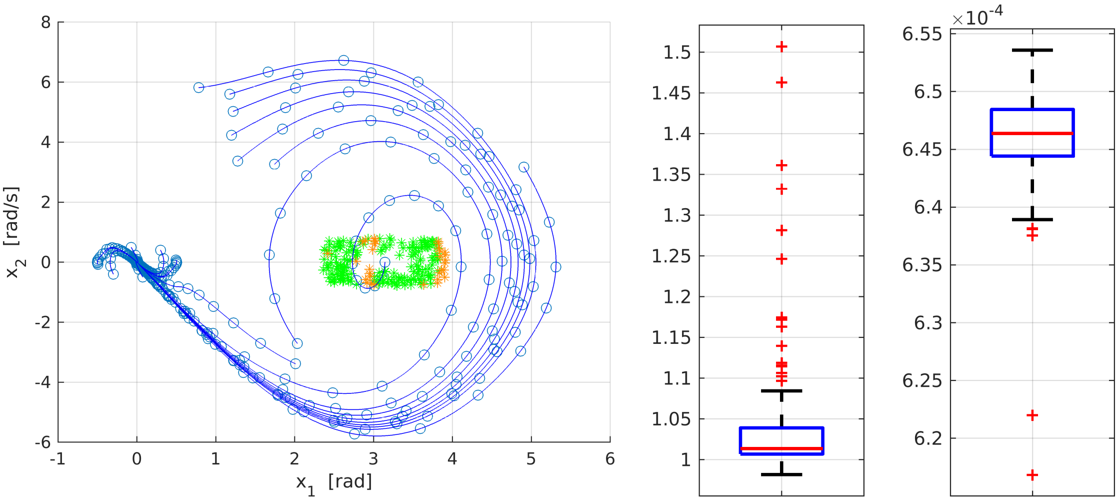
Table 3.1.: Evaluation results for the weakly actuated pendulum problem using different GP kernels.

kernal functions is given in Appendix A.1; a detailed description of all covariance functions can be found in Rasmussen and Williams's book [154].

GPmat provides several sparse approximations of the GPs based on sparse pseudo-input models that aim to reduce the computational burden of models with large training data sets. In this work, the abbreviations used in *GPmat* to denote the different sparse approximations have been adopted. They originate from the terminology introduced in [153]. The following sparse approximations are used: the fully independent training conditional (FITC) by Snelson and Ghahramani [167], the partially independent training conditional (PITC), proposed in [153], the deterministic training conditional (DTC) described by Csato and Opper [38], and the improved version of DTC proposed by Titsias [186] (DTCVAR).

The experiments show that the MLP covariance function with a non-sparse posterior variance approximation provides the best results for the pendulum problem. The result achieved with this kernel function is shown in Figure 3.5. Green stars represent successful test cases; orange stars indicate trajectories reaching the goal state with costs more than $110\,\%$ of the optimum, and red stars show test cases where the goal state has been missed. In Figure 3.5b, the ratio of 19 test cases is slightly lower than 1.0 (the minimum is 0.9816), indicating a lower cost than the optimal trajectory computed with DIRCOL. The reason is a less accurate cost estimation for the simulated trajectories (linear approximation, whereas DIRCOL uses cubic spline interpolation of the states). The near-optimal feedback controller reaches the goal state in all cases, as the Euclidean distance at the final state of the simulated trajectory to the goal state falls below 10^{-3}, as can be seen in Figure 3.5c.

The results for other kernels or approximated GPs are reported in Table 3.1. The Matérn-3/2 kernel and the MLP kernel with DTCVAR provide very good results. Using the MLP kernel with the FITC approximation leads to a significant decrease in the



(a) The trajectories and selected data points that are used to learn the near-optimal policy and the results for the test set with 200 start states.

(b) Ratio "simulated cost to optimal cost".

(c) Euclidean distance from goal state at t_f.

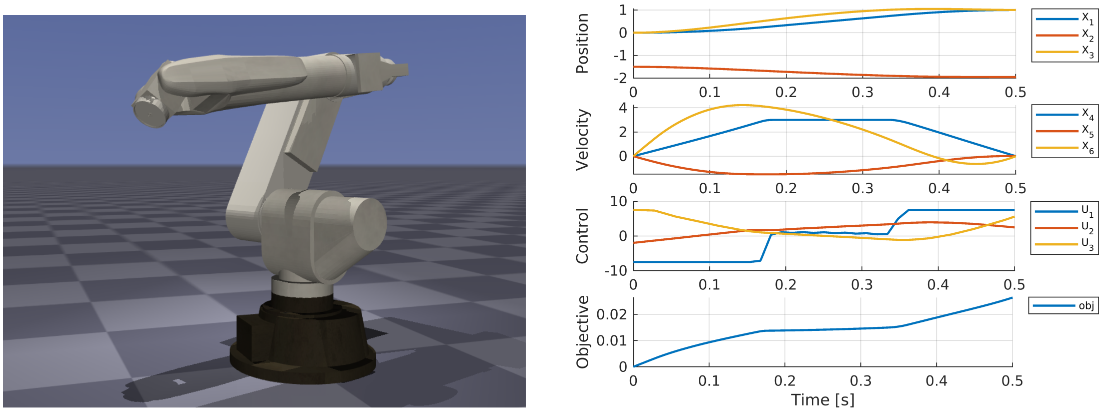
Figure 3.5.: Evaluation for the weakly actuated pendulum using a feedback control approximation based on MLP kernel function (no sparse approximation).

quality of the approximated feedback control. Almost all test instances failed using the Matérn-$5/2$, the RBF kernel, the Matérn-$3/2$ with DTCVAR and MLP with PITC.

It must be noted that the same training set (created using the non-sparse MLP covariance function) is used for comparing the different covariance functions and posterior variance approximations. The effect of the GP model on the start state selection for new trajectories during the creation of the training data set is therefore neglected. A more detailed analysis and comparison of kernel functions and options for GPs is performed in Chapter 5. The preliminary result that the MLP kernel is suitable for the approximation of a feedback control policy will be used in the next subsection, where the method is applied to a more complex dynamic model.

3.3.2. Feedback Control of the Manutec R3 Robot Arm

In the following, the presented approach is used to approximate an optimal feedback control of the point-to-point movement of an industrial Manutec r3 robot arm. The



(a) Visualization of the Manutec r3 robot arm.

(b) Nominal optimal joint trajectory for the Manutec r3 robot arm.

Figure 3.6.: Visualization and nominal joint trajectory of the Manutec r3 robot arm.

dynamic model of the Manutec r3 robot arm (consisting of the equations of motion and the joint constraints) is described in [136][1]. It has been formulated with focus on realism and is highly nonlinear, which makes finding a near-optimal control for this model significantly harder than for the weakly actuated pendulum. The robot arm has six joints, the first three mainly determine the position of the end-effector. The model of the Manutec r3 arm does not include models of the actuator and transmission dynamics. In principle, the modeling can be extended to also include higher order dynamics as well as bandwidth limitations introduced by the actuators. In this thesis, only these first three joints are used. Consequently, the state space is six-dimensional and the control space three-dimensional.

The optimal control problem is to find an energy- and time-minimal point-to-point movement of the end-effector from the joint state $x_0 = \begin{pmatrix} 0 & -1.5 & 0 & 0 & 0 & 0 \end{pmatrix}^T$ (hereafter referred to as *nominal start state*) to the final joint state $x_f = \begin{pmatrix} 1 & -1.95 & 1 & 0 & 0 & 0 \end{pmatrix}^T$. The cost function used throughout this thesis is

$$\mathcal{J}(x, u, t_f) = t_f + \rho \int_0^{t_f} \sum_{i=1}^{3} u_i(t)^2 \, dt \tag{3.7}$$

with $\rho = 10^{-3}$. It has also been used in [175] and leads to collision-free energy-minimal movements with an additional penalty for the free terminal time. The state and control

[1] The implementation used in this thesis can be found at `https://github.com/cztuda/semantic-feature-clustering/blob/master/cppsrc/models/manutec/_dynamics.h`.

	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$
x_{min}	-2.97	-2.01	-2.86	-3.1	-1.5	-5.2
x_{max}	2.97	2.01	2.86	3.1	1.5	5.2

(a) State Constraints

	$u[0]$	$u[1]$	$u[2]$
u_{min}	-7.5	-7.5	-7.5
u_{max}	7.5	7.5	7.5

(b) Control Constraints

Table 3.2.: Upper and lower bounds on the state and control variables of the Manutec r3 robot arm.

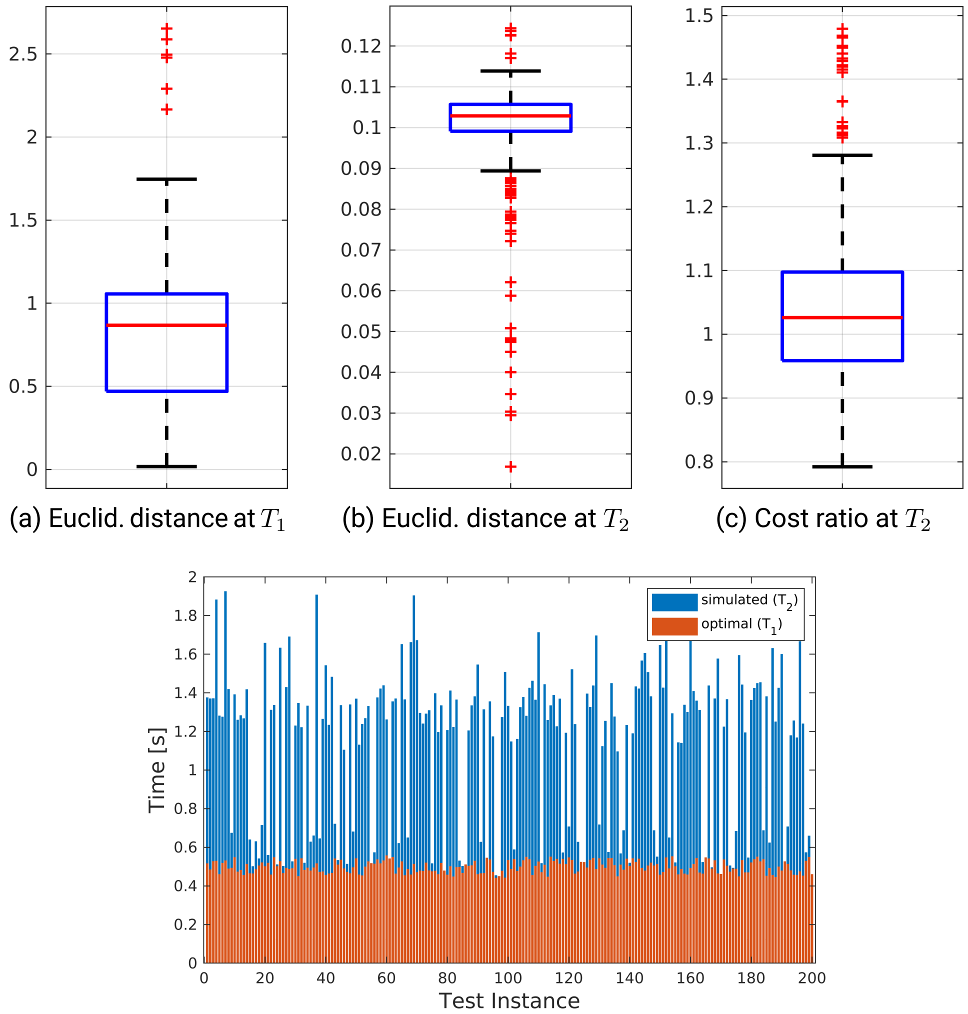
variables are constrained by the box constraints $x_{\text{min}} \leq x \leq x_{\text{max}}$ and $u_{\text{min}} \leq u \leq u_{\text{max}}$, the values are given in Table 3.2. The trajectory of the Manutec r3 robot arm performing an optimal point-to-point movement from the nominal start state is given in Figure 3.6b.

After 20 iterations of Algorithm 1, the near-optimal feedback control policy is trained with 869 data points from 18 optimal trajectories. Two generated trajectories have been rejected because the optimization process failed. Together with nine additional short optimal trajectories around the goal state, this results in a total of 1006 samples from 27 trajectories. The non-sparse GP with MLP covariance function is used again for the approximation.

The movement of the robot arm, controlled by the near-optimal feedback control, is simulated from 200 start configurations. The Euclidean distance of these start states to the nominal start state x_0 in joint space is between 0.1 and 0.3. The result is presented in Figure 3.7. The terminal time of the optimal trajectory of a test instance is designated with T_1, and T_2 is the time at which the minimum distance to the goal state is reached by the test instance. It can be seen that at the terminal time of the respective optimal trajectory, the test instances have a large distance from the goal state. This distance is reduced to 0.1 at T_2, but with a significant time delay. Considering the Lagrange part of the objective function (3.7), which is proportional to the energy cost, the approximated cost of the simulated trajectories are for three out of four test instances below $110\,\%$. However, the total cost of the simulated movements is much higher than the optimum, as it includes the final time.

3.3.3. Comparison with Naive Random Sampling

Finally, the presented approach is compared with a naive start state sampling strategy. Data from 16 trajectories that start at random start states in the joint space is used to train



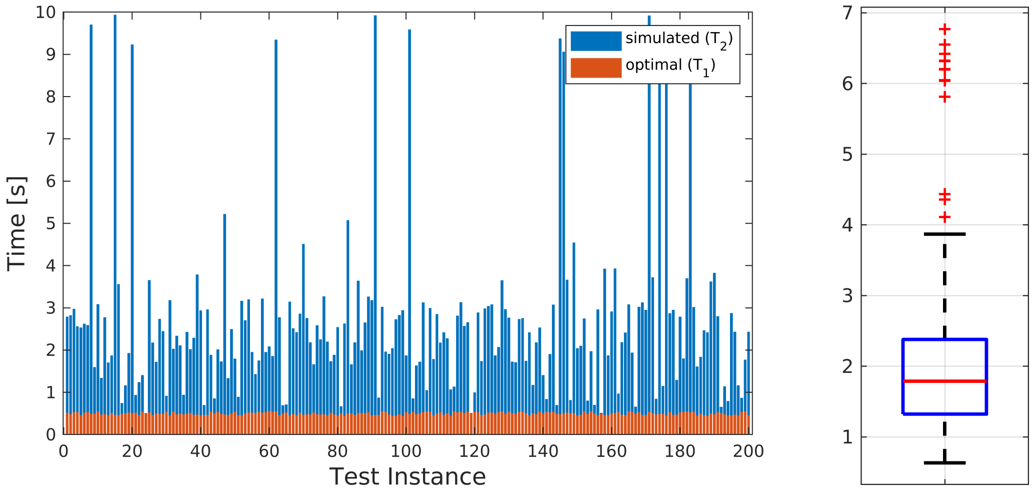
(a) Euclid. distance at T_1

(b) Euclid. distance at T_2

(c) Cost ratio at T_2

(d) Optimal and simulated terminal time T_1 and T_2 for all trajectories in the test set.

Figure 3.7.: Subfigs. 3.7a to 3.7c show the Euclidean distances from the goal state at optimal terminal time (T_1), the Euclidean distances at the time of shortest distance (T_2) and the ratio “simulated Lagrange-term cost to optimal Lagrange-term cost” for all instances of the test set. Subfig. 3.7d shows the optimal terminal time T_1 and the time T_2 at which the simulated trajectory reaches the shortest distance to the goal state for all trajectories in the test set.



(a) Time at which the closest trajectory point to the goal state has been reached

(b) Cost ratio at T_2 of the simulated to optimal trajectories

Figure 3.8.: Results for the naive sampling approach, showing the cost ratio and the terminal time of the test instances.

a GP with a MLP covariance function. These trajectories provide 877 data points, which is comparable to the number of data points used in the previous evaluation. The nine trajectories close to the goal state are reused, increasing the number of training samples to a total of 1013. The results on the test set of 200 start configurations is given in Figure 3.8. At T_1, the distance to the goal state is comparable to the distance achieved with the proposed method. At T_2 the distance is reduced to values between 0.12 and 0.15, which is slightly worse than with the adjoint-based start state selection approach. However, the naively learned policy needs much longer to bring the arm close to the goal state, in consequence, the overall cost at T_2 is typically also substantially higher.

3.4. Discussion and Conclusion

The results for the weakly actuated pendulum problem look convincing, but the results for the Manutec r3 robot arm show room for improvement. In particular, the time required to reach the shortest distance to the goal state is considerably longer than the optimal time to the goal state. A critical analysis of the approach employed in this chapter reveals the following reasons for the unsatisfactory performance of the learned control policy. The optimal control typically exhibits high variability in the small ball around the goal state. In higher-dimensional spaces, the few additional trajectories seem insufficient to cover the state space in a way that allows for the accurate representation of an optimal control policy that can effectively stabilize a more complex and higher-dimensional system around the goal state. Furthermore, the approach to filter redundant training data to prevent ill-conditioned GPs presented in Section 3.2.2 requires careful selection of a distance threshold. The threshold-based approach is thus, in some cases, too strict and still not able to reliably prevent ill-conditioned GPs in all cases. These two shortcomings will be addressed in the next chapter.

An important limitation of the approach presented in this chapter is its dependency on a sufficiently accurate model, which is used in the trajectory optimization step. This model may be difficult to obtain in practice. A possible solution is the implementation of a feedback compensation of model errors using a locally optimal tracking controller, e.g., similar to the approach introduced in [73]. In [72, 73], the cost function used for MPC has been extended to penalize high frequencies of the actuators in the trajectory. The influence of model inaccuracies on the learned feedback controller will be considered in more detail in Section 4.2.3 and 5.4.

In this chapter, different kernels and approximation methods for GPs have been tested on the weakly actuated pendulum problem and the MLP kernel has been identified as suitable for the approximation of the near-optimal feedback control policy. An iterative extremal field approach has been used to successively improve a near-optimal feedback control using samples from optimal trajectories. Focus has been on the placement of start states for new optimal trajectories. The proposed selection strategy employs co-state information from existing trajectories provided by DIRCOL, which is used to identify cost-sensitive parts of the trajectory, and the covariance of the current control policy approximation provided by the GP, which estimates uncertainty. This information is used for the first time to select new trajectories. The adjoint-based method of selecting start states for new trajectories avoids full sampling of the state space, making it an effective alternative to random sampling. It has been successfully applied to an industrial robot arm representing realistic highly nonlinear problems with time-invariant cost functions.

This chapter is based on the paper "Learning Control Policies from Optimal Trajectories", published in IEEE International Conference on Robotics and Automation (ICRA) *[213]. It contains the following major additions: an extended presentation of the related work, Figure 3.1, more details in Section 3.2.4, Figure 3.4, Table 3.1, revised Figures 3.5b to 3.5c, Figure 3.6, Figures 3.7a to 3.7c and Figure 3.8b.*

4. Complementing Start State Selection Methods and Explicit Goal State Handling

In Chapter 3, an iterative extremal field approach with a novel start state selection method is described to use data from optimal trajectories to learn a near-optimal state-dependent feedback control policy represented by a GP. The advantage of the extremal field approach is the direct incorporation of model knowledge into the construction of the control policy during training: Data from optimal trajectories is used that complies with the existing physical model of the system, which includes its dynamics and constraints. In this chapter, several improvements will be proposed regarding the start state selection method, the filtering of training data, and the stabilization of the system at the goal state. The advancements of this chapter extend and complement the approaches presented in Chapter 3.

The selection of start states determines the coverage of the state space with training data. An important insight that motivates this chapter is that more than one strategy is required to select new trajectory start states to ensure variety in the start state selection, as large co-state values are not the only reasonable criterion for good start states.

The evaluation in Chapter 3 shows that the stabilization around the goal state becomes increasingly difficult in higher dimensions. As described in Subsection 3.2.5, using optimal trajectories leads to an inhomogeneous sampling of the state space in the vicinity of the goal. In Chapter 3, this has been counteracted using additional short trajectories around the goal state. This approach becomes impractical for higher-dimensional robot models as the volume of a sphere with a fixed radius around some goal becomes large for higher dimensions. This is exactly the same reason why sampling of the state space is not used. In this chapter, a different approach will be proposed to solve this problem.

Furthermore, the GP used to approximate the feedback control policy becomes ill-conditioned when redundant training data is used. In Section 3.2.2, a minimum distance between the training data samples is used to filter all training data that fall below a predefined distance to an existing data point. However, this method depends on tuning of the minimum distance threshold; it is not able to prevent ill-conditioned GPs in all cases.

This chapter presents additional and complementing approaches to tackle the problems outlined above. The evaluation done for this chapter examines the effect of external perturbations and model inaccuracies caused, e.g., by inadequate or missing modeling of friction. The relevant related work has been discussed in detail in Section 2.2 and 3.1 and is therefore omitted in this chapter.

4.1. Extensions for the Iterative Extremal Field Approach

The selection of start states for new trajectories is crucial, as it determines the data that is added to the learning routine. For the method to be successful, there must be enough information in the relevant region. This requires the optimal trajectories to be distributed such that the learned policy can deal with unexpected deviations from the nominal path.

4.1.1. Complementing Strategies for Start State Selection

In Chapter 3, estimated co-state values (also called adjoint variables) have been used to indicate parts of the trajectory where some deviation has a considerable impact on the trajectory cost (see Section 3.2.3). However, the focus on the co-state variables that identify parts of the trajectory where deviations have a high impact on the cost may not account well for other aspects that are also important for an accurate approximation of the optimal control policy. In the following, various start state selection strategies intended to complement each other will be introduced. The strategies reflect the different requirements on the trajectories that provide data to learn the near-optimal policy.

Sensitivity-Based Start State Selection

The reason for the examination of co-states in Section 3.2.3 is to identify states with high sensitivity to the resulting trajectory cost, i.e., states that have a high impact on the resulting trajectory cost. However, it is also essential to consider the sensitivity to the state variable itself and to focus on regions where small deviations in the state of the trajectory may lead to large changes and thus notably change the trajectory's course from this point.

In the following, a notion of the relative perturbation error, or sensitivity of the state variables, will be used that follows the definition in [85]. Let $x(t_0 + t) = M^t_{f,u}(x(t_0))$ denote the numerical time integration for a fixed control function $u(t)$ of the differential equation $f(x, u)$ representing the dynamical system, starting from state $x(t_0)$.

For $\varepsilon > 0$ small, a fixed time window $\delta t > 0$, and the j-th column vector of the identity matrix e_j, the partial sensitivity approximation of f at state x is defined as

$$P_j^{M^{\delta t}_{f,u},\varepsilon}(x) = \frac{M^{\delta t}_{f,u}(x + \varepsilon e_j) - M^{\delta t}_{f,u}(x)}{\varepsilon}, \tag{4.1}$$

which is the relative perturbation caused by an error ε in the state variable $x_j(t)$ after numerical integration in $x(t + \delta t)$. Let further be

$$p_{M^{\delta t}_{f,u},\varepsilon}(x) = \begin{pmatrix} \|P_1^{M^{\delta t}_{f,u},\varepsilon}(x)\| \\ \vdots \\ \|P_n^{M^{\delta t}_{f,u},\varepsilon}(x)\| \end{pmatrix} \tag{4.2}$$

the sensitivity vector that holds the normed perturbation for all partial derivatives.

Note that the absolute error of the numerical integration scheme with respect to the perturbation error ε must be sufficiently small to provide a meaningful approximation of the sensitivity. For a given optimal trajectory $(x_{\text{ref}}(t), u_{\text{ref}}(t))_{t\in[0,t_f]}$ with terminal time t_f, a coarse time grid $\Gamma = \{t_0, t_1, \ldots, t_k\}$ with $t_0 = 0$ and $t_k = t_f$ is used.

Considering the sequence of sensitivity vectors $p_{M^{\delta t}_{f,u},\varepsilon}(x(t_0))$, ..., $p_{M^{\delta t}_{f,u},\varepsilon}(x(t_f))$ at the grid points Γ, assume that there is a distinguished peak at time $t_\text{p} \in \Gamma$. Identify $t_v \in \Gamma$ with $t_v < t_p$ such that $\|p_{M^{\delta t}_{f,u},\varepsilon}(x(t_i))\|$ is increasing for $t_\text{v} \leq t_i \leq t_\text{p}$ (see Figure 4.1).

Then the two points

$$x_{\text{ref}}(t_v) \pm \mu_{\text{sens}} \frac{p_{M^{\delta t}_{f,u},\varepsilon}(x(t_\text{p}))}{\|p_{M^{\delta t}_{f,u},\varepsilon}(x(t_\text{p}))\|} \tag{4.3}$$

are added to a strategy-specific set of candidate start states for new trajectories. The idea of equation (4.3) is to add a new trajectory in the direction of high sensitivity (at t_p) to increase the number of data points in this sensitive area. Parameter μ_{sens} controls the distance to the reference trajectory. To improve coverage of the full area around the peak, the point at $t_v < t_p$ on the trajectory is used (from which the sensitivity starts increasing) to move the start state of the new trajectory slightly “before” the peak.

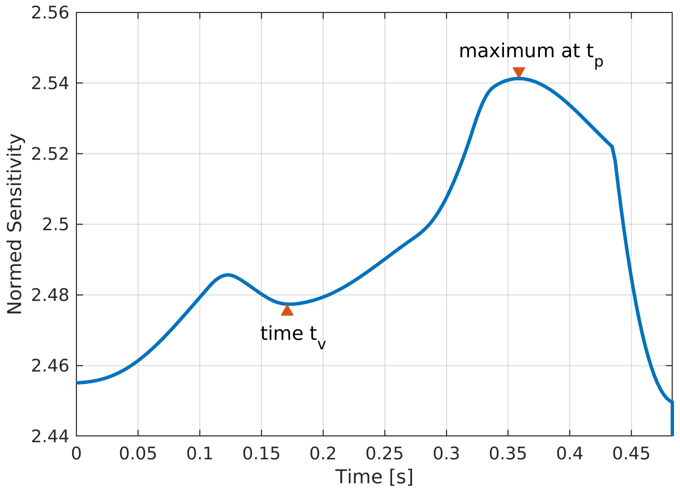


Figure 4.1.: Illustrative example of the selection of times t_p and t_v in the start state selection approach based on the sensitivity to state errors.

Simulation-Based Start State Selection

The rationale of the simulation-based approach is to check the tracking error of the control learned so far in simulation and add a new trajectory where the error starts growing significantly. For a given optimal trajectory $x_{\text{ref}}(t)$ with control $u_{\text{ref}}(t)$ and with terminal time t_f that has been computed in previous iterations, a trajectory is simulated using the currently learned control policy. The start state of the simulated trajectory is the same as that of the reference trajectory $x_{\text{ref}}(0)$. The deviation (i.e. the error) of the resulting trajectory $(x_{\text{sim}}(t), u_{\text{GP}}(t))$ from the reference trajectory is evaluated on a fine grid Γ. Let $t_{\text{p}} \in \Gamma$ be a distinguished peak in the sequence of deviations $\{d\,(t_i) : t_i \in \Gamma\}$ or the time where the deviation exceeds some predefined threshold $d_{\text{sim}}^{\text{max}}$ and let again $\Gamma \ni t_v < t_p$ be the time from which the deviation before t_p is strictly increasing, i.e., $\|d(t_{i+1})\| > \|d(t_i)\|$ for all $t_{\text{v}} \leq t_i < t_{\text{p}}$. A new candidate start state is then given as

$$x_{\text{ref}}(t_{\text{v}}) + \mu_{\text{sim}} \frac{d(t_p)}{\|d(t_p)\|}, \tag{4.4}$$

at a user-defined distance μ_{sim} from the reference trajectory in the direction $d(t_v)$ of the deviation occurring in simulation. This approach is illustrated in Figure 4.2. The rationale for equation (4.4) corresponds to that for equation (4.3).

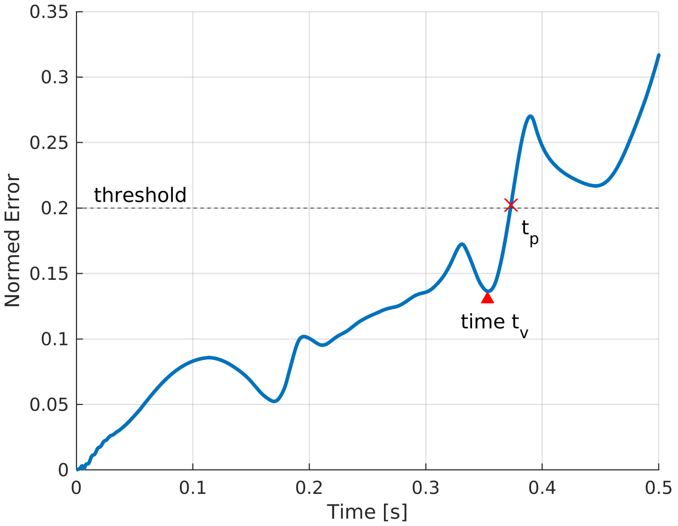


Figure 4.2.: Example of the selection of times t_p and t_v in the simulation-based start state selection approach. Time t_p can be at some maximum or (as in this example) when the threshold is exceeded.

Halton-Based Start State Selection

The start states generated from one of the two previously described methods are typically closer to the goal state than the start state of the original trajectory. To get some longer trajectories, the Halton-based selection strategy selects start states in the proximity of the initial, user provided start state x_0 of the first trajectory. To keep the method deterministic, the quasi-random Halton sequence [75] is used to compute start states for new trajectories:

$$x_0 + \mu_\mathrm{H} \frac{\mathcal{H}_i}{\|\mathcal{H}_i\|}. \tag{4.5}$$

Again, μ_H is some user-defined parameter that denotes the desired distance from x_0, $\mathcal{H}_i$ is the i-th element of the multi-dimensional Halton sequence.

An example of the sequence of start state candidates around the initial start state x_0, produced by the Halton-based approach, is depicted in Figure 4.3. Note that this approach is a simple (quasi-) random sampling strategy. Variance information from the learner is not taken into account. Instead, the next iterate of the Halton sequence is selected in each step. Further, in contrast to the other approaches presented here, this start state generation method also does not use information from already computed trajectories.

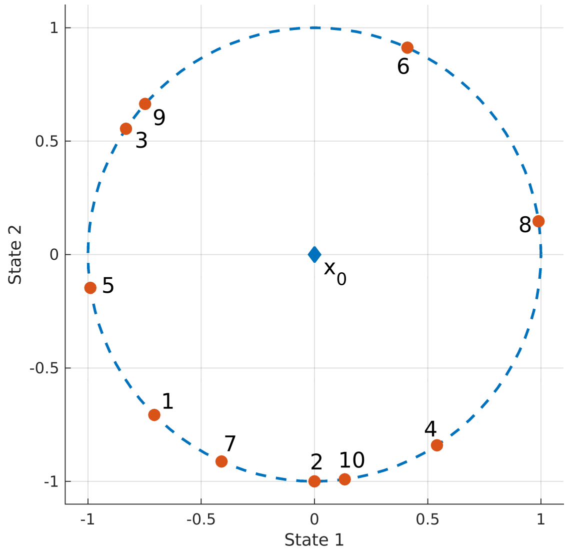


Figure 4.3.: Example of the Halton-based start state selection approach for the initial start state $x_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ (blue diamond). The sequence of the first ten new start states (numbered red dots) on a ball with radius μ_H around x_0.

Ensuring Feasibility and Selection of a new Start State

The new start states provided by the variants may be outside the problem's state bounds (2.3a). To handle this problem, one could discard the infeasible start states and select another start state using the same selection method. While this approach is perfectly feasible for the Halton-based approach, the adjoint-based, simulation-based and sensitivity-based start state selection methods would quickly run out of candidate start states. Instead, a correction is applied in this work to ensure that all start states are feasible. Assume that an infeasible state x_start has been derived using a start state selection strategy from some state x_ref (or x_0 for the Halton-based strategy). The corrected state x_corr is set to be the solution of the optimization problem

$$\begin{aligned} x_\text{corr} = \min_{\nu \in \mathbb{R}} \quad & x_\text{start} + \nu \left(x_\text{ref} - x_\text{start} \right) \\ \text{s.t.} \quad & x_\text{lb} \leq x_\text{start} - \nu \left(x_\text{ref} - x_\text{start} \right) \leq x_\text{ub}, \\ & 0 \leq \nu \leq \nu_\text{ub} \end{aligned} \tag{4.6}$$

if a solution exists. The upper bound $\nu_\text{ub} = 0.7 < 1$ is used to keep the resulting state away from x_ref. If no solution can be found, all entries in the vector x_start are forced separately to comply with the bounds: $x_{\text{corr}} = \min(\max(x_{\text{lb}}, x_{\text{start}}), x_{\text{ub}})$, which effectively skews the direction $(x_{\text{ref}} - x_{\text{start}})$ in which a new start state is placed.

The approach-specific set of candidate start states is expanded for the adjoint-based, sensitivity-based and simulation-based approach each time a new optimal trajectory is computed. If a new start state from one of these approaches is required, the element at which the current control approximation has the highest variance value (corresponding to uncertainty) in its candidate set is chosen. In each iteration of the algorithm, a new start state from one of the four proposed start state selection strategies is selected in a round robin procedure.

4.1.2. Filtering of the Training Data

A special requirement of GPs is that the set of training data $\{(\tilde{x}_i, \tilde{u}_i)\}_{i=1,\dots,N}$ must not contain duplicates, i.e.,

$$\forall i, j \in \{1, \dots, N\} : \|\tilde{x}_i - \tilde{x}_j\| > 0 \tag{4.7}$$

must hold. The reason is that, given some learning data $(\tilde{x}_i, \tilde{u}_i)_{i=1,\dots,N}$, predictions of Gaussian processes require the inversion of a matrix $\left(K + \sigma^2 I\right)$, where I is the identity matrix of suitable size, σ some constant denoting the noise and K is the Gramian matrix of the vectors $\tilde{x}_1, \dots, \tilde{x}_N$ with respect to the selected GP kernel function [154]. The Gramian K is positive semi-definite by construction and positive definite if and only if the vectors $\tilde{x}_i$ have a nonzero distance to each other. To get an invertible matrix $\left(K + \sigma^2 I\right)$ with the noise σ as small as possible, it is necessary to have only relevant and unique training data. In Chapter 3, a new data point $(\tilde{x}_{N+1}, \tilde{u}_{N+1})$ is rejected if its distance to another data point in the existing learning data falls below some constant $c > 0$, i.e.,

$$\min_{i=1,\dots,N} \{\|\tilde{x}_{N+1} - \tilde{x}_i\|\} < \text{c}. \tag{4.8}$$

However, the parameter c must be chosen conservatively to avoid a reduction in the quality of the GP's performance during the algorithm's execution. If parameter c is chosen too small, numerical problems caused by an ill-conditioned matrix inversion must be counteracted by introducing so-called jitter (addition of some constant value to the diagonal of the matrix) that can be interpreted as noise. This prevents effective training and deteriorates the training result. In contrast, training data is thinned out too aggressively and valuable information is thrown away if the parameter c is chosen too large. This makes it challenging to tune this parameter to learn the GP model successfully.

In the following, a less parameter dependent approach is proposed. This approach is inspired by a forum post on *Stack Exchange* by the user Jack Fitzsimons [1]. The new training data from a trajectory is added, the resulting kernel matrix K is computed and a QR decomposition[2] to identify a linearly independent subset of columns $C \subseteq \{1, \ldots, N\}$ in K is performed. The set of training data is restricted to $(\tilde{x}_i, \tilde{u}_i)_{i \in C}$, which ensures that the kernel matrix K stays invertible despite the addition of arbitrary new data points. This new approach more reliably filters training data that would lead to high noise. At the same time, it effectively removes fewer points than the approach in Chapter 3, such that more training data per trajectory can be used.

Marchildon and Zingg [114] propose a method to rescale the training data to avoid ill-conditioned kernel matrices in GPs. With their approach, all data points can be kept. However, their method is only applicable to rational quadratic and Matérn kernels. It needs to be investigated further whether their method can also be applied, with or without adaptations, to other kernels, such as MLP, which is frequently used in this thesis.

4.1.3. PI Control near the Goal State

Exact convergence towards and stabilization around the goal state x_f is difficult to achieve with the learned control, as learning data becomes sparse in its close vicinity. This has been described in detail in Section 3.2.5 where this problem has been mitigated using additional short trajectories sampled around the goal state to increase the number of training data in the sparse areas. However, there are more appropriate and bespoke methods to stabilize a controlled system around some goal state.

After the computation of the policy approximation, the error system is linearized around $x(t_f)$, which allows the use of a LQR approach to design a PI control law around the goal state x_{goal}. To deal with steady state errors (e.g., caused by model errors), the system is augmented to implement an integral action control. The augmented LQR problem is

[1] `https://stats.stackexchange.com/q/189816` (version: 2016-01-08)

[2] Matt J (2022). Extract linearly independent subset of matrix columns (`https://www.mathworks.com/matlabcentral/fileexchange/77437-extract-linearly-independent-subset-of-matrix-columns`), MATLAB Central File Exchange

given by

$$
\begin{aligned}
\min_{u} \quad & \int_0^\infty \hat{x}^T Q_x \hat{x} + \hat{u}^T Q_u \hat{u} + z^T Q_i z \, \mathrm{d}t \qquad (4.9)\\
\text{s.t.} \quad & \dot{x} = A\hat{x} + B\hat{u}, \; \dot{z} = q - q_{\text{goal}}, \\
& \hat{x}(0) = 0, \quad z(0) = 0 \\
& \hat{x} := x - x_{\text{goal}}, \quad \hat{u} := u - u_{\text{goal}} \\
& A = \left. \frac{\partial f(x,u)}{\partial x} \right|_{(x_{\text{goal}}, u_{\text{goal}})}, \quad B = \left. \frac{\partial f(x,u)}{\partial u} \right|_{(x_{\text{goal}}, u_{\text{goal}})}
\end{aligned}
$$

where z is the integrated error that augments the system. The dependency of x, u, z and q on the time t is omitted for brevity. The solution of the quadratic optimal control problem (4.9) is known to be

$$
\hat{u} = -Q_u^{-1} \begin{pmatrix} B \\ 0 \end{pmatrix}^T P \begin{pmatrix} \hat{x} \\ z \end{pmatrix}, \qquad (4.10)
$$

where $P \in \mathbb{R}^{n_y \times n_y}$ with $n_y := n_x + n_q$ is a positive definite, symmetric matrix that can be computed numerically by solving

$$
PA + A^T P - PBQ_u^{-1} B^T P + Q_x = 0, \qquad (4.11)
$$

which is known as the algebraic Riccati equation [5]. For the original variables, this gives the following feedback control law:

$$
u(t) = u_{\text{goal}} - K(x(t) - x_{\text{goal}}) - K_i z(t) \qquad (4.12)
$$

for gain matrices

$$
\begin{aligned}
K &= Q_u^{-1} \begin{pmatrix} B^T & 0 \end{pmatrix} P_{\{1,\ldots,n_x\}} \\
K_i &= Q_u^{-1} \begin{pmatrix} B^T & 0 \end{pmatrix} P_{\{n_x+1\}}
\end{aligned}
$$

where the indices $\{1,\ldots,n_x\}$ and $\{n_x+1\}$ denote the columns of matrix P. A detailed description of LQR and integral action control can be found in [5].

Like this, the learned control does not need to reach the exact goal position but only some ball around the final state, from which the control is passed to the computed PI controller. This renders the use of additional short trajectories around the goal state, as described in Section 3.2.5, unnecessary. This approach is only applicable if a fixed final state is given in the problem description. If some or all state variables are free, then it is unclear around which state to linearize.

	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$
x_{min}	-3.12	-2.11	-3.00	-3.15	-1.58	-5.46
x_{max}	3.12	2.11	3.00	3.15	1.58	5.46

Table 4.1.: The values of the box constraints on the state variables of the Manutec r3 robot arm relaxed by $5\,\%$.

4.2. Evaluation

The advancements detailed in the previous section are evaluated in simulation to obtain highly accurate information on the behavior of the investigated system. The Manutec r3 robot arm used in Chapter 3 is revisited to allow a comparison of the results. See Section 3.3.2 for a detailed description of the optimal control problem.

In simulation, small violations of the state constraints are tolerated and the feasible region defined by the box constraints is increased by five percent (see the values of the relaxed state constraints in Table 4.1). The reason for this relaxation is given later in the following subsection. If the learned controller exceeds the control bounds, the value is set to the boundary of the feasible region.

The methods are implemented and run in Matlab 2022b. As external code, the implementation of GPs provided by the *GPmat* toolbox written by Lawrence et al. [3] and the Fortran implementation of DIRCOL [173, 174] is used.

4.2.1. Performance of the Start State Selection Strategies

The first part of the evaluation aims at analyzing the contribution of each new start state generation strategy to the resulting learned control. The performance of the GP approximation that has been trained using all four proposed start state selection strategies is compared with the learned near-optimal control that has been trained alike but with one selection strategy missing. In the following, these five scenarios are referred to as *full*, *noAdj*, *noSens*, *noSim* and *noHalt*. An overview of the scenarios is given in Table 4.2. For each scenario, the iterative approach is stopped after 30 iterations, such that the number of trajectories and consequently the number of training data is approximately the same as in Chapter 4. The PI control close to the goal state presented in Subsection 4.1.3 is evaluated separately in Subsection 4.2.2 and not used in this part of the evaluation.

	full	*noAdj*	*noSens*	*noSim*	*noHalt*
Adjoint-based	X		X	X	X
Sensitivity-based	X	X		X	X
Simulation-based	X	X	X		X
Halton-based	X	X	X	X	

Table 4.2.: Overview of the five scenarios (given in columns) used in the evaluation of the start state selection strategies. The 'X's indicate which start state selection strategies (given in rows) are used in which scenarios.

For each optimal trajectory $(x_{\text{ref}}(t), u_{\text{ref}}(t))_{t\in[0,t_f]}$ of the test set, the movement of the robot arm is simulated, starting from the start state $x_{\text{ref}}(0)$, using all five learned control policies one after another. Simulations are performed by the `ode45` routine included in Matlab, which is based on the Dormand-Prince integration method (with relative error tolerance 10^{-8}, absolute error tolerance 10^{-9}). For this evaluation, the switch to the PI control close to the end state x_{goal} is not used to avoid tampering the result. Instead, for a simulated trajectory $x_{\text{sim}}(t)$, the time at which the minimal distance between the simulated trajectory and the goal state occurs

$$t'_{\text{f,sim}} := \arg\min_t \|x_{\text{sim}}(t) - x_{\text{goal}}\| \tag{4.13}$$

is considered as final time. The distance at the final simulation state to the goal state is consequently given as

$$d := \|x_{\text{sim}}(t'_{\text{f,sim}}) - x_{\text{goal}}\|. \tag{4.14}$$

A simulation is performed successfully, if this distance falls below some threshold $d < c_{\text{succ}}$. This threshold is set to be $c_{\text{succ}} = 0.15$ since this is the distance at which the switch to the PI controller will be performed in the second part of the evaluation (cf. c_{in} at the end of Section 4.2.2). In this part, the integration method stops as soon as a constraint violation occurs and the simulation is marked as failed.

For the comparison of the five learned control policies, the following performance criteria are considered:

1. Distance of final state in simulation from the goal state:

$$\|x_{\text{sim}}(t'_{\text{f,sim}}) - x_{\text{goal}}\| \tag{4.15}$$

2. Ratio of terminal times:

$$t'_{\mathrm{f,sim}}/t_{\mathrm{f,ref}} \tag{4.16}$$

3. Tracking error using normalized mean squared error (NMSE) on a time grid:

$$n^{-1} \cdot \mathrm{Var}\left[\left\{x_{\mathrm{sim},i,j}\right\}_{i,j}\right]^{-1} \sum_{i=1}^{n} \sum_{j=1}^{n_x} \left|x_{\mathrm{sim},i,j} - x_{\mathrm{ref},i,j}\right| \tag{4.17}$$

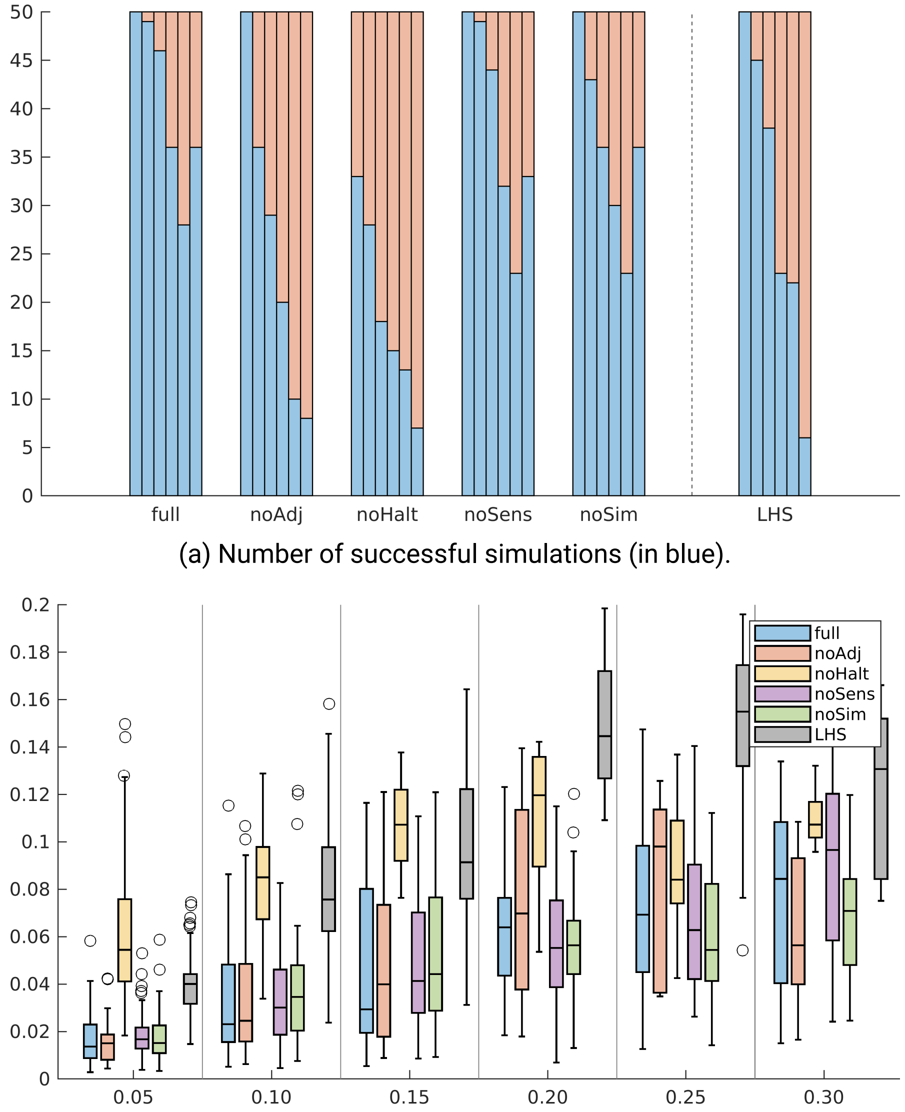
Note that the definition of the NMSE in this thesis differs from that in [214] to be consistent with the definition used in Chapter 5 and the computation of the mean square error in Tensorflow[3]. Figure 4.4d is changed accordingly.

A test set of 300 optimal reference trajectories from random start states with a defined distance to the problem start state x_0 in the joint space, as given in (2.1), is used to evaluate the performance of the five scenarios introduced above. It can be expected that the difficulty of the test instance depends on the distance of the reference trajectory's start state from x_0. For this reason, the test set consists of six groups of 50 trajectories, where the start state of all trajectories in a group has the same distance from the initial start state. The Euclidean distances are 0.05, 0.10, 0.15, 0.20, 0.25 and 0.30. This allows the interpretation of the result depending on different levels of difficulty.

The following parameters are used for the four start state selection methods: adjoint-based: $[b_l, b_u] = [0.2, 0.3]$, $\nu = 1.2$; sensitivity-based: $\mu_{\mathrm{sens}} = 0.18$; Halton-based: $\mu_{\mathrm{H}} = 0.1$; simulation-based: $\mu_{\mathrm{sim}} = 0.2$, $d_{\mathrm{sim}}^{\max} = 0.2$. These parameters influence at which distance to existing trajectories new start states are placed, which determines its usefulness. Consequently, the performance of the complementing methods strongly depends on the parameter tuning.

The results for each scenario are summarized in Figure 4.4. The Figures 4.4b to 4.4d show only the results for the successful trajectories. The y-axes of Figures 4.4c and 4.4d are zoomed in and show only the relevant part of the data. At a close look, it can be seen in the results that the cost or terminal time ratio between simulated and optimal trajectories is slightly below 1.0. This is not a flaw in the trajectory optimizer DIRCOL but results from the fact that, by accepting all trajectories that reach some region around the goal state, the original problem formulation has been relaxed. This makes it possible that some simulation results "outperform" the optimal solution that actually reaches the goal state.

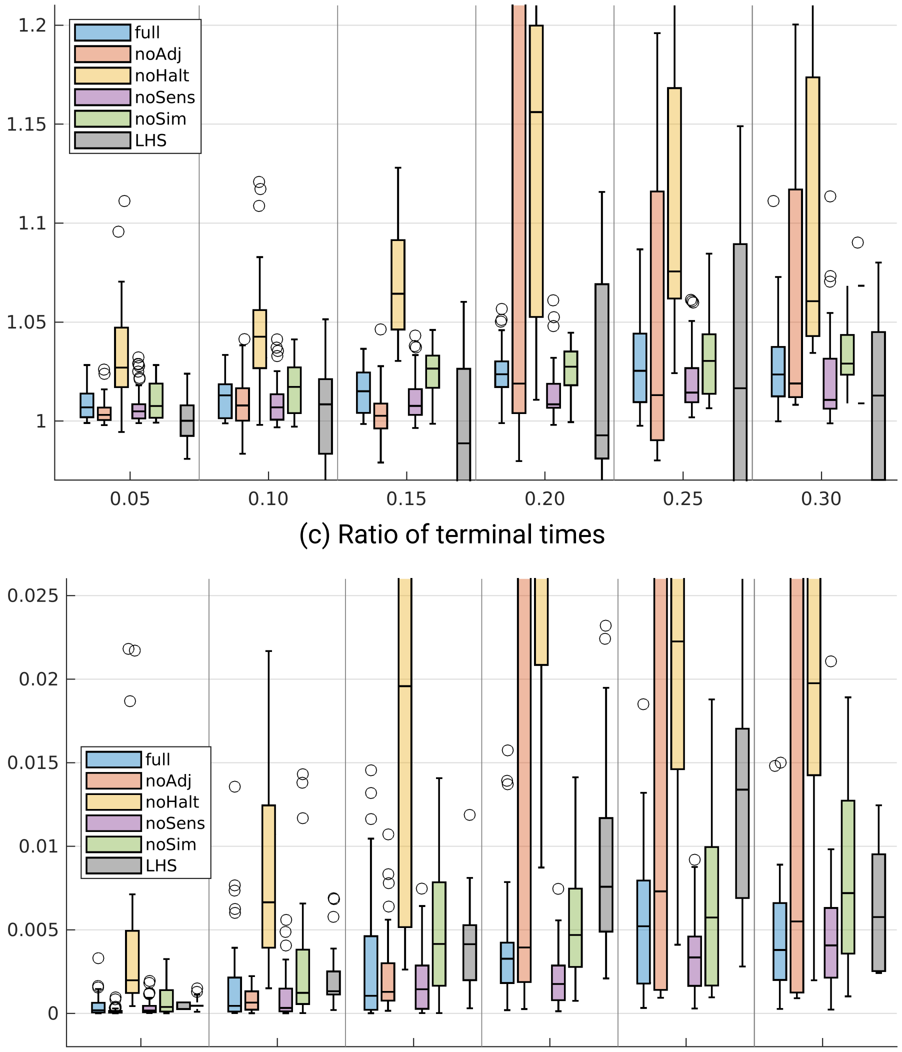
[3] https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredError



(a) Number of successful simulations (in blue).

(b) Distance at closest state

Figure 4.4.: Results for five learned control policies with all start state generation strategies (*full*) or all but one strategy (e.g., *noAdj* means all strategies except adjoint-based etc.). *LHS* gives the results for a state-of-the-art local sampling approach (Sec. 4.2.1). Subfig. 4.4a visualizes the success rate for six different distances of start states from the nominal start state. Subfig. 4.4b presents the results for the performance criterion (4.15); the boxes encompass the interquartile range (IQR), the maximal whisker length is $1.5\,\mathrm{IQR}$, outliers are marked with $\circ$.



(c) Ratio of terminal times

(d) NMSE of the system state

Figure 4.4.: (continued) Results for five learned control policies with all start state generation strategies (*full*) or all but one strategy (e.g., *noAdj* means all strategies except adjoint-based etc.). *LHS* gives the results for a state-of-the-art local sampling approach (Sec. 4.2.1). Subfigs. 4.4c to 4.4d present the results for the performance criteria (4.16) to (4.17). The boxes encompass the interquartile range (IQR), the maximal whisker length is $1.5\,\mathrm{IQR}$, outliers are marked with $\circ$.

The bar plot shows the number of successfully solved test instances. The three box plots visualize the result for all scenarios (see Table 4.2) with respect to the final trajectory distance to the goal state (see (4.15)), the ratio of simulated to optimal trajectory time (see (4.16)) and the NMSE between the states of simulated and optimal reference trajectories (see (4.17)). The scenario *full* gives the highest number of successes and the best end positions for the three shortest start state distances. The results are average for more distant start states and the ratio of terminal times. If the sensitivity-based strategy is missing, the ratio of terminal times is among the best, indicated by a low mean and variance. However, the number of successes is reduced compared to the results where this strategy is included. The distance to the goal state is average. The success rate of the *noSim* scenario is average, and the distance to the goal state is small compared to the other examined scenarios. For both criteria, *noSim* performs worse than *full* and *noSens*. In addition, it has notably higher terminal times and considerably increased NMSEs compared to *noSens* and *full*. Removing the adjoint-based or Halton-based strategy considerably reduces the success rate, which underlines the importance of these strategies. Although the *noAdj* scenario shows decent terminal time ratios (favorable means, but very high variance for test instances with more distance start states), the NMSE is only average. The results for *noHalton* are the worst of all examined scenarios for all criteria.

Overall, *full* and *noSens* are clearly the best scenarios in this evaluation; they show similarly strong results and give a reasonable trade-off between the evaluated performance criteria (4.15) – (4.17). While *noSens* shows slightly better optimality properties (terminal time ratio and NMSE), *full* has a slight advantage in the success rate and the shortest distance to the goal state. However, the latter is more important than moderately better optimality. The learned control generated with the *full* scenario is thus used in the subsequent evaluation.

Constraint Violations in the Performance Evaluation

State constraints are implicitly incorporated into the learned control policy, as the data that is used for training satisfies these constraints. However, they are not enforced during the execution of the learned control policy, so violations may still occur. In contrast to the evaluation done in Chapter 3 where a self-written classical fourth-order Runge-Kutta scheme (RK4) with fixed step size is used to simulate the systems, the simulation routine in this chapter is the built-in MATLAB function `ode45`. These two integration methods behave differently if the state violates a constraint: The self-written RK4 approach resets the system state onto the boundary of the feasible region when this region is left and

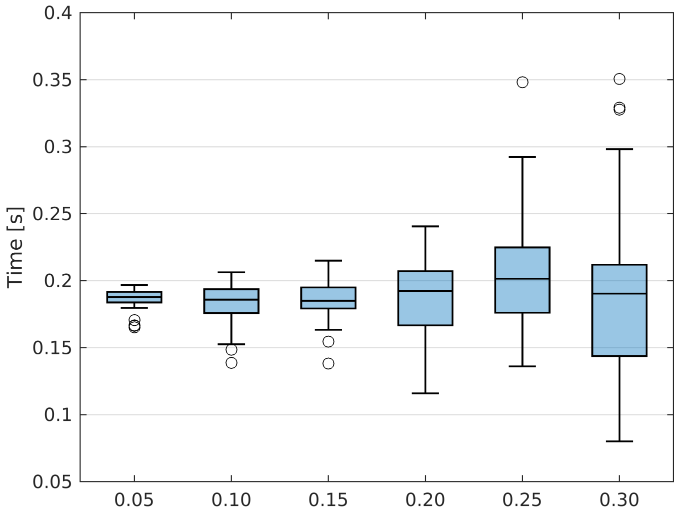


Figure 4.5.: Length of the constrained arcs of the test instances depending on the distance of their initial start state from x_0

continues the simulation. The `ode45` function, on the other hand, stops the simulation as soon as a constraint violation occurs. This typically happens with considerable distance to the goal state such that this instance is marked as a failure. For this reason, the state constraints are relaxed by increasing the feasible region by $5\,\%$ to allow small violations of the state constraint. The values of the relaxed state constraints are given in Table 4.1.

Hence, while there are no failures due to violations of the state constraint in the evaluation of Chapter 3, this occurs more frequently in the evaluation of this chapter. In fact, failure due to the violation of state constraint is the main reason why instances fail in the evaluation in Section 4.2.1: The reason for the failure of 53 out of 55 failed test instances is the violation of the relaxed state constraint. Thus, only two simulations that did not reach the goal state were not aborted prematurely.

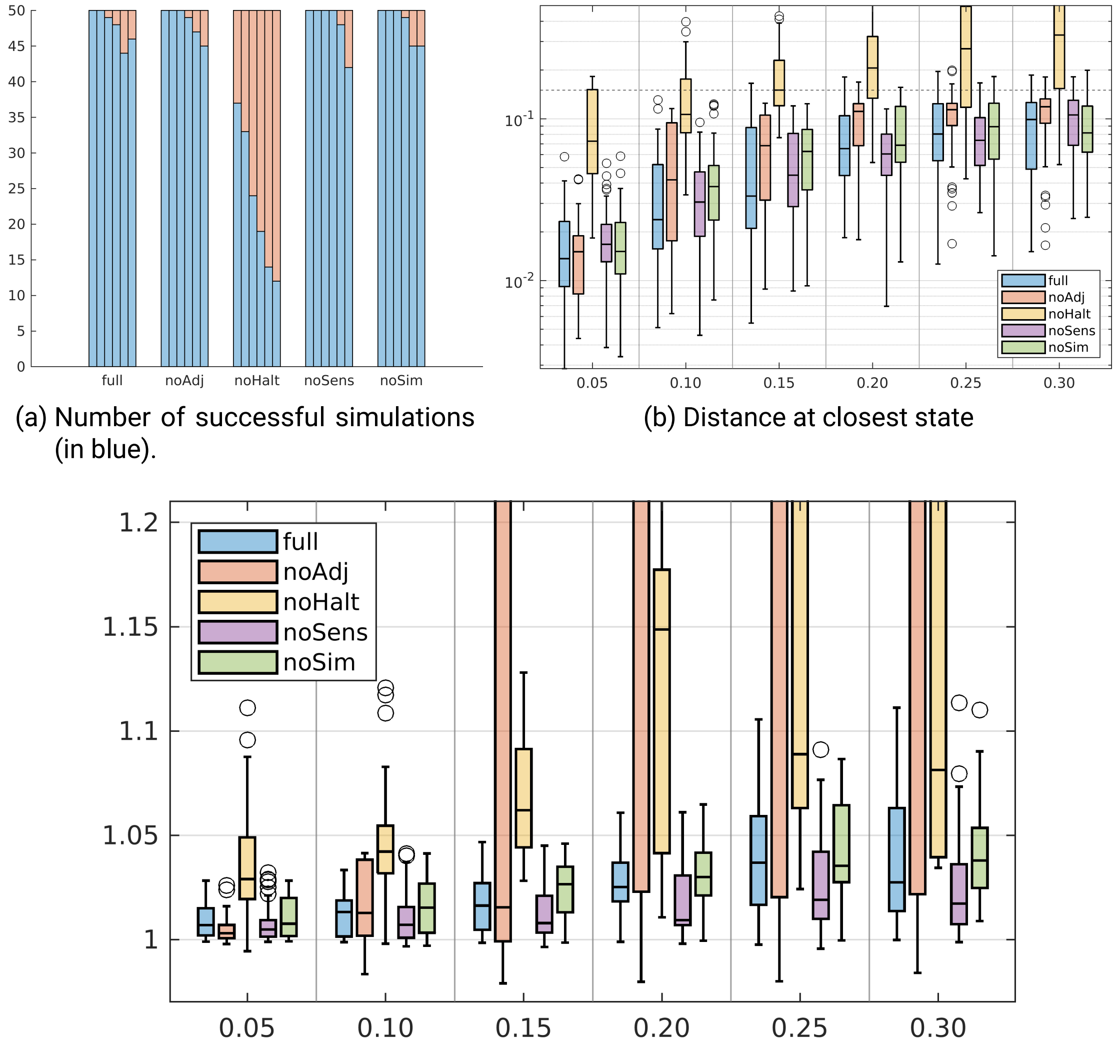
This means that the categorization according to the distance of the start state from x_0 is not sufficient to rate the difficulty of the test instances. It also depends on how close trajectories get to the infeasible region and how long the constrained arcs in the trajectories are. To analyze the difficulty of the test set regarding state constraints, the total time in which some state constraint is active is approximated for each trajectory. The result is summarized as a boxplot for each group of the test set in Figure 4.5. It can be seen that the variance increases with increasing distance from x_0. Further, the median time, in which the state is constrained, is in the "0.25" group substantially higher (0.2 s) than in all other groups (the median for trajectories with distance 0.05 is 0.188 s and the median for distance 0.3 is 0.19 s). If the trajectories with a distance of 0.25 spend more time in a critical state close to the boundary, then the difficulty of these test instances is higher, and consequently, the success rate is potentially lower.

To remove the impact of the state constraints on the evaluation result and evaluate how the learned controls perform on unbounded problems, the evaluation from Section 4.2.1 is repeated without state constraints (i.e., by relaxing the state constraints by a factor of 500). The results are given in Figure 4.6. The overall success rate improves significantly compared to the evaluation with state bounds (compare with Figure 4.4a) and the differences between the different scenarios (except for the *noHalton* scenario, which still has a remarkably low success rate) are negligible. The *full* scenario also gives the lowest median of the shortest distance to the goal state. Its terminal time ratio is decent but (as in the state-constrained case) exceeded by the *noSens* scenario. Since the state bounds are not enforced in this evaluation, the constraint violations can be quantified. The maximum constraint violation on the simulated trajectories of the test set is given in Subfigure 4.6d. The differences between the scenarios are minor, apart from *noAdj* that is notably increased. The mean constraint violation (given in Subfigure 4.6e) is lowest in the *full* scenario and notably highest in *noHalt*.

While the scenario with all start state selection strategies performs slightly worse than most other scenarios regarding success rate, it outperforms all other strategies in terms of the shortest distance to the goal state and mean constraint violation. It is among the best in terms of optimality (terminal time ratio) and maximum constraint violation.

Comparison with the Adjoint-Based Start State Selection Strategy from Chapter 3

To show that the new start state selection strategies improve the results achieved in the previous chapter, the learned control created for the *full* scenario is evaluated on the test set constructed for Chapter 3. This test set consists of 200 optimal trajectories whose start states' distances from x_0 range from 0.1 to 0.3. Further, the evaluation code from the Section 3.3 is used. In particular, simulations in this part rely on the self written integration scheme that has been used for the evaluation in Section 3.3. It must be noted that in Chapter 3, nine distinct trajectories around the goal state x_f have been added to improve convergence. These additional trajectories are not used in this chapter.

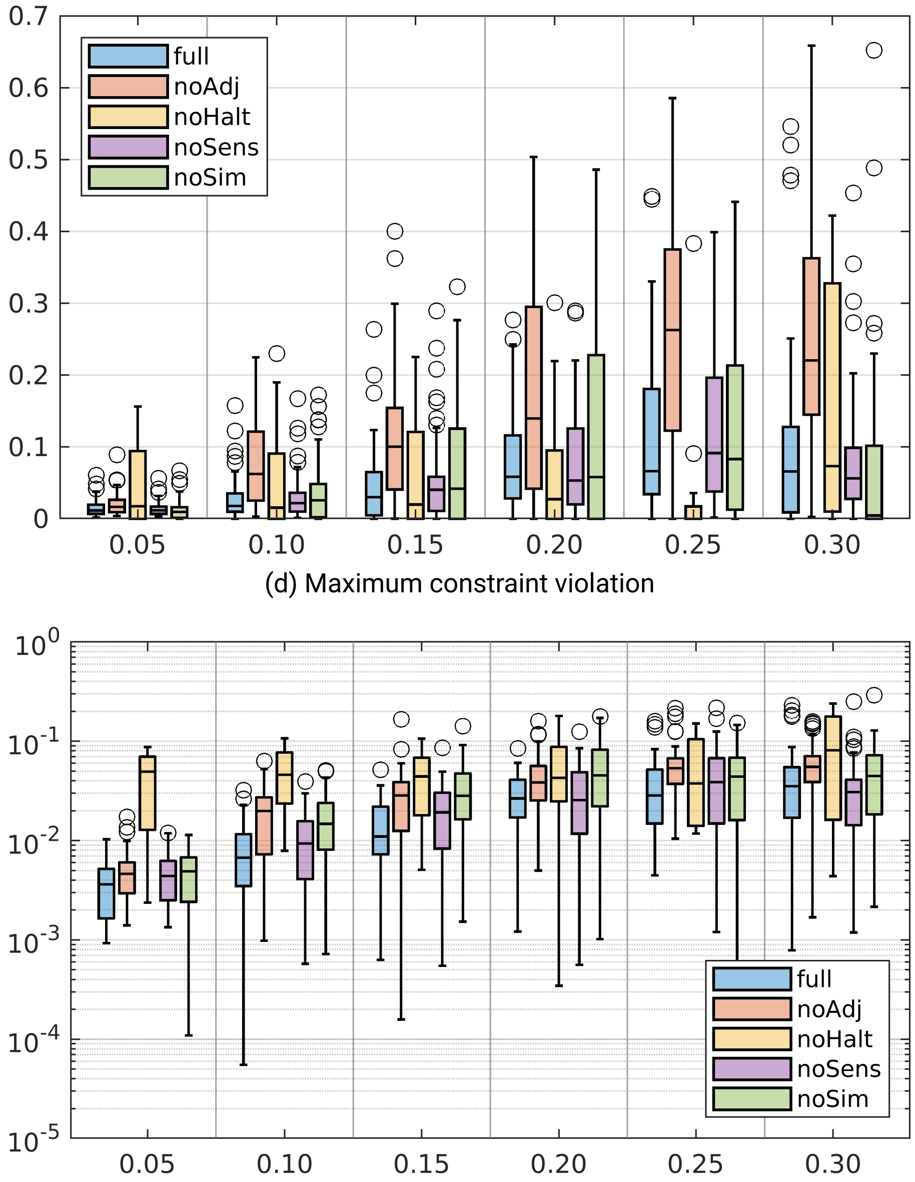


(a) Number of successful simulations (in blue).

(b) Distance at closest state

(c) Ratio of terminal times

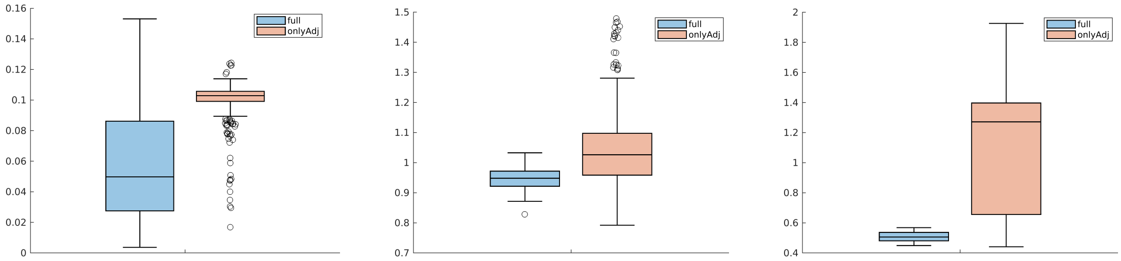
Figure 4.6.: Comparison of the five learned control policies with all start state generation strategies on practically unbounded state. Subfig. 4.6a visualizes the success rate for six different distances of start states from the nominal start state. Subfigs. 4.6b to 4.6c give the results for the performance criteria (4.15) to (4.16). Subfig. 4.6c shows only the values for successful trajectories. The boxes encompass the interquartile range (IQR), the maximal whisker length is $1.5\,\mathrm{IQR}$, outliers are marked with $\circ$.



(d) Maximum constraint violation

(e) Mean constraint violation

Figure 4.6.: (continued) Comparison of the five learned control policies with all start state generation strategies on practically unbounded state. The Subfigs. 4.6d and 4.6e show the hypothetical constraint violation for the state-constrained problem. They show only the values for successful trajectories. The boxes encompass the interquartile range (IQR), the maximal whisker length is $1.5\,\mathrm{IQR}$, outliers are marked with $\circ$.



(a) Shortest Distance (b) Cost Ratio (c) Terminal Time

Figure 4.7.: Comparison of results using only the adjoint-based start state selection method presented in Chapter 3 (*onlyAdj*) with the results of the approach using four different selection methods from this chapter (*full*).

The movement of the robot arm is simulated from the start states in the test set using the learned control. The evaluation focuses on the shortest distance from the goal state that is reached by the simulated arm, on the ratio of the cost at the shortest distance to the goal state and on the times at which the shortest distance to the goal state is reached (i.e., the terminal time of the simulated movement). The results are presented in Figure 4.7. The new start state selection strategies significantly improve, apart from some outliers, the shortest distance to the goal state, the cost ratios, and the terminal times of the simulated trajectories: The shortest distance to the goal state improves for 160 of the 200 test instances compared to the result achieved in Chapter 3. Furthermore, the shortest distance to the goal state is reached faster and with a lower cost value.

Comparison with LHS Sampling

Random sampling of the joint space is commonly used in the extremal field approach to select the start states for optimal trajectories. Advanced approaches sample only in a small subspace to focus on the region around some nominal trajectory. Ghosh and Conway [64] use Latin hypercube sampling (LHS), which is very suitable to be used in high-dimensional joint spaces, as it “simultaneously stratifies on all input dimensions” [200]. They use the *maximin* criterion for LHS to ensure good coverage of the space.

The use of the presented four complementing start state selection methods in the extremal field approach is compared with LHS sampling using the maximin criterion on the test set of 300 optimal trajectories. The number of sampled start states is 30, such that the number of training samples is roughly the same as in the evaluation described in Section

4.2.1. The performance of the learned feedback control policy using LHS sampling to generate start states is evaluated with respect to the three performance criteria, given by Equations (4.15) to (4.17). The result is shown in Figure 4.4.

The success rate for the test instances with start states close to the nominal state is very high if the near-optimal feedback policy is trained using LHS samples. However, the success rate drops for test instances that are further away. The minimum distance of the successful trajectories to the goal state is considerably larger, and the tracking error is slightly larger than for the *full* scenario. On average, this minimum distance is reached in less time than with the proposed method.

The evaluation shows a clear benefit of using multiple complementing start state selection methods, where the sampling strategy is combined with approaches that use information from the numerical optimal control solver or from already computed optimal trajectories.

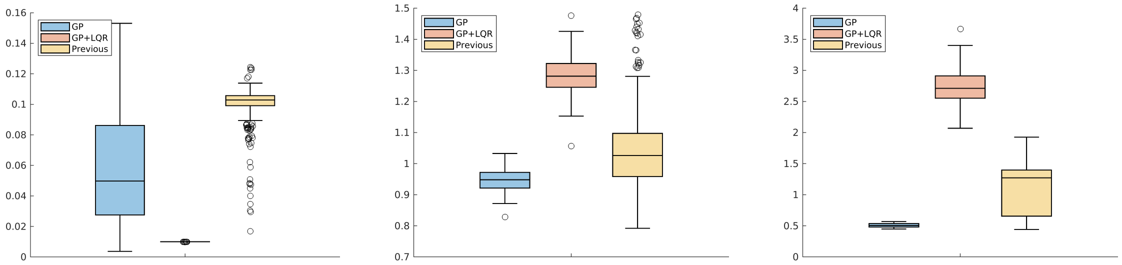
4.2.2. Evaluation of the PI Control near the Goal State

In this subsection, the performance of the learned control policy combined with a linear feedback controller around the goal state x_{goal} is evaluated. An LQR as given in Equation (4.9) is used to design PI gains that can be applied in the close proximity of x_{goal}. The weight matrices Q_x, Q_u and Q_i are diagonal matrices with values as follows:

$$\begin{aligned}
Q_x &= \text{diag}\left(\begin{bmatrix}80 & 350 & 100 & 0 & 0 & 0\end{bmatrix}\right)\\
Q_u &= \text{diag}\left(\begin{bmatrix}1.0 & 1.0 & 1.0\end{bmatrix}\right)\\
Q_i &= \text{diag}\left(\begin{bmatrix}10 & 4500 & 800 & 0 & 0 & 0\end{bmatrix}\right)
\end{aligned}$$

MATLAB's routine `lqr` is used to solve the resulting algebraic Riccati equation (4.11) and get the gain matrices K and K_i that determine the state space controller from Equation (4.12). For each trajectory in the test set, the simulation starts with the trained controller evaluating the GP and switches to the PI controller (4.12) as soon as the normed distance of the first three entries of the system state (which are the joint positions) to the goal state falls below some threshold c_{in}. Once the system has switched to the linear controller, it only switches back to the learned controller if the normed distance of the full system state to the goal state exceeds c_{out}.

- Start LQR control when $\|x_{1,2,3}(t) - x_{\text{goal},1,2,3}\| \leq c_{\text{in}}$
- Leave LQR control when $\|x(t) - x_{\text{goal}}\| > c_{\text{out}}$



(a) Shortest Distance

(b) Cost Ratio

(c) Terminal Time

Figure 4.8.: Comparison of results using only the adjoint-based start state selection method presented in Chapter 3 (*onlyAdj*) with the results of the approach using four selection methods with (*full+PI*) and without (*full*) PI control.

It is said that the robot approaches the goal state if the distance of system state to the goal state falls below c_{in}, and that the system reaches the goal state if this distance is smaller than 10^{-2}. In general, it is advisable to choose $c_{\text{out}} > c_{\text{in}}$ to avoid that the control quickly alternates between the learned and the PI controller. In this evaluation, the simulation is stopped and the respective instance of the test set is marked as failed as soon as the PI controlled system leaves the ball with radius c_{out} around the goal state.

The movement of the robot arm is simulated starting from the start states of the trajectories in the test set from Chapter 3 and the times needed to reach the goal state are compared. The values $c_{\text{in}} = 0.15$ and $c_{\text{out}} = 0.6$ turned out to be viable. The results are given in Figure 4.8. The trajectories reach the goal state (distance below 1×10^{-2}), improving the final distance compared to the simulation without PI control approximately by a factor of five. This comes with an increased cost and terminal time.

To further analyze the impact of the PI controller on optimality, the trajectory that results from the PI controller (starting at the time t_{switch} at which the switch from the learned to the PI controller occurs) is compared to the optimal trajectory starting from the same state. This analysis considers only the time required to reach the goal state as criterion, which has a large influence on the trajectory cost (see the cost function (3.7)). Figure 4.9 shows the result for all test instances. The blue bars give the switching time t_{switch} and thus represent the fraction of the execution time in which the learned feedback control is active. The red bars show the time taken by the optimal trajectory from $x\left(t_{\text{switch}}\right)$ to $x\left(t_f\right)$, and the yellow bars show the time taken by the PI controller to reach the goal state starting from the state at which the switch occurred. The green lines give the terminal time of the entire optimal trajectory starting from the start state $x(0)$ of the test instance.

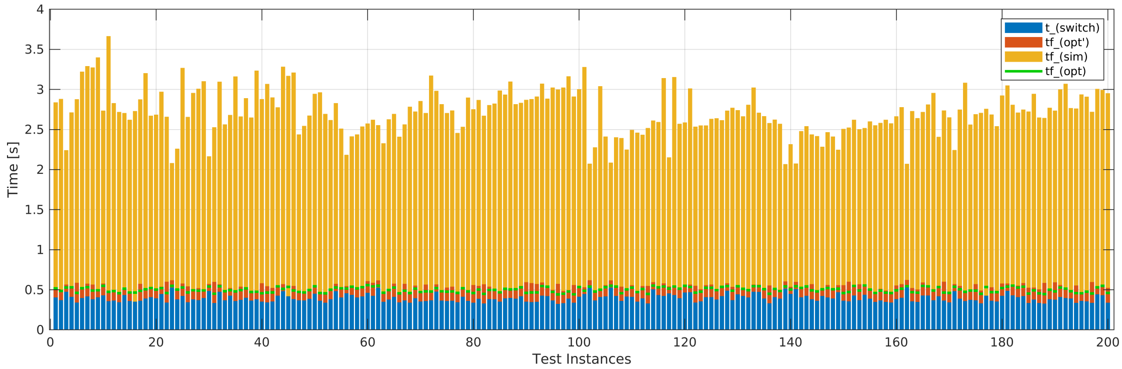


Figure 4.9.: Execution time of learned controller, PI controller and optimal control on the test instances of the Manutec r3 robot arm. The blue bars show the time in which the learned feedback controller is active until the switch to the PI controller occurs. Starting at the state $x\left(t_{\text{switch}}\right)$, the red bars give the time of the optimal trajectory to reach the goal state and the yellow bars the time the PI controllers need to reach the goal state up to 1×10^{-2}. For comparison, the green line gives the terminal time t_f of the entire optimal trajectory starting at the start state x_0 of the test instance.

It can be seen that the PI controller takes about 18 times longer than the optimal controller to reach the goal state. The PI controller is only active in a small ball around the goal state, but takes about $85\,\%$ of the overall trajectory time. Thus, a significant part of the suboptimality is due to the PI controller taking a significant amount of time to reach the goal state. The performance of the PI controller depends on its tuning, if higher gains and joint velocities are accepted than it is able to reach the goal state faster.

4.2.3. Perturbed Dynamics and Exogenous Perturbations

Perturbations acting on the system can be caused by exogenous forces and forces due to imperfect modeling of the system dynamics. In this section, the performance of the near-optimal feedback control policy computed for Section 4.2.1 under these perturbations is analyzed. The exogenous perturbation is simplified to a displacement of the system state during execution of a trajectory, representing, for example, an idealized collision with a lightweight object.

Displacement in the System State during Execution of the Learned Control

The focus of the evaluation in Section 4.2.1 is on perturbations of the trajectories' start states. This section will continue this investigation and also consider perturbations in the course of the trajectories. These perturbations are simplified to be discontinuous displacements or shifts of either the joint states or velocities. They can be interpreted as sudden shifts or corrections in the state estimation or as short collision with a lightweight object that causes an 'instantaneous' change in the joint velocity. The perturbations are applied to the optimal point-to-point movement of the Manutec r3 robot arm that solves the OCP described in Section 3.3.2.

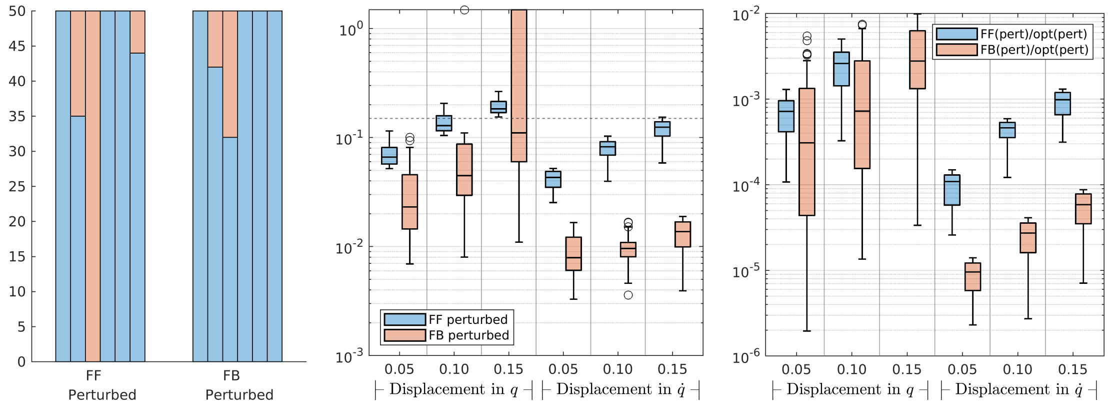
To evaluate the impact of displacements representing exogenous perturbations during execution on the performance of the learned feedback control, a test set of 300 displacements of three different magnitudes is created, similar to the test set described in Section 4.2.1. All trajectories in the test set start from the same start state at $x_0 = \begin{pmatrix} 0 & -1.5 & 0 & 0 & 0 & 0 \end{pmatrix}^T$, but are subject to different displacements that affect either the joint states or the velocities. The optimal unperturbed trajectory starting at x_0 has a length of about 0.5 s; the displacements are applied after exactly 0.2 s in simulation. For a given magnitude $\mu > 0$, a displacement v_{pert} for the joint state or the joint velocity is created as follows using a uniformly sampled random number $r \in \mathcal{U}\left([0,1]\right)$:

$$\hat{v}_{\text{pos}} = \begin{pmatrix} -1+2r \\ -1+2r \\ -1+2r \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \hat{v}_{\text{vel}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1+r \\ r \\ -1+2r \end{pmatrix}$$

$$v_{\text{pert}} = \frac{\mu \hat{v}_X}{\|\hat{v}_X\|} \quad \text{for } X \in \{\text{pos}, \text{vel}\}\,. \tag{4.18}$$

Equation (4.18) provides samples of a fixed magnitude and ensures that the displacement does not violate the state constraint (in particular the velocity limits for x_4 and x_5 that are active at $t = 0.2$ for the given trajectory, cf. Figure 3.6b). To create the test set, 50 displacements of magnitude $\mu = 0.05$, $\mu = 0.1$ and $\mu = 0.15$ are sampled using Equation (4.18) for both joint position and velocity, resulting in 300 perturbations.

The arm motion is simulated for 0.2 s starting from x_0 to some state $x(0.2)$ controlled by the open-loop optimal trajectory (FF) or by the near-optimal feedback control policy (*full* scenario) used in Section 4.2.1 (FB). At $t = 0.2$, the displacements from the test set are



(a) Number of successful simulations (blue)

(b) Shortest distance to the goal state

(c) NMSE of the system state

Figure 4.10.: Results for the feedforward optimal control (FF) and the near-optimal feedback policy (FB) subject to a displacement of the joint state q or joint velocity $\dot{q}$ occurring at $t = 0.2$ during motion. Subfig. 4.10a visualizes the success rate for six different distances of start states from the nominal start state. The NMSE given in Subfig. 4.10b uses the optimal reference trajectory that takes the discontinuity into account (i.e., it continues optimally after the displacement at $t = 0.2$), the time grid of optimal and simulated trajectories are matched using DTW.

applied and the simulation is continued from this new state $\tilde{x}(0.2) = x(0.2) + v_{\text{pert}}$. For the computation of the NMSE, the optimal trajectory starting from $\tilde{x}(0.2)$ is computed for each perturbation in the test set using DIRCOL. The NMSE is computed on the interval $[0, 0.2]$ based on the original unperturbed optimal trajectory and on the interval $(0.2, t_f]$ based on the optimal trajectory that starts from the respective perturbed state $\tilde{x}(0.2)$. To reduce the effect of different time scales in the trajectories, the samples from the simulated and optimal trajectory are matched in time using dynamic time warping (DTW) (cf. Figure 6.7) before they are used to compute the NMSE. The results are presented in Figure 4.10. The success rate for the six different distances of start states to the nominal start state is visualized in Subfigure 4.10a for the feedforward and the feedback controller, similar to Subfigure 4.4a. Subfigure 4.10b shows the final distance to the goal state for all simulations, subdivided depending on the displacement. Subfigure 4.10c shows the NMSE and is restricted to successful simulations to achieve a practical scaling of the y-axis.

The results for the learned feedback control are consistently better than for the feedforward control, regarding both shortest distance to the goal state and overall NMSE. Obviously, a larger displacement causes a larger final distance to the goal state. The results presented in this section cannot easily be compared with the results for the perturbation in the start state (see Figure 4.4), since the perturbations in Section 4.2.1 are applied for joint states and velocities at the same time. Nevertheless, it seems that in general the effect of perturbations is more severe when they occur closer to the goal state.

In general, the effect of a displaced joint state on the performance of the learned feedback controller is much stronger than the effect of a displacement with the same magnitude on the joint velocity. The results indicate that the learned feedback control can be successfully applied in simulations with notably larger perturbations of the velocity than applied in this evaluation. The robustness of the learned feedback control seems to be limited with regard to disturbances of the joint state: For displacements of more than 0.1, the success rate of the controller on the test set decreases considerably. Further experiments have shown that if disturbances are applied at both the start state and at $t = 0.2$, the learned control is no longer successful.

To summarize, the learned near-optimal control policy is able to cope with small shifts in the joint states and moderate perturbations in the joint velocities. The results further show a clear advantage of the learned feedback controller compared to a simple optimal open-loop control.

Friction Dynamics in the Manutec r3 Robot Arm

In the following, the effect of unmodeled friction will be considered using the Manutec r3 robot arm as an example. According to Dupont [50], "friction is present to some degree in all mechanical systems", which underlines its importance for robotic systems [26, 50, 4]. To investigate how model inaccuracies in the training data affect the quality of the learned near-optimal feedback control, the dynamic model is extended with different friction models: In this evaluation, the Coulomb [26] and the Dahl [41] model are used. Both models are combined with a viscous friction [26] term, since the "majority of servo-controlled machines [...] are lubricated with oil or grease" [4].

Hollerbach et al. [163] have noted that "Coulomb and viscous friction are the most important components of a friction model". At the same time, these two models are very simple (one parameter each), their friction force is piecewise constant or depends piecewise linearly on the velocity. The used friction models are adopted to use torques instead of forces (cf. Bona and Indri [26]). According to the combined Coulomb and viscous model [134, 26], the friction torque at joint i is given by

$$F_i(q_i) = F_c \operatorname{sgn}(\dot{q}_i) + \sigma_v \dot{q}_i, \tag{4.19}$$

where F_C is the Coulomb friction and σ_v is the viscous friction coefficient.

Many dynamic models are derived from the well-known Dahl model [145], which "is basically a Coulomb friction model with a lag in the change of the friction force" [141]. Introducing an additional state to model the deflection of microscopic bristles [26], "it was designed to simulate a symmetrical hysteresis loops [sic] observed in bearings" [145]. In contrast to more complex models like LuGre [145], the Dahl model is unable to reproduce stiction or the Stribeck effect. Nevertheless, the Dahl model was chosen as a supplement to the Coulomb model for this evaluation because it depends nonlinearly on the joint velocities and is still relatively simple with only three parameters (compared to seven parameters of the LuGre model). For the Dahl model combined with a viscous friction term, the friction torque at joint i is given by

$$\begin{aligned} F_i\left(\dot{q}_i\right) &= \sigma_0 z_i + \sigma_v \dot{q}_i \qquad (4.20)\\ \dot{z}_i &= \dot{q}_i \operatorname{sgn}\left(1 - \frac{\operatorname{sgn}\left(\dot{q}_i\right)\sigma_0 z_i}{F_c}\right)\left|1 - \frac{\operatorname{sgn}\left(\dot{q}_i\right)\sigma_0 z_i}{F_c}\right|^{\alpha}, \end{aligned}$$

where $z \in \mathbb{R}^{n_q}$ are additional internal states to describe the bristle deflection, F_c is the Coulomb friction, σ_0 the viscous friction, and $1 < \alpha < 2$ is a parameter that determines the friction shape of the hysteresis loop [141, 145, 26]. As above, the constant σ_v is the viscous friction coefficient.

The friction parameters selected for this evaluation are not based on experimental data. The maximum external torque exerted by the Coulomb+Viscous and Dahl+Viscous friction models is set to be approximately $5\,\%$ of the maximum control torque. The parameters for the three joints are selected as follows:

$$\begin{aligned} F_c &= \begin{pmatrix}25 & 50 & 14\end{pmatrix} \quad [\mathrm{N\,m}]\\ \sigma_v &= \begin{pmatrix}8 & 32 & 2.7\end{pmatrix} \quad [\mathrm{N\,s}]\\ \sigma_0 &= \begin{pmatrix}800 & 1000 & 100\end{pmatrix} \quad [\mathrm{N}]\\ \alpha &= \begin{pmatrix}1.1 & 1.1 & 1.1\end{pmatrix} \quad [-]\,. \end{aligned}$$

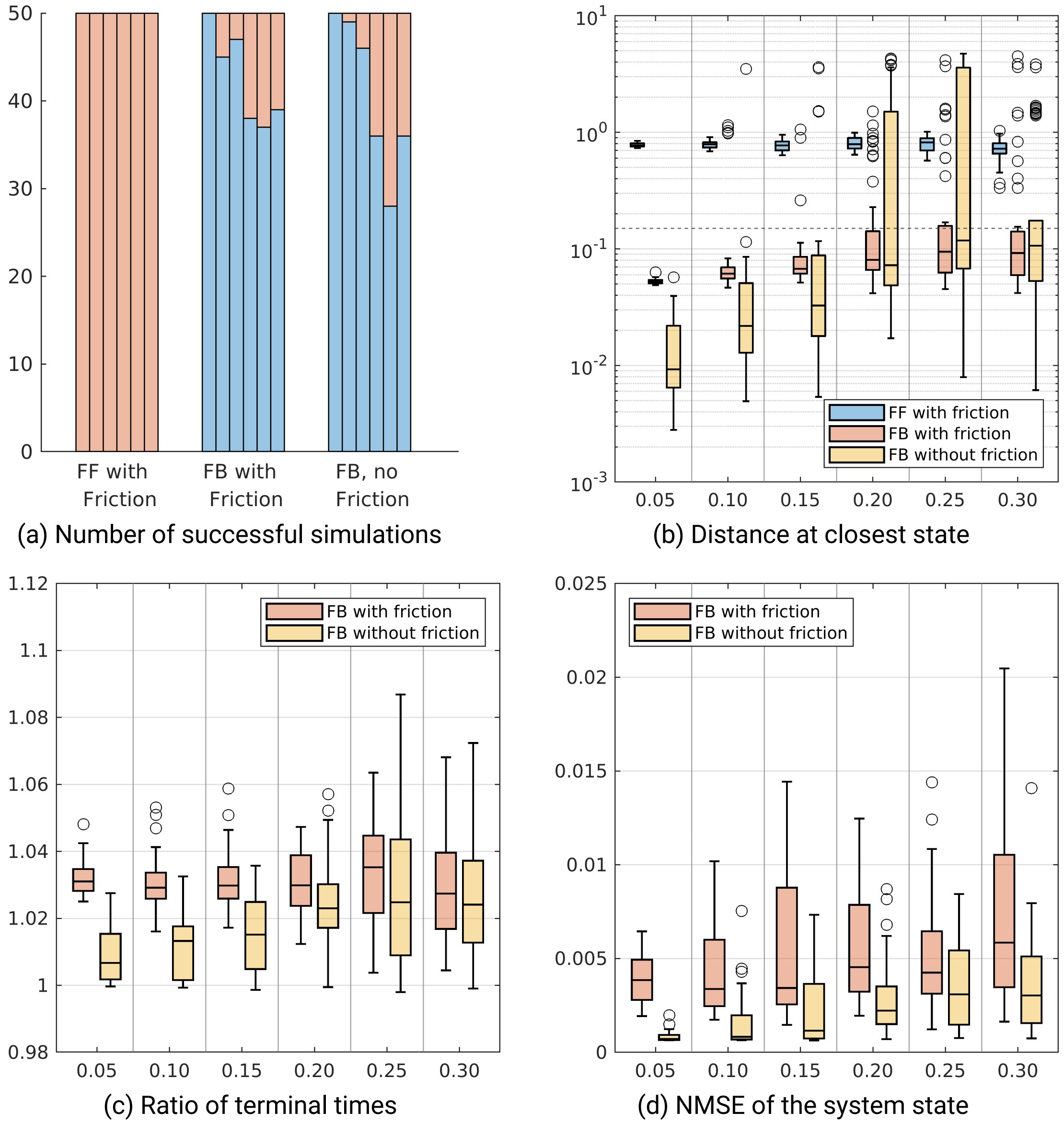
The Dahl model saturates at F_c; the maximum viscous friction torque resulting from σ_v and the maximal joint velocity (see Table 3.2) is approximately the same as the Coulomb friction torque. Together, they amount to approximately $5\,\%$ of the maximum torque output per robot joint (which is $\left(\,945\ 1890\ 540\,\right)\,\mathrm{N\,m}$, see [136]).

The evaluation done in Section 4.2.1 is repeated with the same policy (of the *full* scenario, trained on data based on a dynamic model that does not include any friction model) on exactly the same test set. In this section, the dynamic model used in the simulation is extended by the two friction models described above. The results are presented in the Figures 4.11 and 4.12. As in the foregoing paragraph, the NMSE is computed using DTW-matched simulated and optimal trajectories. Again, Figures 4.11c, 4.11d, 4.12c and 4.12d show only the values of successful simulations to achieve a practical scaling of the y-axis.

As expected, trajectories that use only the optimal feedforward control computed by DIRCOL stay far away from the goal state due to the changed dynamics and are thus all marked as unsuccessful. The open-loop case is thus excluded in the figures showing the ratio of terminal times and the NMSE. For all start state distances, the NMSE increases noticeably if the system dynamics with additional friction term is used. This shows that, regardless of the good success rate and the acceptable increase in execution time, the movement of the system is substantially altered by the changed system dynamics.

For the test instances with start states close to the nominal start state (0.05 to 0.15), the median of the shortest distance to the goal state and the median of the ratio of terminal times (see Equation (4.16)) increases slightly if the perturbed system dynamics is used in the evaluation. For the more difficult test instances (0.20 to 0.30), hardly any difference can be detected or the median is even smaller than in the evaluation without friction.

It is understandable that the effect of friction on the performance of the learned control is clearly recognizable for trajectories that are closer to the nominal trajectory, since here the learned control is trained better and provides decent performance for the unperturbed system dynamics. However, it is remarkable that the learned control still works relatively well for more distant trajectories and brings the system state close to the goal state in the majority of cases. Interestingly, it can be seen that the success rate even improves if the Dahl friction model is used in the evaluation. While the median distance from the goal state is slightly increased in evaluations with friction (as noticed above), it stays below the threshold of 0.15, below which test instances are considered successful. Furthermore, the number of outliers decreases. The increased success rate for the evaluations with friction can be explained by the fact that friction removes kinetic energy from the system, resulting in fewer violations of the state constraints during the dynamic movement of the robot arm.



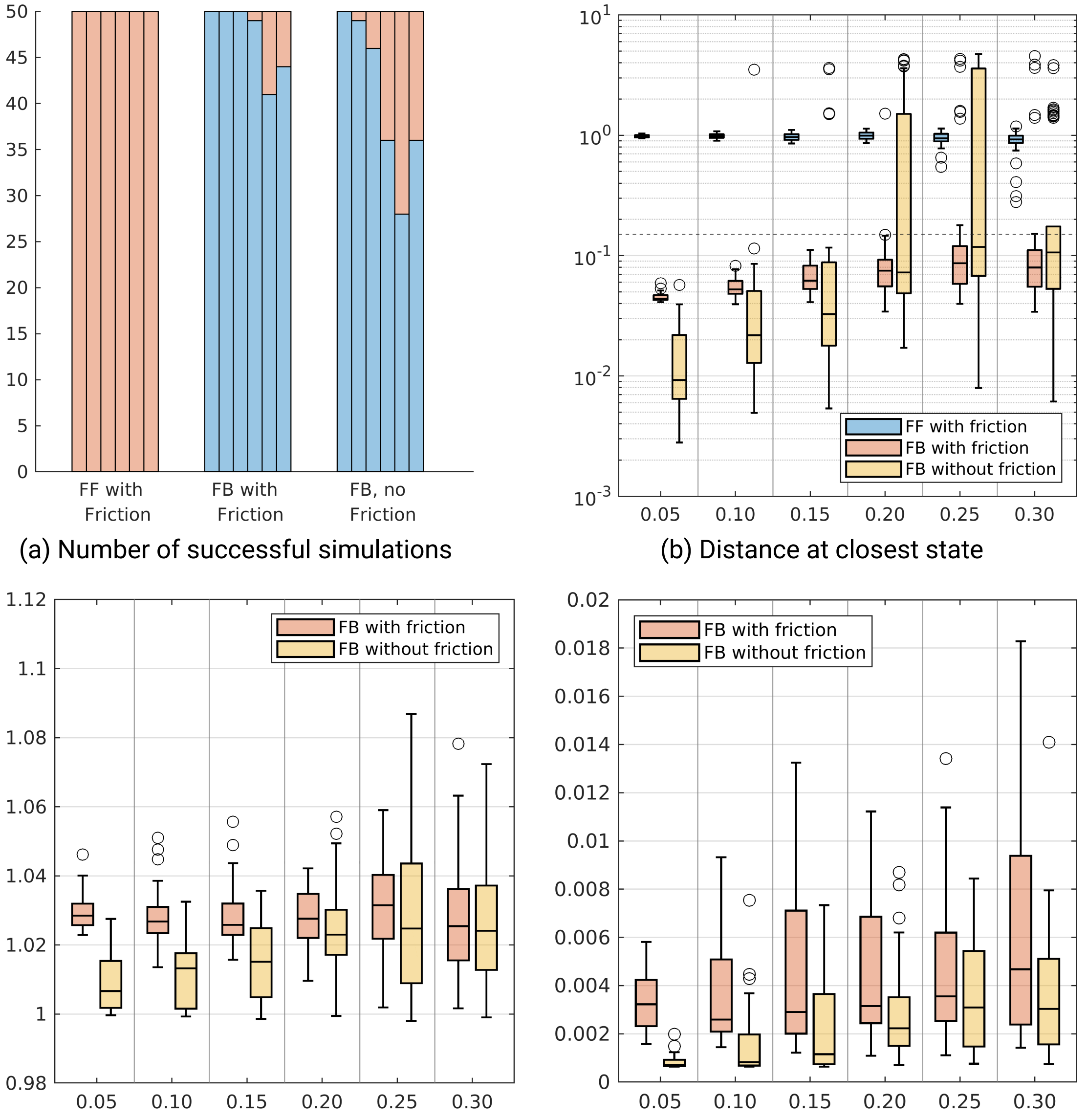
(a) Number of successful simulations

(b) Distance at closest state

(c) Ratio of terminal times

(d) NMSE of the system state

Figure 4.11.: Evaluation of the near-optimal feedback policy (FB) using the Coulomb+Viscous friction model. Subfig. 4.11a visualizes the success rate for six different distances of start states from the nominal start state. Subfigs. 4.11b to 4.11d present the results for the performance criteria (4.15) to (4.17). Subfigs. 4.11c and 4.11d show only the values for successul simulations. The boxes encompass the interquartile range (IQR), the maximal whisker length is $1.5\,\mathrm{IQR}$, outliers are marked with $\circ$.



(a) Number of successful simulations

(b) Distance at closest state

(c) Ratio of terminal times

(d) NMSE of the system state

Figure 4.12.: Evaluation of the near-optimal feedback policy (FB) using the Dahl+Viscous friction model. Subfig. 4.12a visualizes the success rate for six different distances of start states from the nominal start state. Subfigs. 4.12b to 4.12d present the results for the performance criteria (4.15) to (4.17). Subfigs. 4.12c and 4.12d show only the values for successul simulations. The boxes encompass the interquartile range (IQR), the maximal whisker length is $1.5\,\mathrm{IQR}$, outliers are marked with $\circ$.

These results indicate that perturbations are not necessarily a hindrance, but some of them may even be beneficial to reaching the goal. This can be well illustrated by the example of an airplane that has to reach a destination under the influence of unknown winds. In this example, tailwinds allow a faster and more fuel-efficient flight than the precomputed optimal solution. An optimal feedback control policy, as pursued in this thesis, is able to take advantage of these perturbations: It would not artificially slow down the airplane to stay on the precomputed trajectory but would continue from the advantageously perturbed state towards the destination.

It can be concluded that the addition of a friction model to the system dynamics is clearly noticeable in the error of the resulting trajectory. However, the number of cases in which the goal state cannot be successfully reached due to the friction term is low. Decent results regarding terminal time and proximity to the goal state can be achieved even in case of perturbed system dynamics. The results indicate that a feedback control policy can provide satisfying results on a real system even if it is trained using a dynamics model with a small error in the friction model. The problem of modeling errors when learning the feedback control policy revisited in Section 5.4, in which experiments on a real system are also conducted.

4.3. Discussion and Conclusion

By applying the changes described in this chapter, the performance of the iterative extremal field approach outlined in Chapter 3 can be substantially improved. In this chapter, three new strategies (sensitivity-based, Halton-based and simulation-based) are proposed to identify start states for new trajectories. These complementing approaches support the adjoint-based approach from the previous chapter to guide the iterative computation of new optimal trajectories to ensure adequate state space coverage and provide meaningful data that is used to train a near-optimal control policy. The evaluation demonstrates that the four start state generation strategies, particularly the adjoint-based and Halton-based methods, are valuable strategies for improving the extremal field approach. The results of a direct comparison with a single start state selection strategy (the adjoint-based method from Chapter 3) show clear advantages of the approach using several complementing selection strategies: Most of the trajectories controlled with a control policy trained with the combined strategy get closer to the goal state and have a lower average cost. This underlines the importance of a sensible selection of start states for new trajectories and justifies the effort in this subject. As noted before, finding appropriate parameters for the start state selection strategies is crucial for a good performance. Unfortunately, this requires a substantial amount of trial and error. It would be desirable to reformulate the strategies with fewer parameters or find some auto-tuning approach that handles this work.

Further, a more sophisticated method is used to filter linearly dependent data points that produce noise in the kernel matrix of the GP. Both improvements, the new complementing start state selection strategies and the improved filtering of redundant data points, allow the generation of more meaningful training data, which enables a more precise approximation of the near-optimal policy.

Moreover, a switch-over in the proximity to the goal state from the learned near-optimal control to a stabilizing LQR controller has been proposed. It allows the system to reach the goal state precisely and makes deliberate data collection around the start state, as done in Chapter 3, unnecessary. However, this negatively impacts optimality, as outlined in Section 4.2.2, and therefore should only be used in a small ball around the goal state. Section 5.5 presents the result of a near-optimal feedback controller trained with all four start state selection methods used on a real system, including a switch-over close to the goal state.

The presented work demonstrates that it is not necessary to reach the exact goal state with the learned control. Instead, it suffices to reach a ball around the goal state, from which control is passed to a traditional PI controller that guides towards and stabilizes around the goal state. An extension of the switch-over approach that needs to be further evaluated would be to consider the switch-over during the trajectory optimization: The condition at the final time can be relaxed since it is no longer necessary to reach the exact goal state, but it is sufficient to reach the region around it from which the PI controller can take over.

The evaluation in this chapter considers not only perturbations at the start states but also abrupt changes in the state or velocity that occur during the execution of the trajectory and uses system dynamics perturbed by different friction models. The motivation to use near-optimal feedback control policies is their inherent robustness to perturbations. The learned controller of the Manutec r3 arm has shown decent performance when subjected to abrupt changes in the state or velocity during the execution of the control policy (intermittent disturbances) and perturbed system dynamics using different friction models (continuous nonlinear disturbances). This indicates that the presented approach may benefit applications where significant perturbations occur (see, e.g., [34, 190, 83]).

This chapter is based on the paper "Start State Selection for Control Policy Learning from Optimal Trajectories", published in 2023 IEEE International Conference on Robotics and Automation (ICRA) *[214]. It contains the following major additions: visualizations of the selection methods in Figure 4.1, 4.2 and 4.3, extended section 4.1.3, added Table 4.2 a largely extended Section 4.2.1 with experiments considering constraint violations and a comparison with LHS sampling, extended evaluation in Section 4.2.2, Section 4.2.3 on the effect of perturbed dynamics and external perturbations.*

5. Comparison of Approximate Policy Representations

The iterative extremal field approach presented in the previous chapters relies on some representation of the learned near-optimal feedback controller that is learned from the data provided by the optimal control solver. This representation of the feedback control policy approximation must be sufficiently general to reproduce the often highly nonlinear features of control policies. Furthermore, it must generalize well enough to deal with the non-uniform distribution of the data in the input space, which results from the fact that the training data is sampled from trajectories. At the same time, parameter tuning, i.e., training of the approximations using available data, must be efficient, and the evaluation of the trained feedback control approximation should be fast to allow an application in real-time. Hence, the approximation of feedback control policies from trajectory data has specific requirements on the method used, but a systematic comparison of common approximation approaches that takes these requirements into account has, to the best of the author's knowledge, not been carried out so far. In this chapter, GP and NN, two widely used function approximators, are compared in terms of their suitability for the approximation of feedback controls. Suitable choices of GP kernels, NN topologies and other hyperparameters are evaluated and discussed. The evaluation is performed on a Manutec r3 robot arm in simulation and on a real-world Furuta pendulum.

For reinforcement learning, NNs are mostly used, as they are general function approximators and can be trained for large data sets. NNs are ubiquitous in data science and machine learning. They are universal function approximators but are also known to require substantial amounts of training data and lack the ability to inherently provide information about prediction uncertainty. GPs [154] are used as function approximators in many applications as well, in particular if information about the prediction uncertainty is required. GPs are well suited for small amounts of training data and the standard approach does not scale well for large amounts of data. This is why a plethora of approximations exist [108] to make the approach suitable for large amounts of data.

Evidently, GPs and NNs have specific strengths and weaknesses. However, methods have been developed to reduce or even eliminate the most serious shortcomings of these approaches. Furthermore, the selected hyperparameters and training options significantly influence the achievable accuracy of the feedback control approximation. The choice of the approximation method for the feedback controller in Algorithm 1 (cf. Section 3.2) and appropriate hyperparameters is important and must be considered carefully, as this affects its suitability for real-time applications and the achievable approximation accuracy of the overall iterative approach.

Training data sampled from the optimal trajectories depends on the system model used in the OCP. This implies that the iterative extremal field approach is, to some extent, susceptible to inevitable model errors. To assess the ability of the approach examined in this thesis, and in particular the various approximation methods considered here, to compensate for model errors, the learned controllers are applied to the real-world pendulum with perturbed dynamics. The Furuta pendulum dynamics are perturbed by placing weights (in the form of coins) at different positions on the pendulum.

Requirements on Approximation Methods for Feedback Control Policies

The following requirements to specify the suitability of an approach to approximate feedback control policies are used. They are motivated subsequently.

R1) The approach should be able to generalize to unseen data and provide a high prediction accuracy, measured by computing the NMSE on a test set not used for training. The accuracy depends on the amount of training data and its distribution in space. One cannot expect the approximation accuracy on a small training set to be the same as for large amounts of training data. Nevertheless, the accuracy should be reasonably high when the policy is trained on small training sets.

R2) It must be possible to evaluate the learned policy fast enough to allow execution in real-time. Predictions must take at most $2\,\text{ms}$ to allow control of the system with $500\,\text{Hz}$. This includes the scalability of the approximation approach with respect to the input and output dimensions and the amount of training data.

R3) The trained approximation needs to provide information about the epistemic prediction uncertainty if required.

R4) The runtime performance of the training routine is of minor importance. Nevertheless, efficient training routines are desirable. At best, efficient retraining or online updates are possible.

Accuracy and computational performance are standard requirements for comparing different approximation methods. The accuracy is expected to increase with the amount of training data. Since experimental data to train a control policy may be expensive to collect, the accuracy is examined also for small amounts of training data. To allow the application of the learned feedback controller on dynamic robotic systems, the evaluation (or prediction) of the control policy is required to be sufficiently fast. While it may be desirable that control policies can be updated with data collected online (cf. R4)), it is assumed in this chapter that all training data are collected in advance and the training is conducted offline, such that online updates are not further considered. The effort to train control policies should be reasonable. Information about the current model uncertainty, covered by R3), is beneficial in general and can be used in exploration strategies to guide collection of new data [213]. Iterative approaches to approximate a near-optimal feedback controller from optimal trajectories, as proposed by [64, 213], have the advantage that further progression can be guided by the data collected so far. In particular, uncertainty information as required in R3) can be used in exploration strategies to guide collection of new data [213]. This justifies the emphasis on uncertainty estimation for the approximation methods.

5.1. Related Work on Approximate Policy Representations

In their work, Ommer et al. [135] include a comparison of sparse online GPs, LWPR and recursive least squares for approximation of a feedforward controller. Furthermore, they compare the performance of the approximation methods on a real robot. However, their main focus is real-time applicability with online updates of the training data, which is of minor importance in this work. They do not consider uncertainty estimation or different kernel functions for the GPs. The approximation of inverse dynamics is considered in Nguyen-Tuong et al. [130]; they compare the performance of their proposed local GP with various approaches like standard GP or LWPR. In their analysis, they restrict themselves to Gaussian kernels for the GPs. Vasudevan et al. [194] investigate the suitability of GPs for modeling large-scale terrain. While the application is different, they also try to approximate a highly nonlinear nonstationary function over a large input domain and emphasize the importance of uncertainty estimation. They analyze in detail how to apply GPs to their problem, propose the use of local models to cope with a large amount of training data and compare different kernels for GPs. However, there is no comparison with NNs.

Uncertainty can be divided into two categories: aleatoric and epistemic uncertainty [193, 88]. Aleatoric uncertainty (data uncertainty) is introduced by noise in the observations, caused by the technical limitations of the sensors used. It cannot be reduced by collecting more data. In contrast, epistemic uncertainty, or model uncertainty, describes the errors introduced by the modeler's limited knowledge about the inspected physical system. Further, this category also contains uncertainty caused by missing or sparse data points, e.g., in unexplored regions of the state space. Epistemic uncertainty can be reduced by collecting more data to improve the model knowledge. Various methods exist to capture model uncertainty in NNs [1, 166]. In Bayesian NNs, a probability distribution over all parameter weights is used to provide epistemic uncertainty information for the network output. However, these networks are hard to implement and training is substantially slower and more difficult than training deterministic networks [97, 62]. An alternative are deep ensembles to estimate uncertainties as proposed by Lakshminarayanan et al. [97]. In this approach, a small set of networks with the same topology but different start values of the weight parameters are trained independently on the same data. For a given input, the prediction of the ensemble is the mean and variance over the values predicted by all networks. Another popular approach, that is used in this chapter, is Monte-Carlo (MC) dropout [62]. Dropout-layers are inserted before the hidden layers to randomly disable some nodes in them. Dropout is a standard regularization method for training of NNs, the dropout layers are normally only used during training. According to Kendall and Gal [88], it is a "practical approach" to get uncertainty information from NNs compared to Bayesian NNs (parameter weights are distributions) in which it is "difficult to perform inference". If the dropout layers are enabled during prediction, the output of the NN becomes stochastic. Gal and Ghahramani [62] showed that the distribution of the results of multiple forward passes can be interpreted as variational Bayesian approximation that captures the epistemic uncertainty of the network [88]. Wu et al. [206] compared MC dropout with ensemble learning and other uncertainty quantification methods and conclude that MC dropout has less computational overhead and needs fewer training episodes. Valdenegro-Toro et al. [193] note that the ensemble method provides slightly better results than MC dropout, but also comes with a larger computational overhead since all networks in the ensemble must be trained separately. In contrast to MC sampling, Lakshminarayanan et al. provide no formal proof that the ensemble-based approach converges to the Bayesian uncertainty approximation [206].

There are many other important function approximation methods, for example, LWPR by Vijayakumar and Schaal [198, 197] has been used frequently for robotic applications [135, 130]. It uses multiple locally linear models to fit nonlinear functions, provides an uncertainty estimation for predictions, and is suitable for online applications due to its efficient updates on new training data. However, it is known to require extensive tuning. Tests done for this work have indicated that the achievable accuracy is lower than that of GPs and NNs. While this may be due to insufficient parameter tuning, other works ([203, 135]) confirm this observation. A systematic comparison of adequately tuned LWPR with the methods examined in this chapter may provide interesting insights and is subject to future work.

Another important function approximation method is kriging, which is very popular in the field of geospatial statistics [35]. Christianson et al. compare Gaussian process regression and kringing and note that these techniques are conceptially very similar, as they use the same equations to "form predictions and quantify uncertainty". However, an important difference is that GPs automate the parameter tuning by maximizing the likelihood while kriging requires human intervention.

5.2. Comparative Study Approach

The scope of the study performed here is set out in this chapter. This includes the selection of approximation methods and the hyperparameters and extensions considered. Further, the construction of the data sets that are used in the evaluation is described. Finally, the steps that constitute the analysis are set out.

5.2.1. Selection of Function Approximators

The literature on GPs and NNs contains a huge amount of options for construction and training, approaches for tuning, and extensions designed for various purposes that cannot be fully covered in this study. In the following, the design decisions made in this analysis and the options considered for comparison are briefly justified.

Neural Networks The decision on a network topology (structure in which its neurons are connected) is essential and must be made problem-specific. The number of existing network topologies is large, but most of them are designed for specific purposes. For example, auto-encoders are designed for unsupervised learning and can be used among others for denoising and dimensionality-reduction [71]. Convolutional neural networks (CNNs) are "specialized [...] for processing data that has a known, grid-like topology" [71] and are thus typically applied to image recognition and other computer vision tasks to extract features. Moreover, they can also be applied to time-series data by interpreting it as a 1D grid. However, this still does not fit the problem considered in this thesis since the time information is removed from the data sampled from the optimal trajectories and used to approximate a static map from the system state to the feedback control that is independent of time. Thus, only fully connected feedforward networks are considered in the subsequent analysis. For simplicity, all hidden layers have the same number of nodes, such that the network topology can be described by two numbers: the number of hidden layers and nodes per layer. The activation functions for the hidden layers are rectified linear activations (ReLu), which are commonly used and recommended in [71]. The input is scaled to be between -1 and 1 using the state constraints in the problem formulation. To get a stochastic model, a dropout layer is added before each hidden layer. The dropout rate is $5\,\%$, which seems to be sufficient for uncertainty estimations. Higher values make the training very difficult as the progress becomes very erratic.

Gaussian processes The kernel function selected for a GP has a significant influence on its performance. In [213], the MLP kernel has been identified to provide the best results when approximating a highly nonlinear feedback control policy. Additionally, the performance provided by different kernels is evaluated and compared in this chapter. The kernels used are the MLP, Gibbs and Matérn-$3/2$ kernel as well as the compound kernels Sqexp and ARD that are defined in the *GPmat* toolbox [3]. The definition of these kernel functions can be found in Appendix A.1. The size of the kernel matrix that needs to be inverted during training depends cubically on the number of training samples. To be able to cope with large amounts of training data, several scalable GPs have been developed [108]. In this chapter, the FITC approximation [167] with 100 inducing variables is used. The abbreviation FTC follows the notation of the *GPmat* toolbox and refers to inference done on the full training set (without any approximation using inducing variables).

5.2.2. The Steps for Evaluation and Comparison

To compare GPs and NNs, their performance is evaluated in two main steps. The focus of the first step E1 is to evaluate the achievable prediction accuracy depending on the topology/kernel, other hyperparameters and the number of samples used during training (see R1)). Step E1 is divided into four sub-steps. The goal of step E1.1 is to find suitable hyperparameters and training options for large amounts of training data. For NNs, the systematic tests comprise four to ten hidden layers with 50 to 200 nodes each and 0% and 5% dropout rate. For GPs, the kernels MLP, Gibbs, Matérn-$3/2$, Sqexp and ARD are

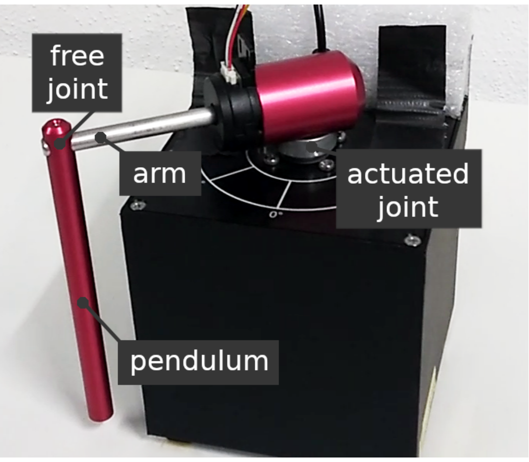


Figure 5.1.: Annotated Quanser Furuta pendulum

considered in step E1.1. The FITC approximation with 100 inducing points is used to deal with the large number of training samples. The hyperparameters and options that provide the best results in E1.1 are used in step E1.2 to train feedback control approximations on training data sets with different numbers of training samples. This allows to assess the extent to which the amount of training data affects the prediction accuracy. Steps E1.3 and E1.4 repeat the steps E1.1 and E1.2 for small amounts of training data. E1.3 again determines suitable parameters using approximately 1000 training samples by systematically varying the parameters and selecting the configuration that produces the lowest error. In E1.4, the parameters determined in step E1.3 are used to evaluate the accuracy on a test set with 500 to 5000 training samples. Step E1 considers large and small training sets separately such that hyperparameters can be found that are specifically suitable for large or small training sets. The second evaluation step E2 determines the times to evaluate the learned control policies for some given input. In case of NNs, this is done for different network topologies. For GPs, this is done for different amounts of training data.

5.2.3. The Dynamic Models

The Furuta pendulum (see Figure 5.1) is an underactuated system (only the first of two joints can be controlled directly) with four-dimensional state and one-dimensional control space. A detailed description of the motion dynamics and the definition of the OCP used

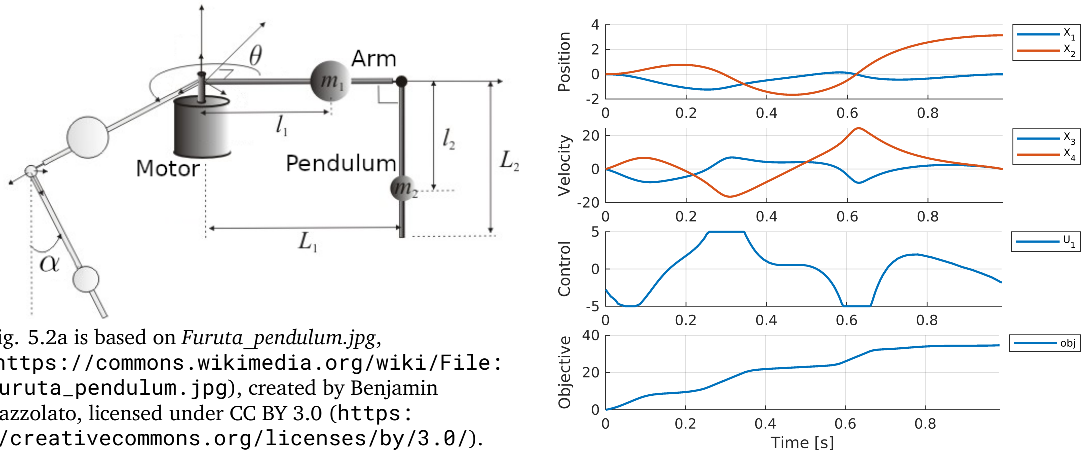


Fig. 5.2a is based on *Furuta_pendulum.jpg*, (`https://commons.wikimedia.org/wiki/File:Furuta_pendulum.jpg`), created by Benjamin Cazzolato, licensed under CC BY 3.0 (`https://creativecommons.org/licenses/by/3.0/`).

(a) Schematic of the Furuta pendulum.

(b) Nominal trajectory of the Furuta pendulum swing-up movement.

Figure 5.2.: Schematic and nominal joint trajectory of the Furuta pendulum.

in this thesis is given in the next subsection. The objective of the OCP is to bring the pendulum into an upright position, which is a very instable state. The Manutec r3 arm is an industrial robot with highly nonlinear dynamics [136]. As in the previous chapters, the first three joints that determine the end-effector position in task space are used. Hence, the control policy is a mapping from the six-dimensional state to the three-dimensional control space. See Section 3.3.2 for a complete description of the OCP to generate point-to-point movements.

5.2.4. The Furuta Pendulum Optimal Control Problem

The Furuta pendulum [61] is an underactuated system with four-dimensional state and one-dimensional control space. The motion dynamics are based on the model described in [32]. The model presented here corrects some terms in this model and accounts for systems that are set up with a slight angle. For the angle θ of the arm and the pendulum angle α, the dynamic model of the Furuta pendulum solves the differential equation

$$M\left(\theta,\alpha\right)\begin{pmatrix}\ddot{\theta}\\ \ddot{\alpha}\end{pmatrix}+C\left(\theta,\alpha,\dot{\theta},\dot{\alpha}\right)+G\left(\theta,\alpha\right)g-F\left(\dot{\theta},\dot{\alpha}\right)+u=0 \tag{5.1}$$

with mass matrix $M \in \mathbb{R}^{2\times 2}$, coriolis terms $C \in \mathbb{R}^2$ gravity matrix $G \in \mathbb{R}^{2\times 3}$, vector of damping forces $F \in \mathbb{R}^2$ and control terms $u = \left(\begin{smallmatrix} 0 & \tau \end{smallmatrix} \right)^T$ defined as

$$\begin{aligned}
M_{11} &= I^1_{yy} + I^2_{yy} + L_1^2 m_2 + l_1^2 m_1 + l_2^2 m_2 + \left(I^2_{xx} - I^2_{yy}\right)\cos^2(\alpha) - l_2^2 m_2 \cos^2(\alpha)\\
M_{12} &= M_{21} = L_1 l_2 m_2 \cos(\alpha)\\
M_{22} &= I^2_{zz} + l_2^2 m_2\\
C_1 &= I^2_{yy}\dot{\alpha}\dot{\theta}\sin(2\alpha) - I^2_{xx}\dot{\alpha}\dot{\theta}\sin(2\alpha) - L_1\dot{\alpha}^2 l_2 m_2 \sin(\alpha) + \dot{\alpha}\dot{\theta} l_2^2 m_2 \sin(2\alpha)\\
C_2 &= -\frac{1}{2}\left(\dot{\theta}^2 \sin(2\alpha)\left(I^2_{yy} - I^2_{xx} + l_2^2 m_2\right)\right)\\
G_{11} &= -L_1 m_2 \sin(\theta) - l_1 m_1 \sin(\theta) - l_2 m_2 \sin(\alpha)\cos(\theta)\\
G_{12} &= l_1 m_1 \cos(\theta) + L_1 m_2 \cos(\theta) - l_2 m_2 \sin(\alpha)\sin(\theta)\\
G_{13} &= 0\\
G_{21} &= -l_2 m_2 \cos(\alpha)\sin(\theta)\\
G_{22} &= l_2 m_2 \cos(\alpha)\cos(\theta)\\
G_{23} &= l_2 m_2 \sin(\alpha)\\
g_1 &= g_{\|\cdot\|}\sin\left(g_{\mathrm{Y0}}\right)\cos\left(g_{\mathrm{Z0}}\right)\\
g_2 &= g_{\|\cdot\|}\sin\left(g_{\mathrm{Y0}}\right)\sin\left(g_{\mathrm{Z0}}\right)\\
g_3 &= g_{\|\cdot\|}\cos\left(g_{\mathrm{Y0}}\right)\\
F_1 &= d_1\dot{\theta}\\
F_2 &= d_2\dot{\alpha}.
\end{aligned} \tag{5.2}$$

The motor model that maps motor voltage a to the torque τ is adapted from [124]:

$$\tau = \frac{k_m(a - k_m\dot{\theta})}{R_m}. \tag{5.3}$$

The objective of the OCP considered in this thesis is to bring the pendulum from $x_0 = \left(\begin{smallmatrix} 0 & 0 & 0 & 0 \end{smallmatrix} \right)$ (hanging) into an upright position $x_f = \left(\begin{smallmatrix} 0 & \pi & 0 & 0 \end{smallmatrix} \right)$. To remove symmetry, the goal state is exactly x_f neglecting all rotationally equivalent positions. The functional to be minimized penalizes long execution times, large control values and deviations from the goal state of the second joint:

$$\begin{aligned}
J[u] &= c_T \cdot t_f + \int_0^{t_f} L(x(t), u(t))\,\mathrm{dt}\\
L(x, u) &= \Lambda(x_1) + c_u u_0^2\\
\Lambda(x_1) &= -\exp\left(1 - \gamma\cos\left(x_1\right)\right) + \exp\left(\gamma + 1\right).
\end{aligned} \tag{5.4}$$

Parameter	Value	Explanation
γ	1.6	Penalty factor in Λ
c_T	10.0	Factor on terminal time penalty
c_u	3.0	Factor on control penalty
$g_{\mathrm{Y}0}$	$-1.293\,44 \times 10^{-4}$	System angle around y-axis
$g_{\mathrm{Z}0}$	0.0	System angle around z-axis
$g_{\|\cdot\|}$	9.81	Length of the gravity vector
I^1_{yy}	$2.077\,19 \times 10^{-6}$	First diagonal entry in the arm's inertia tensor
I^2_{xx}	$1.153\,53 \times 10^{-7}$	First diagonal entry in the pendulum's inertia tensor
I^2_{yy}	$1.603\,68 \times 10^{-7}$	Second diagonal entry in the pendulum's inertia tensor
I^2_{zz}	$2.704\,32 \times 10^{-5}$	Third diagonal entry in the pendulum's inertia tensor
L_1	0.085	Arm length
l_1	0.0425	Center of mass on arm
l_2	0.0645	Center of mass on pendulum
m_1	0.095	Arm mass
m_2	0.024	Pendulum mass
k_m	$4.228\,57 \times 10^{-2}$	Motor model: Back-emf constant
R_m	8.4	Motor model: Armature resistance
d_1	$3.446\,93 \times 10^{-4}$	Damping in the first joint
d_2	$9.147\,15 \times 10^{-6}$	Damping in the second joint

Table 5.1.: Parameters that determine the Furuta pendulum OCP.

The parameters are $c_T = 10.0$, $\gamma = 1.6$ and $c_u = 3.0$. The nonlinear mapping Λ has a maximum at hanging positions $0 + 2k\pi$, $k \in \mathbb{N}$ and attains its minimum at $\pi + 2k\pi$. This heavily penalizes values around a hanging position of the pendulum arm, and barely penalizes values near the goal state. The parameters used in the Equations (5.1) to (5.4) that determine the OCP are given in Table 5.1. The state and control variables are constrained by the box constraints $x_{\mathrm{min}} \leq x \leq x_{\mathrm{max}}$ and $u_{\mathrm{min}} \leq u \leq u_{\mathrm{max}}$, the values are given in Table 5.2.

5.2.5. Construction of the Data

The data used for learning and evaluation of the control policy approximations is generated from solutions of OCPs. This approach has the advantage that no experiments are required

	$x[0]$	$x[1]$	$x[2]$	$x[3]$
x_{min}	-2.0071	-2π	-60.0	-60.0
x_{max}	2.0071	2π	60.0	60.0

(a) State Constraints

	$u[0]$
u_{min}	-5.0
u_{max}	5.0

(b) Control Constraints

Table 5.2.: Upper and lower bounds on the state and control variables of the Furuta pendulum.

and large amounts of training data can be collected using a dynamic model of a system and a numerical optimal control solver. As basis of the test set for the Furuta pendulum, 5052 solutions of the OCP are computed, starting from start states sampled randomly from a uniform distribution over $[-0.25, 0.25] \times [-0.25, 0.25] \times [-4, 4] \times [-4, 4]$, which is a subset of the joint space. Samples for which DIRCOL was unable to provide an optimal solution have been skipped. The 5052 solutions are divided into a training set and a test set. From each solution in the training and the test set, state-control pairs are sampled. It has been ensured that no state value from the training set appears in the test set by filtering samples that fall below some distance threshold. From the training set, a validation set is separated; it is again ensured that all state values in the validation set have a positive distance from all state values in the remaining training set. The full training set consists of $446\,680$, the validation set of $78\,780$, and the test set of $131\,000$ samples.

For the second test set based on the Manutec r3 robot arm, 2617 solutions are computed; their start states are sampled uniformly over the full joint space (the state bounds are given in Table 3.2a). The distribution into training, validation and test set is done as for the Furuta pendulum problem. The training set consists of $231\,400$, the validation set of $40\,820$, and the test set of $67\,990$ samples.

The training set is characterized by the fact that its samples are not evenly distributed in space but are located on trajectories. This is also taken into account in the creation of the training sets with reduced size used in E1.2 to E1.4: They are not just random subsets of the full training set but are sampled from a random subset of the computed solution trajectories. The same evaluation and test set is used in all steps E1.1 to E1.4, regardless of the size of the training set.

		dropout	batch size	learn rate	learn rate factor
Furuta	E1.1/1.2	0 %	512	1×10^{-3}	0.98
		5 %	512	1×10^{-4}	0.97
	E1.3/1.4	0 %	128	1×10^{-3}	0.98
		5 %	128	1×10^{-3}	0.97
Manutec r3	E1.1/1.2	0 %	512	1×10^{-2}	0.99
		5 %	512	1×10^{-4}	0.98
	E1.3/1.4	0 %	128	1×10^{-3}	0.95
		5 %	128	1×10^{-3}	0.95

Table 5.3.: Neural network training hyperparameters

5.3. Performance Evaluation and Comparison

Training of the GPs is done in Matlab using the GPmat scaled conjugate gradient optimization method or the old conjugate gradients routine if the former failed. The neural networks are trained in Python with *keras* using RMSprop with mean squared error as loss function; the learning rate is reduced after each epoch. The validation set is used in the stopping criterion: The training stops when the NMSE on the validation set has not improved over 20 consecutive iterations. The hyperparamers for training are summarized in table 5.3. The initial learn rate and the learn rate schedule are manually tuned to provide the best results in E1.1 and E1.3 and kept fixed in E1.2 and E1.4. The NMSE (mean squared error divided by the variance of the approximated control values, see Equation (4.17)) on the test set is used as a measure of the accuracy.

5.3.1. Accuracy of the Control Approximations

The NMSE of GPs trained with different kernels on the large Furuta pendulum and Manutec r3 dataset are reported in Table 5.4a. The Sqexp kernel performs best for both problems, closely followed by the ARD and MLP kernels. The Gibbs and Matérn-3/2kernel fail to approximate the Manutec data set. As a side note, the error for different numbers of inducing points with FITC differs not much, but a large number increases the computational burden for training and evaluation significantly. It must be noted that it takes several hours to train the GPs. Since training takes a long time, step E1.2 is not carried out.

	ARD	Gibbs	M-3/2	MLP	Sqexp
Furuta pendulum					
FTC	–	–	–	–	–
FITC100	0.065	0.050	0.056	0.074	**0.049**
Manutec r3 arm					
FTC	–	–	–	–	–
FITC100	0.403	22.62	2.406	0.407	**0.399**

(a) E1.1: NMSE of GPs for the Furuta pendulum and Manutec arm trained with full 447 k (resp. 231 k) samples.

	ARD	Gibbs	M-3/2	MLP	Sqexp
Furuta pendulum					
FTC	0.181	0.209	0.209	**0.108**	0.787
FITC100	0.122	**0.100**	NaN	0.114	0.118
Manutec r3 arm					
FTC	**0.953**	3.975	3.068	1.219	1.758
FITC100	0.889	17.07	0.909	**0.855**	0.871

(b) E1.3: NMSE of GPs for the Furuta pendulum and Manutec arm trained with approximately 1000 samples.

Table 5.4.: E1.1 and E1.3: NMSE of GPs for Furuta pendulum and Manutec r3 arm. Result of the systematic search for suitable kernels.

Furuta pendulum					Manutec r3 arm				
440 k	270 k	100 k	62 k	24 k	231 k	169 k	100 k	62 k	24 k
0.034	0.035	0.035	0.041	0.055	0.236	0.275	0.331	0.459	1.056

Table 5.5.: E1.2: NMSE of NNs for Furuta pendulum and Manutec r3 arm trained on large subsets of the training set.

The examination of the GPs continues with the hyperparameter tuning for the small-size datasets, for which both standard and sparse GPs are considered.

The results using NNs are reported in Tables 5.6a and 5.6b. The error tends to decrease with increasing number of nodes. In contrast, it can be seen for both systems that the number of layers barely impacts the error. Adding dropout layers during training notably reduces the accuracy of the approximation, and training progression becomes more fluctuating. They are, however, necessary to allow an estimation of the uncertainty as described in Section 5.1. It is advisable to keep the dropout rate low. Generally, the accuracy for both systems achieved with NNs in E1.1 is significantly higher than that obtained using GPs.

Step E1.2 for NNs is done using networks with six layers of 200 nodes for the Furuta pendulum and ten layers of 200 nodes for the Manutec arm problem, both with 5% dropout. Incrementally reducing the size of the training set of the NNs leads, as expected, to increased errors, see Table 5.5, most notably for the more complex Manutec arm problem. The error on the data set with 24 k samples exceeds the error for the better

	no dropout				5 % dropout			
	50	100	150	200	50	100	150	200
4	0.010	0.009	0.008	**0.008**	0.043	0.036	0.036	0.035
6	0.009	**0.009**	0.009	0.010	0.039	0.034	0.033	**0.031**
8	0.009	0.011	0.010	0.010	0.044	0.034	0.036	0.032
10	**0.012**	0.01	0.010	0.010	0.046	0.044	0.039	0.036

(a) E1.1: NMSE of NNs for Furuta pendulum trained with 447k samples.

	no dropout				5 % dropout			
	50	100	150	200	50	100	150	200
4	0.114	0.120	0.115	0.121	0.563	0.409	0.332	0.318
6	0.111	0.113	0.112	0.118	0.510	0.315	0.282	0.247
8	0.116	0.118	0.109	0.112	0.494	0.307	0.255	0.241
10	0.115	0.116	0.108	**0.102**	0.506	0.304	0.256	**0.240**

(b) E1.1: NMSE of NNs for Manutec r3 arm trained with 231k samples.

	no dropout				5 % dropout			
	50	100	150	200	50	100	150	200
4	0.090	0.092	0.079	0.086	0.129	0.109	0.110	0.097
6	0.098	0.081	0.079	0.094	0.111	0.103	**0.090**	0.098
8	0.091	0.075	0.081	0.096	0.107	0.106	0.099	0.096
10	0.108	0.076	**0.074**	0.142	0.120	0.097	0.101	0.097

(c) E1.3: NMSE of NNs for Furuta pendulum trained with ~ 1000 samples.

	no dropout				5 % dropout			
	50	100	150	200	50	100	150	200
4	2.414	2.625	0.901	1.744	1.775	1.384	1.195	1.291
6	1.306	1.229	**0.769**	2.008	1.632	1.325	1.636	**1.113**
8	1.794	0.888	0.970	1.613	2.014	1.655	1.647	1.239
10	1.475	1.160	0.803	1.041	4.309	1.691	1.360	1.799

(d) E1.3: NMSE of NNs for Manutec r3 arm trained with ~ 1000 samples.

Table 5.6.: E1.1 and E1.3: NMSE of NNs for Furuta pendulum and Manutec r3 arm. Systematic search for suitable network topology.

Furuta pendulum	5 k	4 k	3 k	2 k	1 k	500
Gibbs (FITC100)	0.09	0.06	0.07	0.27	0.07	0.15
MLP	0.05	0.06	0.06	0.09	0.10	0.10
NN (6 × 150)	0.05	0.05	0.06	0.09	0.09	0.12
Manutec r3 arm	5 k	4 k	3 k	2 k	1 k	500
MLP (FITC100)	0.69	0.6	0.96	0.64	0.86	0.96
ARD	0.64	0.57	0.94	0.70	0.95	1.60
NN (6 × 200)	0.98	0.78	1.23	1.12	1.45	6.95

Table 5.7.: E1.4: NMSE of NNs and GPs determined in E1.3 for Furuta pendulum and Manutec r3 arm trained on small subsets of the training set.

GP kernels on the full training set. The numbers in Table 5.5 indicate that while for the Furuta pendulum, 100 k samples are sufficient, the Manutec arm controller needs all 231 k samples to provide the best performance. Due to the very long training times (several hours in the worst case), it has been refrained from evaluating step E1.2 for GPs. It can be noted that using even sparse approximations of the GPs for these amounts of training data is impractical.

The evaluation of the accuracy on small training sets ($\sim$1000 training samples) for the Furuta pendulum and the Manutec r3 arm considering GPs and NNs is given in Tables 5.4b and 5.6c to 5.6d. Here, the accuracy achieved with GPs is on par with that of NNs. Several GPs could not be trained properly in all cases, leading to large outliers in accuracy. ARD and MLP kernel provide a reliable performance on the two problems. For the NNs on the small training set, the error tends to decrease with increasing layer size as well as with increased number of layers. Moreover, the difference between the networks trained with and without dropout layers is less than on the large dataset. The reason is that training using only 1000 training samples is difficult and often stops after less than 100 episodes to avoid overfitting.

The networks used in step E1.4 consist of six layers of 150 nodes for the Furuta pendulum and six layers of 200 nodes for the Manutec arm problem, both with 5% dropout. As in step E1.3, the performance of the NNs on the small training set is on par or worse than that of GPs with suitable kernels. The increase in the error for decreasing sizes of training sets is more significant for NNs than for GPs. Further, it can be seen that the accuracy of the GP-based approximations trained on the small-size training sets is only 50 to 100 % higher than of those trained on the full training set.

5.3.2. Local Online Gaussian Processes

The optimization of the GPs' parameters may be relatively time-consuming, which makes their online training difficult. However, there are several approaches that aim at training GPs online during data acquisition. For example, there is the influential work of Csató and Opper [38], which is implemented in the *Online Gaussian Processes* toolbox[1]. A more recent and slightly different approach by Le et al. [99] is a sparse method that is influenced by online support vector regression approaches. In contrast to the two previously mentioned global approaches, Nguyen-Tuong et al. [129] propose a local method that partitions the input space into multiple regions for which local GPs are trained. Due to the smaller covariance matrix, training and prediction can be performed much faster for local models that use only a few training samples than GPs that operate on the full training set.

Wilcox and Yip [201] build upon the work of Nguyen-Tuong et al. and several others and propose *SOLAR-GP*[2], which is designed to be trained online to adapt to a stream of training data. These properties perfectly fit R4), preliminary tests on the Furuta and Manutec test sets have thus been performed during the work for this chapter. However, the accuracy is notably worse than that of the standard GPs or the global approximations used in this thesis. The implementation of *SOLAR-GP* enforces the use of the anisotropic squared exponential kernel function (see Equation (A.1)) – other kernel functions would require the rewrite of a significant part of the code. It is reasonable to assume that the use of other kernels (e.g., MLP, Gibbs or ARD kernels (cf. Appendix A.1), which are more suitable for the approximation of feedback control policies, see Section 5.3.1), will improve the accuracy of this method.

The advantages in terms of runtime performance for predictions are limited and depend on one important parameter: the number of local GP models. To provide a prediction for some given state, *SOLAR-GP* has to evaluate all local GP approximations to compute a weighted sum of these predictions. The weights depend on the distance of the respective GP data from this state. This limits the number of local models for which the requirement R2) regarding the prediction time can be met to around 10 to 15 for the Manutec arm.

Due to its inferior accuracy and the limitation on the number of local models to meet the real-time requirement, *SOLAR-GP* is not considered in more detail in this chapter. Local models have been extensively studied and the number of papers on this topic is large (cf.

[1] `https://www.cs.ubbcluj.ro/~csatol/SOGP/code/`

[2] `https://github.com/ucsdarclab/SOLAR-GP/tree/master`

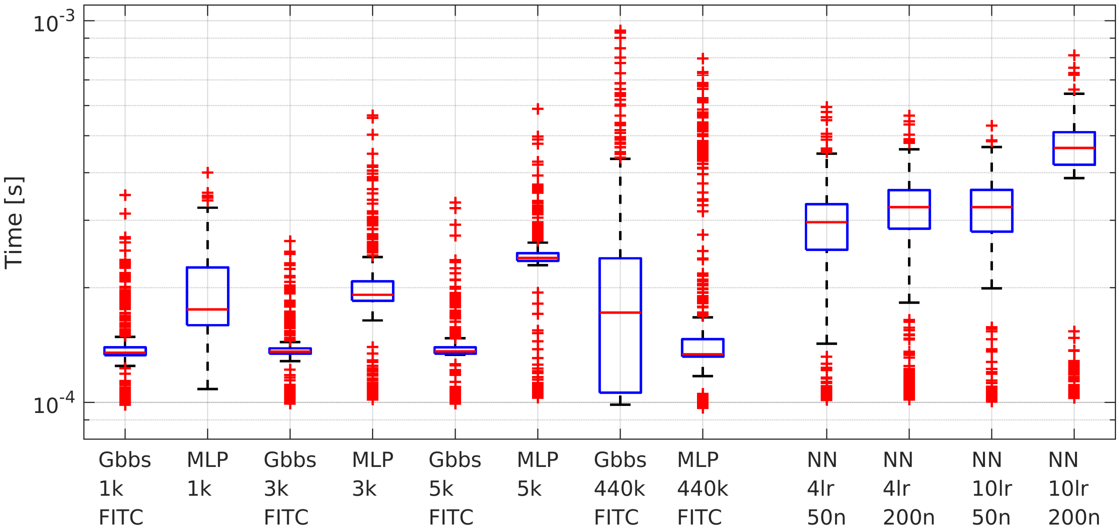


Figure 5.3.: Prediction times of learned controllers based on GPs and NNs with different hyperparameters, trained with differently sized training sets.

the review paper by Liu et al. [108]). It may therefore be worth investigating the use of local models further in future work.

5.3.3. Runtime Performance of the Control Approximations

Prediction performance is evaluated in simulation on a laptop with Intel i7-6500U processor with two cores at 2.5 GHz and Intel HD Graphics 520; training has been done on a better machine with a dedicated GPU. When controlling real systems, there is also some communication delay. The runtime of the approximated controller for the Furuta pendulum based on a selection of control approximations from Section 5.3.1 is given in Figure 5.3. During simulation, uncertainty information is not needed. Thus, MC dropout is not used such that the NN prediction is called only once for each control value. All approximation methods provide predictions within 1 ms. For GPs, the choice of the kernel function has a relevant impact on performance. The size of the training set has a huge impact on the runtime for exact GPs; sparse GPs reduce this impact significantly. The prediction times for NNs are, on average, somewhat higher than for GPs.

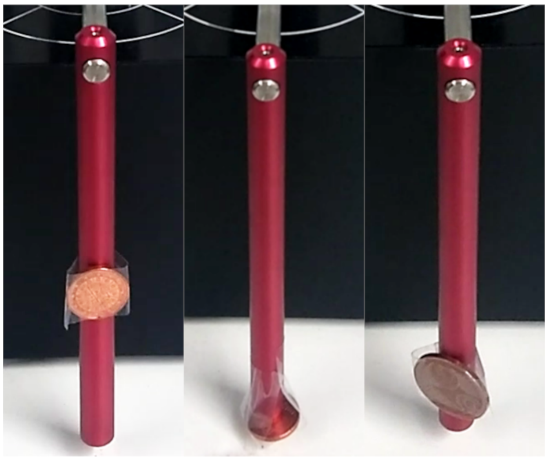


Figure 5.4.: Furuta pendulum with perturbed dynamics using coins. From left to right: one, one, and five euro cents as weights attached at different positions to the pendulum.

5.4. Experiments on the Quanser Furuta Pendulum

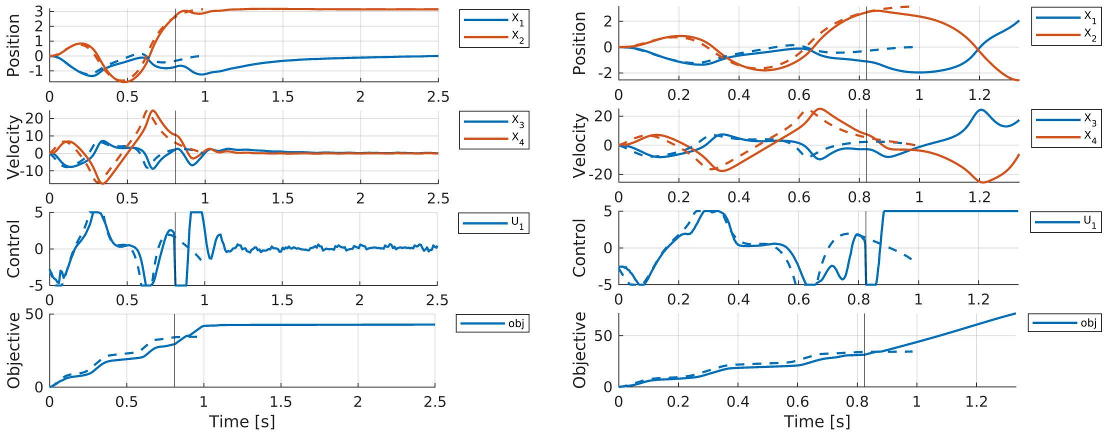
The feedback controls for the Furuta pendulum problem computed in Section 5.3 for various topologies, kernels and sample sizes are applied to the Quanser rotary inverted pendulum, see Figure 5.1. Pyrado, which is part of the SimuRLacra framework [123], is used for communication with the hardware and to run the control loop (at $500\,\mathrm{Hz}$). The learned feedback controller brings the hanging pendulum into an upright position close to the goal state. Then, the control is switched to a PI controller to stabilize the pendulum around the goal state, as described in Section 5.4 (also cf. [214]). The condition that triggers the switch to the PI controller is $t > 0.3 \wedge |1 + \cos(x_2)| < 0.1$, i.e., when the pendulum is within $25.8°$ from the upright position and at least $0.3\,\mathrm{s}$ have been passed. The time at which the switch occurs is denoted by t_{switch}. The Furuta pendulum is controllable (see Chapter 6 of Åström and Murray [5]) at the final state. The PI gains are designed with a linear quadratic regulator for the dynamic model linearized around the goal state with an experimentally tuned cost function. For this problem, the integral part of the PI controller is omitted (effectively resulting in a simple P controller) since gravity compensation is not required at the final state. The parameters used are $Q_x = \left(\begin{smallmatrix} 2.0 & 1.0 & 0.3 & 0.5 \end{smallmatrix}\right)$ and $Q_u = \left(\begin{smallmatrix} 1 \end{smallmatrix}\right)$. The output from both controllers is clipped to stay within the interval $[-5, 5]$. All controllers are implemented in C++ (NNs using the TensorFlow C API) and called

	440k	270k	100k	62k	24k	5k	4k	3k	2k	1k	500
Unperturbed Furuta pendulum											
Gibbs	2.07	-	-	-	-	1.12	1.91	2.07	2.08	0.89	2.08
MLP	1.25	-	-	-	-	1.58	1.58	1.25	1.74	2.09	0.67
NN	2.08	1.34	0.91	0.69	0.48	0.89	0.78	1.18	1.63	2.09	0.64
Furuta pendulum with 1 cent (2.3 g) at side											
Gibbs	0.86	-	-	-	-	2.07	0.81	2.07	0.82	2.09	2.08
MLP	0.79	-	-	-	-	0.76	0.66	0.77	0.66	2.08	0.69
NN	0.79	0.62	2.07	2.09	2.09	2.08	1.14	2.08	0.41	2.08	2.08
Furuta pendulum with 1 cent (2.3 g) at tip											
Gibbs	1.53	-	-	-	-	2.08	1.38	NaN	1.35	2.09	2.09
MLP	0.97	-	-	-	-	0.83	0.87	0.96	0.81	2.09	0.67
NN	1.22	0.91	0.85	2.09	2.09	2.08	0.53	0.64	0.58	2.09	2.09
Furuta pendulum with 5 cents (3.92 g) at tip											
Gibbs	0.96	-	-	-	-	2.03	0.74	NaN	2.08	2.06	2.09
MLP	2.08	-	-	-	-	2.08	0.72	2.08	0.91	2.09	0.93
NN	0.97	2.08	2.08	2.09	2.09	2.07	2.09	2.08	2.08	2.09	2.09

Table 5.8.: Experiments on real-world Furuta pendulum depending on the number of training samples. The numbers give the arm deflection (in rad) required to stabilize the system and indicate how accurately the trajectory was followed. The colors visualize the underlying numbers, red indicates that the system has not reached a stable final state.

via the Python C interface from Pyrado. In the approach presented in this thesis, the computation of the trajectories that provide the training data is based on a system model, which may be inaccurate (sim-to-real gap, see [125]). To analyze how well the learned controllers can be applied to real systems that potentially deviate from the model used for data generation, the performance of the learned feedback controls is also evaluated on systems with perturbed dynamics by attaching weights to the pendulum (Figure 5.4). It is, of course, possible to consider the additional weights in the system model or to use machine learning to learn the perturbed system dynamics from recorded data. However, this is not the subject of this evaluation.

When during operation the state constraint for the first joint is violated, the execution is stopped immediately. The state constraints are considered during the creation of the training data for the feedback controllers, but the PI controller is completely agnostic of this limitation and will violate the constraints if required to compensate for deviations from



(a) Successful execution of the unperturbed Quanser Furuta pendulum. The learned control policy is represented by a GP with MLP kernel and trained with 3000 data points.

(b) Unsuccessful execution of the unperturbed Quanser Furuta pendulum. The learned control policy is represented by a GP with Gibbs kernel and trained with 3000 data points.

Figure 5.5.: Results of the execution of two different learned policies on the Quanser Furuta pendulum. The dashed lines show the optimal trajectory, the full lines give the values measured on the real system. The switch to the PI controller at $0.86\,\mathrm{s}$ and $0.82\,\mathrm{s}$ is marked by the vertical line.

the goal state. Trajectories that directly approach the goal state and are already close to it when the switch is performed (the condition considers only the second joint's state) will cause only small balancing movements of the PI controller. In contrast, large deviations from the goal state or a disadvantageous state at t_{switch} lead to strong deflections of the arm caused by the PI control to balance the pendulum, which makes a failure of the execution because of constraint violation likely. The extent of the arm deflection that the PI controller needs to balance the system is used as an indicator of how well the learned control performs. Larger values are worse because these trajectories get closer to the joint constraint (cf. Table 5.2) of the arm.

The results from the experiments on the real system with and without perturbations (see Figure 5.4) are given in Table 5.8. The reported values represent the maximum arm deflection caused by the PI controller when combined with different learned feedback control policies. For illustrative purposes, Figure 5.5 presents the results of a successful and a failed execution for two different control policies. The joint constraint of the mechanical system is $120°$ (or $2.094\,\mathrm{rad}$), whereas the optimal control problem formulated in this thesis employs a slightly stricter constraint of $115°$ ($2.007\,\text{rad}$). Consequently, arm deflections above 2.007 are considered as failed executions and marked in red, but the values in the table range up to the mechanical constraint of 2.094. The NaNs denote cases where execution has failed before the switch to the PI controller was performed. Apparently, the MLP kernel performs better than the Gibbs kernel on the real system. In the unperturbed cases, the NN controller outperforms the GP-based control. In general, however, it seems that GPs are more capable of handling deviations from the model used for training. With a suitable kernel (MLP in this case), GPs perform on par with the NNs on the real system with perturbed dynamics, although their accuracy on the test set examined in Section 5.3.1 is lower.

5.5. Training Data using the Complementing Start State Selection Methods

In Chapter 4, the proposed start state selection methods are evaluated only in simulation. In this section, a brief evaluation of the iterative extremal field approach using all four complementing start state selection methods presented in Chapter 4 on the (unperturbed) real Quanser Furuta pendulum is performed. The parameters used for the start state selection methods are as follows: adjoint-based: $[b_l, b_u] = [0.15, 0.25]$, $\nu = 1.2$; sensitivity-based: $\mu_{\text{sens}} = 0.15$; simulation-based: $\mu_{\text{sim}} = 0.2$, $d_{\text{sim}}^{\max} = 0.15$; Halton-based: $\mu_{\text{H}} = 0.2$. Based on the findings in this chapter, an MLP kernel is used on the full training data (FTC). The parameters of the PI controller and the switch condition are the same as for the experiments in the previous section. Fifteen iterations, resulting in 1362 data points from 15 trajectories, suffice for a successful swing-up. The result is given in Figure 5.6, which shows the joint trajectories and control values of the simulation (full) and the optimal reference trajectory (dashed). As can be seen, the Furuta pendulum can be stabilized in an upright position. The vertical line marks the time $t_{\text{switch}} = 0.86\,\text{s}$ at which the switch to the PI controller is performed. The result supports the claim that a sophisticated selection of start states for trajectories allows the effective use of few data.

5.6. Discussion and Conclusion

In this chapter, GPs and NNs are systematically analyzed and compared with respect to specific requirements to determine their suitability for approximating feedback control

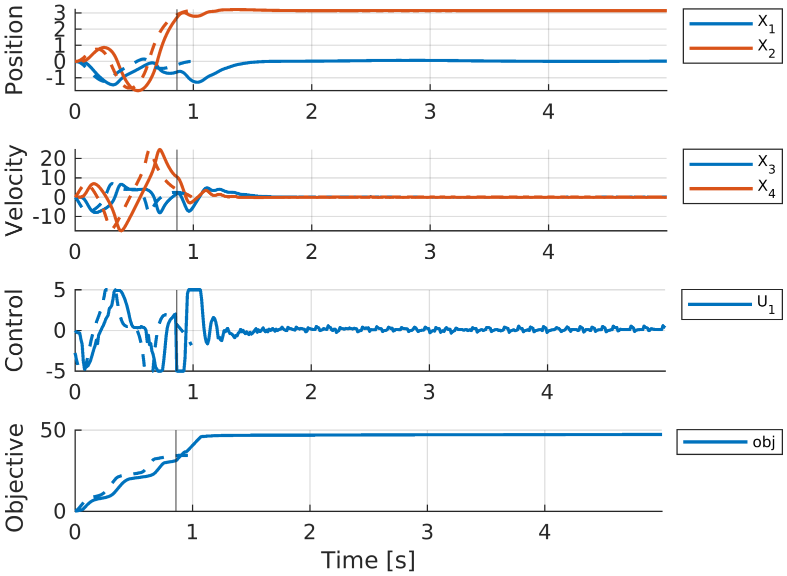


Figure 5.6.: Result of an experiment on the Furuta pendulum using training data from the extremal field approach with complementing start state selection strategies. The dashed lines show the optimal trajectory, the full lines give the simulation result. The switch to the PI controller at $0.86\,\mathrm{s}$ is marked by the vertical line.

policies on a test set and a real-world system. Feedforward networks of different depths and sizes and GPs with different kernel functions are examined to identify the hyperparameters for each approximation method that are best suited to the task. For evaluation, the full nonlinear dynamic models of an industrial Manutec r3 robot arm [136] and the Furuta pendulum [61] are used. The systematic comparison of NNs and GPs is performed with a focus on the requirements of feedback control policy approximations resulting from optimal control problems for robotic systems with highly nonlinear system dynamics. To the best of the author's knowledge, a systematic comparison focusing on the suitability of feedback control policy approximation has not been done before.

For NNs, the results indicate that the number of nodes per layer has a stronger influence on the error than the number of layers: A high number of layers does not seem to be necessary to obtain a high accuracy. However, the selection of the network topology is most likely problem specific. For the two systems considered, 150 to 200 are sufficient, but the results for the Manutec r3 robot arm suggest that more nodes per layer may be necessary for more complex systems. The use of dropout layers notably decreases performance but is required (if MC dropout is used) to provide uncertainty information. For GPs, the selection of kernels is important and must be done problem specific and depending on whether exact or sparse GPs are used. The sparse GP approximation FITC barely affects accuracy for suitable kernel functions and sufficient amounts of training data. However, using more than a few thousand samples to train a GP gives little benefit as training becomes cumbersome and does not justify the moderate increase in accuracy. NNs have a lower error than the GPs on large training sets. However, the runtime of NNs for predictions is somewhat higher than for small and efficiently trained GPs, especially when a sparse approximation like FITC is used.

The most accurate feedback control approximations are used as controllers on the real-world Furuta pendulum. In these experiments, the NN is more reliable on the system with unperturbed dynamics. However, the success rate of the NNs drops rapidly if the system dynamics are perturbed. Both GPs used in the experiments outperform the NNs for all perturbed dynamics. The NNs seem to be more susceptible to the sim-to-real gap than GPs. The results presented in this chapter clearly indicate that the GPs can better generalize to perturbed dynamics, at least if an appropriate kernel is used. This result is independent of the amount of training data used to train the feedback control policy. Overall, it is demonstrated that the learned feedback control policy can also handle systems with perturbed dynamics if an appropriate approximation method is used. This shows that control policies trained with data from a set of optimal trajectories can compensate for deviations from the model used to generate the data.

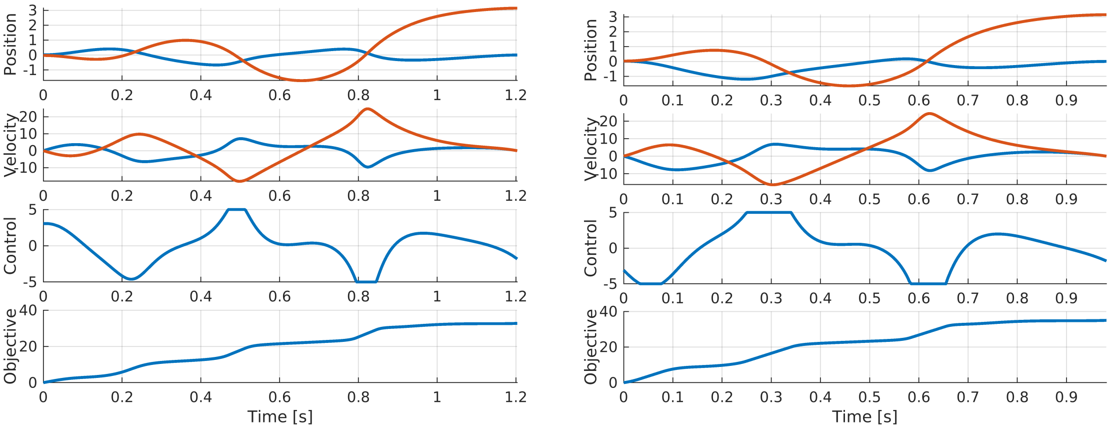
6. Identification of Solution Clusters

In Chapter 5, a Quanser Furuta pendulum is used in simulation and on real hardware for evaluation. Considering the underlying OCP (see Section 5.2.4), the following observation can be made: For different start states in close proximity to the nominal start state, DIRCOL provides several different locally optimal solutions with approximately the same cost. This happens even though the rotational symmetry of the problem is not considered by accepting only the pendulum arm position $\alpha = 0$ as final state. An example of two different trajectories representing a locally optimal swing-up of the Furuta pendulum is shown in Figure 6.1. The running cost is 32.7 for the first trajectory and 35.0 for the second, so the difference between the trajectory costs is about $10\,\%$.

Merkt et al. [117] call this “multimodality”. They note that “multimodality can greatly impact the quality of prediction obtained using function approximation as regressors smooth across the boundaries between clusters or modalities.” In this chapter, it is examined if a decrease in accuracy of the feedback control policy approximation provided by GPs occurs for the Furuta pendulum problem considered in this thesis. One way to deal with multimodality is to cluster the optimal trajectories computed by DIRCOL and use only trajectories from one cluster to provide the training data.

In this chapter, a novel distance measure for robotic trajectory data is presented, based on a semantic compression step. It transforms the raw trajectory data into a sequence of features (which are characteristic points in the graph) that can be used in an existing sub-sequence based distance metric. This measure is used in a hierarchical clustering algorithm to identify trajectories with similarly shaped graphs. The advantage of this approach is that relevant feature classes can be chosen depending on the application or the considered problem.

The idea behind the novel trajectory distance measure is that the exact path of a trajectory is less relevant for clustering. Much more important is the sequence of some meaningful features (like extrema, jumps, roots etc.) and their salience in the trajectory. Thus, the focus is not on measuring small differences between very similar trajectories as this is,



(a) Example trajectory 1 with running cost 32.7

(b) Example trajectory 2 with running cost 35.0

Figure 6.1.: Example of two different locally optimal solutions of the Furuta pendulum swing-up problem with approximately the same cost.

for example, done when the error between a reference and some executed trajectory is measured. Instead, a measure is developed that captures differences in the general characteristic shape of trajectories, with the goal in mind to use this measure to cluster a set of trajectories into groups of similar paths.

At the end of the chapter, control policies are trained with data from trajectories of a single and of multiple clusters. The performance of these feedback control policies controlling the real-world Furuta pendulum is compared to find out if the extremal field approach on the considered OCP is affected by multimodality.

6.1. Related Work on Trajectory Clustering

Trajectories and trajectory clustering are ubiquitous in various research fields like geographics and traffic (airport data [133]), biology (tracking of animals [13]), or animation and computer graphics (motion capturing [209]). In the field of robotics, trajectories play a central role for path and motion planning (e.g., [179], [117]). Clustering of trajectories is an important tool for analysis, recognition and prediction of motions and has been used in humanoid robotics, e.g., to estimate the gait phase [146], to learn motion primitives from observations [95], to analyze posture data [12], or in the field of human-robot interactions to build a database structure for human motions that allows the robot to respond faster [207], to recognize hand gestures as input commands [111], [132], [100] or to recognize and predict human or robotic actions and movements [208], [110], [179].

Gaussian Mixture Models have been used by Lee [100], Piperakis et al. [146] or Luo et al. [110] for clustering trajectories. However, this approach requires estimating multiple Gaussian distributions, which can be computationally costly, and the result is sensitive to initial points. Research has been done to cluster trajectory data using NNs, e.g., using auto-encoders as in [208, 133, 146] or specialized structures like self-organizing incremental NNs [132]. NNs can provide good performance on large-scale datasets [133]. However, they require large amounts of training data to generalize well and provide limited or no insight into how the clustering is done.

In Yao et al. [208], an auto-encoder is trained to extract a representation of fixed length from a trajectory, which is used in a classic clustering algorithm. The input of the auto-encoder is based on the changes of speed and rotation in sliding windows. This exhibits some similarities with the feature-based approach regarding the use of a feature sequence. However, the approach presented in this chapter does not rely on learning. Olive et al. [133] train an auto-encoder and use the low-dimensional information from its latent space for a second clustering step.

Widely used clustering algorithms are hierarchical clustering (Basoeki et al. [12], Kulić et al. [95]), k-means (Maharani et al. [111]) or density-based clustering. See, for example, the review of Bian et al. [24] for an overiew. The vast majority of clustering algorithms require a distance function to measure the distance or similarity of two trajectories. The construction of centroids needed for the k-means algorithm is not intuitive in the context of trajectory clustering. An advantage of hierarchical over density-based clustering is the better interpretability of the results, as hierarchies of clusters can be represented in dendrograms. For these reasons, hierarchical clustering is used in this chapter.

A large part of trajectory distance measures can be divided into those based on Lp-norms (well-known are, e.g., DTW and related, or the *Fréchet* distance) and those based on an edit distance (like *edit distance with real penalty*, *longest common sub-sequence*) [177]. Their properties have been studied extensively [181, 192]. These distance measures are typically applied to some condensed representation of the trajectories that are computed, for example, using principal component analysis as in [146], multiple correspondence analysis [12] or hidden Markov models (HMM) [132, 95]. These reductions all are not intuitive to interpret, and HMMs additionally require training.

In the context of social sciences, distance measures are often used to compare sequences of categorical elements, which are “an ordered list of successive elements chosen from a finite alphabet” [176]. Distance measures for semantic sequences are reviewed in [176]. The work in this chapter is built upon the work of Elzinga et al. [54], who present a measure called SVRspell, which is adopted to work for trajectory clustering. SVRspell is based on the number of matching sub-sequences in strings, which correspond to feature sequences of trajectories. The extraction of features from raw trajectories is related to Schmid et al. [162], who proposed a trajectory compression method for geographical movement data that uses semantic information to transform raw trajectory data. A review of this topic has been presented by Parent et al. [137]. However, there are several differences between the trajectory representations that reflect the specific requirements of the application considered here. In particular, there is no underlying map; consequently, the raw trajectory cannot be reconstructed from feature-based representation, in contrast to [162].

6.2. Description of the Feature-Based Trajectory Distance Measure

In motion planning for robotic systems, there are state and control trajectories to describe the state of the system in combination with the control values required to generate the motion. In this chapter, the combination of a state trajectory s and a control trajectory c synchronized in time is called *motion plan*:

$$T = \{s : [0, t_f] \rightarrow R_s \subset \mathbb{R}^n, c : [0, t_f] \rightarrow R_c \subset \mathbb{R}^m\}, \tag{6.1}$$

where $t_f > 0$ is the trajectory’s terminal time. Trajectories with $t_f = 1$ are called time-normalized.

Trajectories are typically represented as sequences of data points associated with time stamps. They can be extracted from image data, the output of a motion capture system or internal sensors that provide data for trajectories at discrete time steps. This data format can be interpreted as linear splines. Trajectory data may also originate from numerical optimal control solvers like DIRCOL that is used in this thesis. The state and control trajectories provided by DIRCOL are represented as cubic and linear splines. To generalize from the trajectory representation, trajectories are in this chapter represented as mappings (as in (6.1)). The presented approach is still applicable to trajectories consisting of discrete sequences, resulting from observations, as these can be interpreted as continuous linear spline functions.

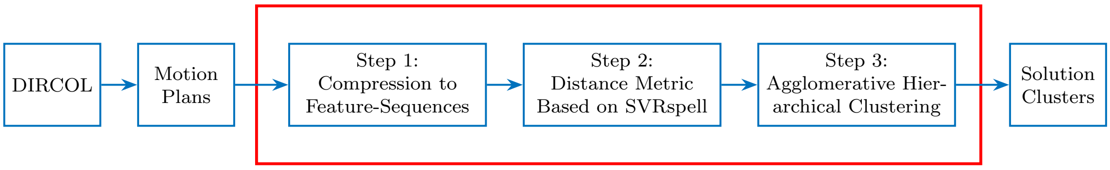

DIRCOL
Motion Plans
Step 1: Compression to Feature-Sequences
Step 2: Distance Metric Based on SVRspell
Step 3: Agglomerative Hierarchical Clustering
Solution Clusters

Figure 6.2.: The three major steps of the trajectory clustering approach.

Further, in this chapter it is assumed that there are box-constraints on the trajectories' codomains that are induced, for example, by joint limits or control bounds. This means that the joint space can be represented as

$$R_s := \prod_{i=1}^{n} [l_{s,i}, u_{s,i}] \tag{6.2}$$

for some lower and upper bounds $l_s, u_s \in \mathbb{R}^n$. The control space R_c is defined analogously. Nevertheless, the presented approach is applicable with small changes to problems without bounds. These changes are briefly described below when necessary.

In this chapter, an approach to cluster motion plans generated by the optimal control solver DIRCOL is presented. The trajectory clustering can be roughly divided into three major steps (see Figure 6.2). In the first step, each trajectory is compressed to a sequence of high-level features augmented with additional information about timing and salience. In the second step, the sequence-based representation of the trajectories computed in step one to measure distances among them is used. The distances from the second step form a distance matrix used in the third step as input for standard agglomerative hierarchical clustering (see [126]) to identify similar trajectories in an examined set. The distance measure for trajectories and motion plans, consisting of step one and two, is the main contribution of this chapter and will be described in detail in the following subsections.

6.2.1. Step 1: Construction of Sequence-Based Representation

This subsection describes the extraction of features from time-dependent raw trajectory data. Following the definition of a multidimensional sequence in [60], a *feature sequence* is defined as a sequence of elements $e_1, \ldots, e_q$ where each element e_i is a triplet of attributes $e_i = \left(e_i^{\text{cat}}, e_i^{\text{time}}, e_i^{\text{val}}\right)$. In this triplet, e_i^{cat} denotes the kind of feature this element represents (*feature class*), e_i^{time} gives the time stamp of this feature in the normalized trajectory and $e_i^{\text{val}} \geq 0$ provides information about the importance of the feature in the sequence. Several examples of such attributes will be given later on. For a given trajectory, $m + n$ sequences are computed, one for each dimension in the state and control (cf. Equation (6.1)).

The selection of relevant features is a crucial part of the application of this method and requires careful design. It depends on the one hand on the features that occur in at least some trajectories of the examined set and on the other hand on the relevance of these features for the problem or application underlying the trajectories. In this work, maxima ($\wedge$) and minima ($\vee$), active box constraints (upper bound L_u, lower bound L_l) and for one problem roots (0) are used as feature classes. These are the dominant characteristics that are apparent in the trajectories' graphs resulting from the problems considered in this thesis. Examples of other features are jumps, steep inclines or roots/signs of derivatives. Since these do not occur in the examined trajectories they are not considered in the following. The selection of relevant characteristics for the distance measure is highly problem specific.

The state trajectories are represented as cubic splines, the control trajectory as linear splines. This is DIRCOL's output format, typical for collocation-based optimal control solvers. For different representations (e.g., quintic splines), it may be necessary to adapt the feature extraction approach. Trajectory data from sensors or images is typically a sequence of (potentially multidimensional) time-stamped values. This can be interpreted as linear splines, such that the presented approach can be used without further adaptation.

To extract features from a trajectory, a proceeding is required for each feature class to find the times e_i^{time} at which the feature e_i^{cat} occurs. Furthermore, a value e_i^{val} needs to be assigned to each feature representing its salience in the trajectory shape. The following describes the approach to get these features from linear and cubic splines.

Extrema

In linear splines, maxima and minima can only occur at knots (times at which the polynomial changes). To find the extrema, the splines are thus searched for sign changes between slopes of consecutive intervals:

$$\frac{u^i - u^{i-1}}{t^i - t^{i-1}} \cdot \frac{u^{i+1} - u^i}{t^{i+1} - t^i} < 0 \tag{6.3}$$

for values u^i and time t^i at node i. For cubic splines, each polynomial $s_a^i \bar{t}^3 + s_b^i \bar{t}^2 + s_c^i \bar{t} + s_d^i$ (where $\bar{t}$ is the normalized time) is searched for extrema by checking if its derivative, a quadratic function, has roots inside the interval on which the spline is defined:

$$\bar{t}_{1,2} = \frac{-s_b^i}{3s_a^i} \pm \sqrt{\frac{-s_b^i - 3s_a^i s_c i}{9\left(s_a^i\right)^2}} \in [0,1]\,. \tag{6.4}$$

In e_i^{cat}, it is distinguished between *maximum* ($\wedge$) and *minimum* ($\vee$). To assign a value e^{val} to all maxima and minima, a concept called *(topographic) prominence* [109] as defined in [92] is used. Topographic prominence has been developed to value the significance of mountain summits; see Figure 6.3 for an illustrative example. In this work, the algorithmic definition of topographic prominence for functions with one-dimensional domain as given in [185] is used:

> To measure the prominence of a peak:
>
> 1. Place a marker on the peak.
> 2. Extend a horizontal line from the peak to the left and right until the line does one of the following:
>   - Crosses the signal because there is a higher peak
>   - Reaches the left or right end of the signal
> 3. Find the minimum of the signal in each of the two intervals defined in Step 2. This point is either a valley or one of the signal endpoints.
> 4. The higher of the two interval minima specifies the reference level. The height of the peak above this level is its prominence.

This definition is slightly adapted: the computed prominence is normalized by dividing it by $u_s - l_s$ (or $u_c - l_c$). At this point, the assumption of bounded codomains is used. For unbounded co-domains, one could still normalize the prominence using the inverse tangent or any other mapping $[0, \infty) \to [0, 1)$. Since prominence is defined for maxima only, the prominence of a minimum of a function f at t_m is defined as the prominence of the maximum of the function $(-f)$ at t_m. Finally, all extrema with a prominence lower than a certain threshold are discarded to avoid long feature sequences with many irrelevant features.

Active box constraints

To find the active box constraint feature in linear splines c, it is again sufficient to consider only knots at times t_i and check if the condition

$$u_c - c(t_i) < \varepsilon \vee c(t_i) - l_c < \varepsilon \tag{6.5}$$

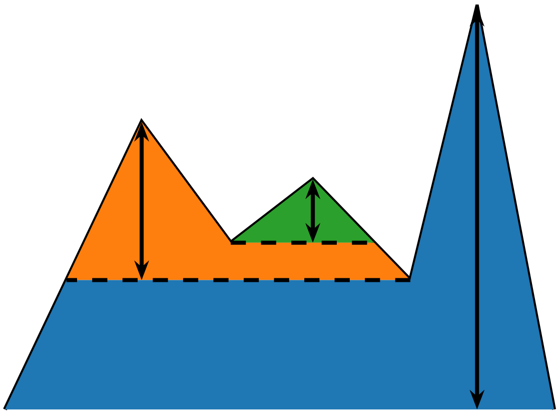


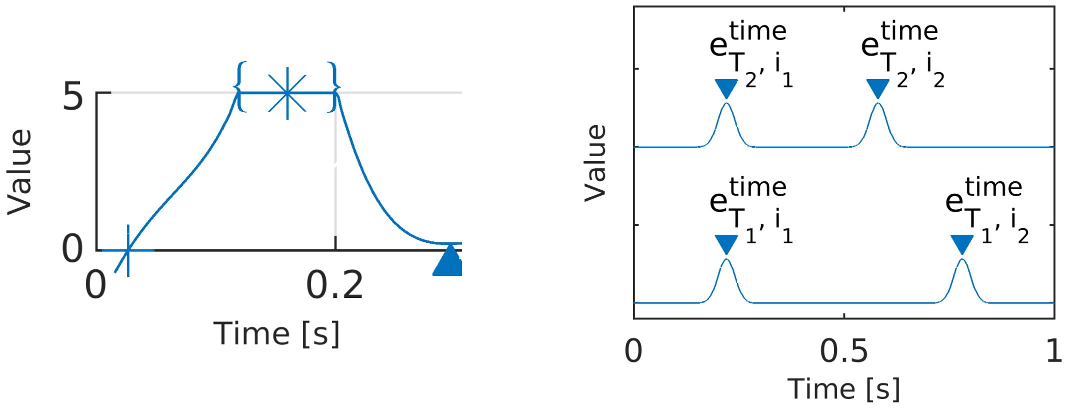
Figure 6.3.: Example illustrating topographic prominence

Fig. 6.3 is based on *ProminenceDiagram.jpg*, “Diagram of mountain range showing peaks and cols [...]” (`https://commons.wikimedia.org/wiki/Category:Topographic_prominence#/media/File:ProminenceDiagram.jpg`), created by SGBailey, licensed under CC BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0/`). Own work based on original version by SGBailey. ©2024. Figure 6.3 is openly licensed under CC BY-SA 3.0.

is fulfilled, with a small threshold $\varepsilon > 0$. For cubic splines, it is reasonable to check the condition (6.5) at knots and extrema and proceed as for linear splines. This simplification for cubic splines has shown to be effective in practice. Depending on whether the upper or lower bound is active, different attributes (L_u, L_l) are used in e_i^{cat}. Further, one could distinguish between a single touch point and the two points start and end of a constrained arc. In this work,a single feature is used for both cases and the time e_i^{time} of an arc is set to the center point between the start and end of the active constraint (see Figure 6.4a for an example). To get salience values e_i^{val}, active box constraints are treated as extrema (which they actually are) and compute the prominence value at these points.

Roots

Considering roots can be helpful for motions where symmetry is relevant. The roots (e_i^{cat} indicated by 0) are identified by finding the roots of all (linear or cubic) polynomials that



(a) Example: Two events "start/end of constrained arc" ('{' and '}') are merged into a single event "constrained arc" ($\ast$) in the center of the constraint.

(b) Example: Two trajectories with each two features in different temporal distances, which is penalized by Equation (6.9).

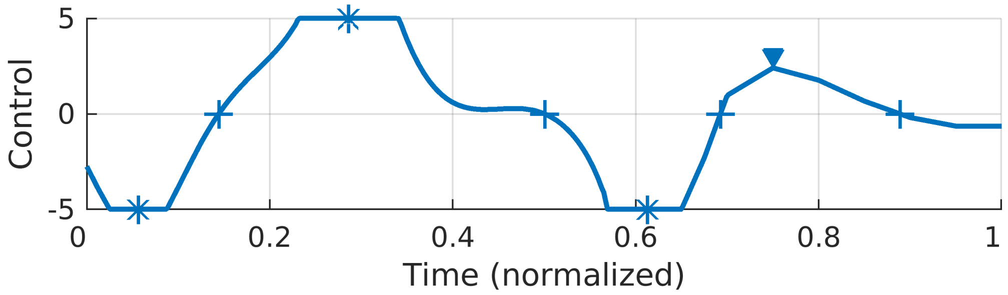
Figure 6.4.: Two examples to illustrate aspects of the construction of the sequence-based trajectory representation.

are part of the trajectory. The value of each root at e_i^{time} is computed using the slope m:

$$e_i^{\text{val}} = \frac{2}{\pi} \left|\arctan\left(m\right)\right| \tag{6.6}$$
$$m = \left.\frac{\mathrm{d}}{\mathrm{d}t} s(t)\right|_{t=e_i^{\text{time}}} .$$

Equation 6.6 assigns salience values close to one to roots that intersect the x-axis with large slope and values close to zero for roots with flat angles. For all roots at knots of linear splines, the subderivative $m = \frac{1}{2}(m^+ + m^-)$ is used, where m^- and m^+ denote the left and right limit, as they are not differentiable at these points. An example of a feature sequence for a one-dimensional trajectory, including time and salience values, is given in Figure 6.5.

In the selection of root as a feature for comparing trajectories, it must be considered that roots are not translation invariant and are thus not suitable for every problem. However, it can provide some benefit for trajectories that describe motions relative to some important reference points. Furthermore, it can be easily extended to similar feature classes: For some problems (e.g., foot position in task space, cf. 6.2.1), it may be necessary to consider



(a) Trajectory graph with features indicated by symbols: ▲ for minima, ▼ for maxima, $+$ for roots and $\ast$ for active constraints. The time is normalized to the interval $[0, 1]$.

i	1	2	3	4	5	6	7	8
e_i^{time}	0.06	0.14	0.29	0.50	0.61	0.69	0.75	0.89
e_i^{cat}	L_l	0	L_u	0	L_l	0	$\wedge$	0
e_i^{val}	0.22	0.99	1.00	0.97	0.74	1.00	0.31	0.97

(b) Feature sequence for the above trajectory graph. Feature labels: $\wedge$: maximum, $\vee$: minimum, 0: root, L_u/L_l: upper/lower box constraint active.

Figure 6.5.: Example illustrating the representation of a trajectory graph using a feature sequence.

not $y(t) = 0$ but more generally $y(t) = const$. Another simple extension of the root feature is to consider roots of higher derivatives.

Variability in the Feature-Class Design

The three presented feature classes are meant to be examples from a wealth of possibilities; feature-based distance measures are not restricted to those three feature classes. Examples for other features that may be considered are turning points, roots of the first or second derivative, jumps or discontinuities. Feature-classes can also be designed to require that multiple features occur at the same time in different dimensions of the trajectory.

The presented distance measure is not only applicable for trajectories in the joint space but also for trajectories in the task space. Moreover, it is also possible to combine joint and task space trajectories of the same motion in the same way as state and control trajectories are used together (see Equation (6.8)) to use features from both spaces. Task space trajectories can be computed from joint space trajectories either as a preprocessing step or on the fly by the feature-class. This offers a large variety of new feature-classes for several use cases in (humanoid) robotics. Feature-classes can be created that consider other derived trajectory information, for example, the end-effector position relative to some object, the center of mass, the center of pressure, swing-velocities of feet, the altitude of the feet, the head orientation or the view direction.

The flexibility and adaptability provided by the ability to define custom feature-classes is considered as important advantage of this method. Nevertheless, defining feature-classes that are appropriate for a specific problem can be elaborate, finding those that perform well may require some trial and error. The defined classes need to work on trajectory features that appear sufficiently frequent in the trajectories to be clustered and are suitable to distinguish between the different clusters. The implementation of new feature-classes requires a definition of how a features is detected in the trajectories and how its importance in the trajectory (salience value) is rated. The latter must be balanced among all the feature-classes used.

6.2.2. Step 2: Distance Metric for Feature Sequences

For two motion plans T_1 and T_2, the feature sequences are constructed as described in Section 6.2.1. For these sequences, a distance needs to be computed to compare T_1 and T_2. This is done using the measure based on the string kernel method described in [52, 54], which allows the incorporation of the features e^{time} and e^{val}. This measure is a metric, which means in particular that the triangular inequality holds, which "ensures coherence between computed dissimilarities" [176].

In the following, Elzinga's distance measure *SVRspell* [52, 54] is briefly summarized. Consider a finite alphabet Σ and the set Σ^* of all possible strings (character sequences) that can be built from Σ. Enumeration of the elements in Σ^* gives an infinite-dimensional vector space S where each entry corresponds to one string. For two strings $s_1, s_2 \in \Sigma^*$ the two corresponding vectors $r_{s1}, r_{s2} \in S$ list the number of respective sub-sequences that can be found in the strings. A short example is given in Table 6.1.

To get the distance between s_1 and s_2, Elzinga et al. compute the Euclidean distance between r_{s1} and r_{s2}. This can be formulated using only inner products, which can be

$$\begin{aligned} \Sigma = \{a, b\}, &\quad \Sigma^* = \{a, b, aa, ab, ba, bb, aaa, \dots\} \\ s = abb, &\quad r_s = \begin{pmatrix} 1 & 2 & 0 & 2 & 0 & 1 & 0 & \dots \end{pmatrix} \end{aligned}$$

Table 6.1.: Example of a string s and its vector r_s denoting how often all sub-sequences occur. In this example, the sub-sequences in s are *a*, *b*, *ab*, *bb*, *abb*.

computed even if the vectors are infinite-dimensional using *kernels*:

$$d(s_1, s_2) = \sqrt{\sum_{i \in \mathbb{N}} (r_{s1,i} - r_{s2,i})^2} = \sqrt{r_{s1}^T r_{s1} + r_{s2}^T r_{s2} - 2 r_{s1}^T r_{s2}}. \tag{6.7}$$

In [54], several extensions of SVRspell are described, some of them are used in this work. In the following, it is explained how this measure and its extensions are used to allow the application on the representation of motion plans from 6.2.1.

The alphabet in [54] corresponds to the set of trajectory feature classes, a string to the feature sequence, and a character to a feature e_i^{cat}. The distance between motion plans can be computed by applying (6.7) to each dimension of the feature sequence representation of the state and control trajectories. The resulting $n + m$ distance values d_i are combined into a single distance measure by taking the root of the sum of squared distances:

$$d_{\text{final}} = \left(\sum_{i=1}^{n+m} d_i \right)^{-\frac{1}{2}}. \tag{6.8}$$

The following extensions of [54] are used to include e^{time} and e^{val} in the distance measure.

Soft-Matching

Elzinga et al. describe how to add a concept named *soft-matching* to the string kernel, which allows quantification of similarity between distinct characters of the alphabet. As an example, consider the alphabet $\{a, b, c\}$ and the soft-matching given by the similarity 0.7 between a, b and 0.0 between both a, c and b, c. Then, the distance between words ab and aa is smaller than between the words ac and aa.

Soft-matching has been used to take the similarity of extrema and box constraints into account. The parameter p that gives the similarity between $\wedge$ and L_u or $\vee$ and L_l is between 0 and 1. A section of the soft-matching matrix for the extremum and box constraint features is given in Figure 6.6. When selecting the parameter p, care must be taken to ensure that the resulting soft-matching matrix is positive definite.

$$\begin{array}{c|ccccc}
 & \wedge & \vee & L_u & L_l & \dots \\
\hline
\wedge & 1 & 0 & p & 0 & \dots \\
\vee & 0 & 1 & 0 & p & \dots \\
L_u & p & 0 & 1 & 0 & \dots \\
L_l & 0 & p & 0 & 1 & \dots \\
\dots & \dots & \dots & \dots & \dots & I
\end{array}$$

Figure 6.6.: The soft-matching matrix in the SVRspell algorithm for extremum and box constraint features, depending on the parameter p.

Gap Penalty

The *Trail-algorithm* in [54] allows the introduction of a penalty term to decrease the influence of sub-sequences with large gaps. Gaps are differences between indices where a character of a sub-sequence occurs in a string. For example, the sub-sequence *abc* of the string *abac* has gaps 1 and 2 between *ab* and *bc*, respectively. This gap penalty can be an arbitrary data-dependent function, which allows to incorporate the normalized trajectory times to penalize differences in the time spans between two features in their trajectories, as exemplified in Figure 6.4b. To measure distances between feature sequences in the feature-based distance measure, the following gap weighting function is used:

$$g(i_1, j_1, i_2, j_2) = 1 - \left| \left(e^{\text{time}}_{T_1,i_2} - e^{\text{time}}_{T_1,i_1} \right) - \left(e^{\text{time}}_{T_2,j_2} - e^{\text{time}}_{T_2,j_1} \right) \right| . \tag{6.9}$$

The inputs i_1 and i_2 are indices of two features in the trajectory T_1 and j_1, j_2 are indices for T_2. In the kernel, each entry of r_s of a feature sequence is weighted by a product of the gap weights computed by g in the respective sub-sequence (slightly simplified). How to handle the case where a sub-sequence occurs more than once in a string is described more detailed in [54].

Run-Lengths

Finally, it is necessary to add the information provided by the salience e^{val} into the distance measure. In [54], Elzinga et al. propose an adaptation of their method to cover so-called *run-length encodings* of strings where repeated characters are encoded as numbers (e.g., *aaabcccb* as $a^3b^1c^4b^1$). This can also be used for “any quantifiable property of the characters”, which is the salience e^{val} in the presented distance measure. Accordingly, each sub-sequence of a string is weighted by the sum of such a run-length. However, products of the weights are more appropriate for this application. They ensures that the importance of sub-sequences containing features with very little salience is reduced as a whole. This extension requires a single line change in the existing algorithm: In the notion of [54], it suffices to modify line 5 of Elzinga’s Grid-algorithm to assign $m^1_{ij} \leftarrow t_{xi}t_{yj}$.

To conclude, *SVRspell* modified with three presented extensions (two of them already proposed in [54], one implemented for this work) provides a distance metric to compare feature sequences that represent motion plans. It allows incorporation of the additional information provided by temporal relation e^{time} and salience e^{val} of the features.

6.2.3. Step 3: Application of Hierarchical Clustering

The distance measure described in the previous section enables the construction of a distance matrix for a given set of motion plans, for which the distance between each pair in this set needs to be computed. This distance matrix is required by the agglomerative hierarchical clustering algorithm that constructs the set into subsets of similar motion plans. The merging of clusters is done using single-linkage. The number of resulting clusters is either predefined or results from a cutoff value. Both values must be tuned, which is facilitated by considering dendrograms that visualize the hierarchy of the clustering and the distances between the trajectories. The distance measure described in 6.2.1 and 6.2.2 is by no means restricted to agglomerate hierarchical clustering but can be used in every clustering approach that is based on pairwise distances.

6.3. Evaluation

The feature-based distance measure of motion plans based on SVRspell is compared with DTW on the raw pairs of state and control trajectories. DTW is based on dynamic programming and aligns the time scales of two different sequences or trajectories. See Figure 6.7 for an exemplary comparison of Euclidean and dynamic time warping matching of two sequences. DTW has first been applied for speech recognition by Sakoe and Chiba [158, 128]. It has been selected because it is, according to Su et al., “one of the most widely used distance measures” [177]. Computations are performed in MATLAB R2020a on a laptop with an Intel i7-6500U processor with two cores at 2.5 GHz. The DTW implementation of the MATLAB Signal Processing Toolbox (compiled code) and the

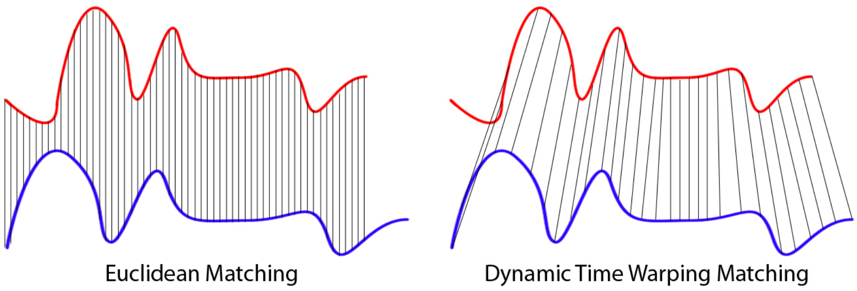


Figure 6.7.: Visual comparison of Euclidean matching and dynamic time warping matching of two time series.

Fig. 6.7 is based on *Euclidean_vs_DTW.jpeg*, “Difference in matching between Euclidean and Dynamic Time Warping” (`https://commons.wikimedia.org/wiki/Category:Dynamic_time_warping#/media/File:Euclidean_vs_DTW.jpg`),
created by XantaCross, licensed under CC BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0/`).

agglomerative hierarchical clustering from the MATLAB Statistics and Machine Learning Toolbox is used. The SVRspell algorithm is implemented in C++ and used in MATLAB via a MEX file. The feature extraction proposed in this chapter is written in MATLAB code. The sets of trajectories that are used in this evaluation are, if not stated otherwise, provided by the OCP solver DIRCOL [173].

6.3.1. Clustering of Furuta Pendulum Motion Plans

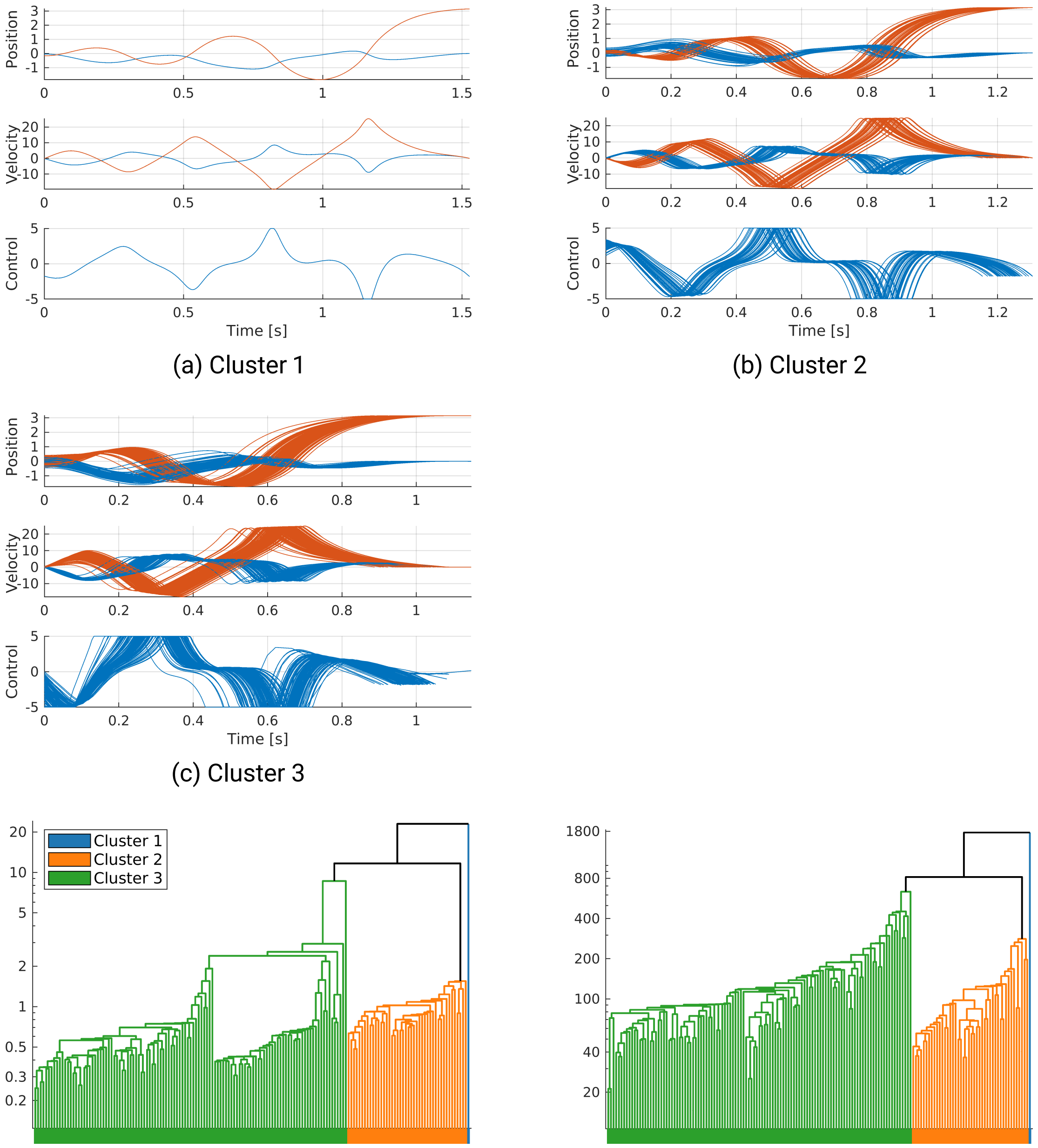
The underlying physical system of the motion plans in this subsection is the Furuta pendulum [61]. The motion plans are solutions of the OCP described in Section 5.2.4 for different start states, computed with the numerical solver DIRCOL. The problem formulation contains no explicit information about trajectory clusters or distance measures. The differences in the motion plans originate from the differences in the start states; the motion plans can be clustered into visually different solutions (see Figures 6.8a to 6.8c, Figures 6.9a to 6.9d or Figures 6.10a to 6.10e). By changing the parameterization of the dynamic model, three different test sets *Furuta 1* to *3* of 160 motion plans each have been created that are clustered separately. The 160 trajectories are manually clustered for each test set, these clusters are used as ground truth. Trajectories with visually similar graphs are grouped into one cluster. The ground truth clustering has been done before developing the feature-based clustering approach.

For this problem, the *root* feature is used for the state trajectories as the roots of the second joint indicate the number of swing-up motions. A qualitative evaluation is performed by comparing the dendrograms and clusters resulting from DTW and the presented method. Each motion plan in the test set is listed on the x-axis of the dendrogram, the distances between clusters are on the y-axis. The trajectory plots show the distinct clusters of the motion plans, which contain state (consisting of two joint positions and two velocities) and control trajectories. The qualitative evaluation is supported by comparing the number of correctly clustered motion plans as quantitative evaluation. The separation into clusters depends on the cutoff value chosen. Lower cutoff values lead to fragmentation into more clusters. Misclassifications with high distance to the other elements make a lower cutoff value necessary to separate the clusters. The results for both methods are given for the separation into the most favorable number of clusters to allow a fair comparison. The number of clusters into which the trajectories are separated is reported as number in brackets next to the distance measure name. If more than one cutoff value provides equally favorable results, both confusion matrices are reported.

The first test set *Furuta 1* has three different clusters containing 44, 115 and 1 trajectories. Clustering based on both DTW and the feature-based distance measure identifies all clusters correctly (see Table 6.2a). The dendrograms and trajectory plots in Figure 6.8 illustrate the subdivision into three clusters.

The second test set *Furuta 2* has four different ground truth clusters with 90, 38, 1 and 31 trajectories. The feature-based approach identifies the four clusters apart from two elements (Table 6.2b). DTW separates three single trajectories from the ground truth clusters and thus needs a lower cutoff value, leading to seven clusters. Supporting the quantitative result, the dendrogram for the feature-based approach (Figure 6.9e) is structured more clearly: The four clusters are separated and the more fine-grained splitting happens only at a lower cutoff level. The two misclassified trajectories are clearly visible in the trajectory plots 6.9b and 6.9d.

The third test set *Furuta 3* consists of four trajectory clusters with sizes 30, 30, 99 and 1. On this test set, the clustering computed with DTW equals the ground truth clustering, the feature-based distance measure assigns the single trajectory cluster to the largest cluster. The results for the third test set are illustrated in Figure 6.10, the quantitative results can be found in Table 6.2c.



(a) Cluster 1

(b) Cluster 2

(c) Cluster 3

(d) Result with Feature-based

(e) Result with DTW

Figure 6.8.: Clustering result of the *Furuta 1* test set. Subfigs. 6.8a to 6.8c give the clustering result of both distance measures (which equal the ground truth in this case). The two dendrograms illustrate the hierarchical structure of the clusters resulting from the different distance measures. The ground truth is given as colored bar on the x-axis.

GT	Feature-based (3)			DTW (3 clusters)		
	C_1	C_2	C_3	C_1	C_2	C_3
C_1	**1**			**1**		
C_2		**44**			**44**	
C_3			**115**			**115**

(a) Confusion matrices for *Furuta 1* test set

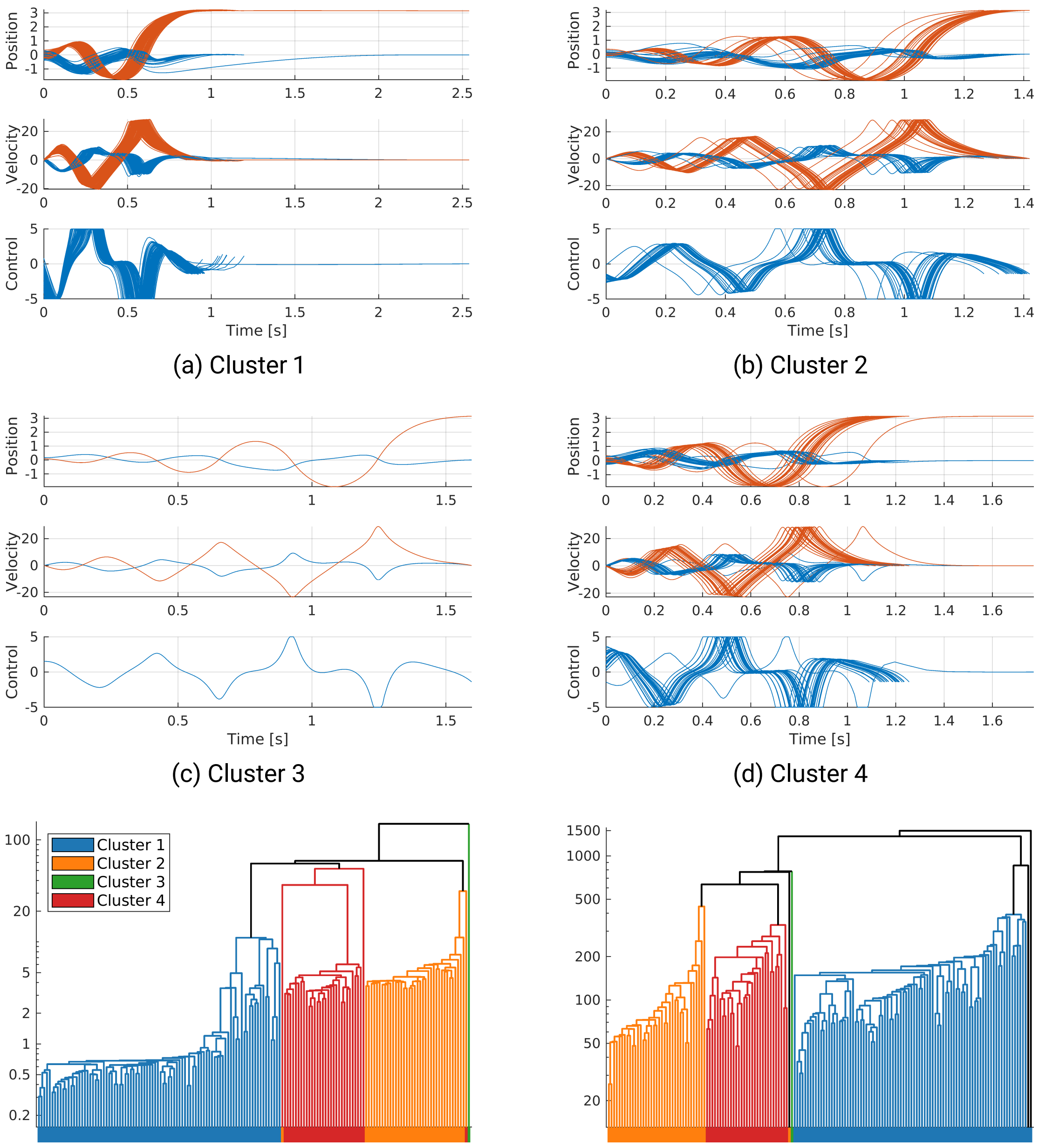
GT	Feature-based (4)				DTW (7 clusters)				
	C_1	C_2	C_3	C_4	C_1	C_2	C_3	C_4	...
C_1	**90**				**88**				2
C_2		**37**		1		**37**			1
C_3			**1**				**1**		
C_4		1		**30**				**31**	

(b) Confusion matrices for *Furuta 2* test set

GT	Feature-based (3)			DTW (4 clusters)			
	C_1	C_2	C_3	C_1	C_2	C_3	C_4
C_1	**30**			**30**			
C_2		**30**			**30**		
C_3			**99**			**99**	
C_4			1				**1**

(c) Confusion matrices for *Furuta 3* test set

Table 6.2.: The clusterings of the Furuta test sets as confusion matrices using the feature-based approach and DTW. The rows give the ground truth clusters, the columns the respective cluster assignments resulting from the two distance measures. The number of clusters is given as number behind the method name. Additional clusters are summarized in "...". The number of correctly classified motion plans is given on the diagonals.



(a) Cluster 1

(b) Cluster 2

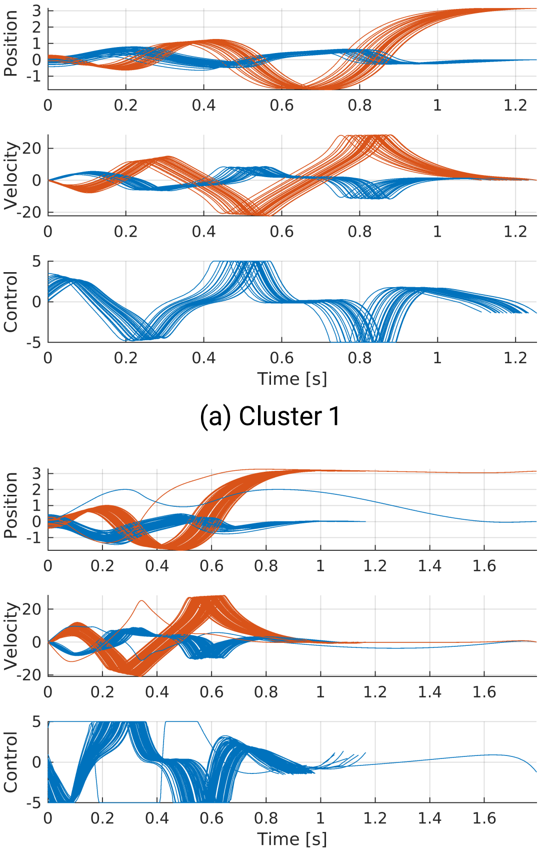
(c) Cluster 3

(d) Cluster 4

(e) Result with Feature-based

(f) Result with DTW

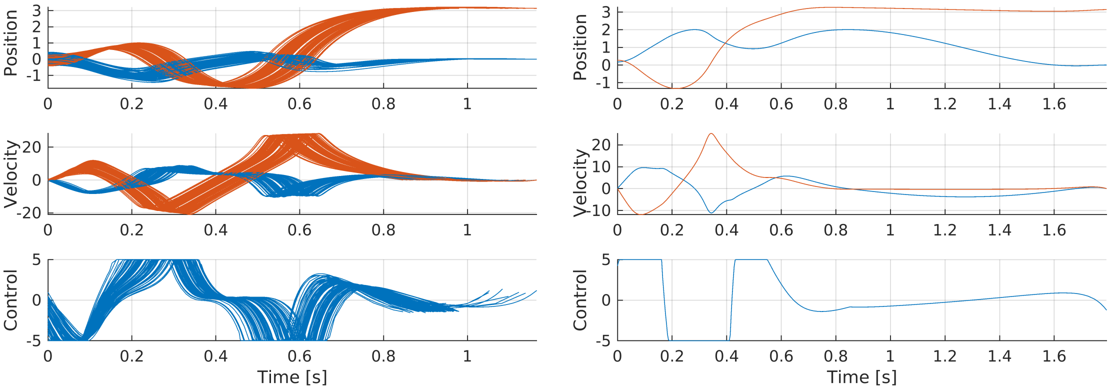
Figure 6.9.: Clustering result of the *Furuta 2* test set. Subfigs. 6.9a to 6.9d give the clustering result of the feature-based distance measure. The two dendrograms illustrate the hierarchical structure of the clusters resulting from the different distance measures. The ground truth is given as colored bar on the x-axis.



(a) Cluster 1

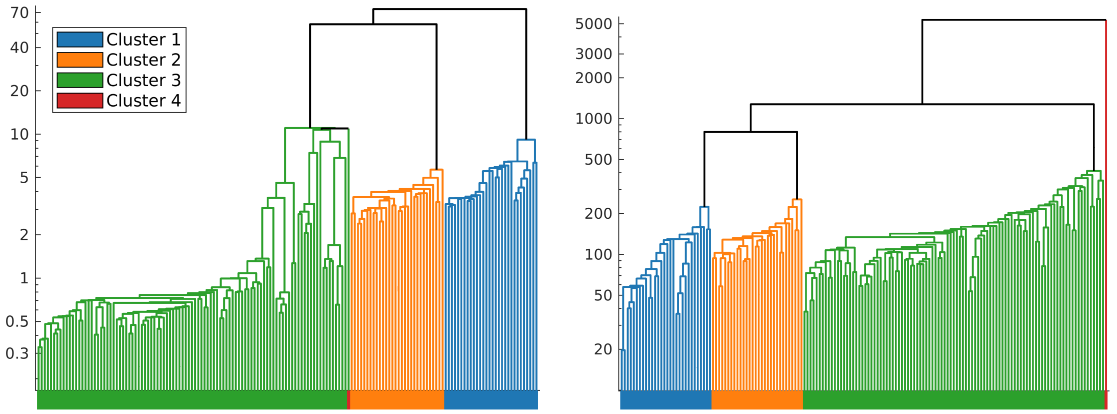
(b) Cluster 2

(c) Cluster 3



(d) Ground truth cluster 3

(e) Ground truth cluster 4



(f) Result with Feature-based

(g) Result with DTW

Figure 6.10.: Result of the clustering of the test set *Furuta 3*. Subfigs. 6.10a to 6.10c give the three clusters created using the feature-based distance measure. Cluster 1 and 2 are identical with the ground truth. Subfigs. 6.10d and 6.10e give the ground truth clusters for cluster 3 and 4. The plots show the progression of the state (position and velocity of two joints) and control trajectories. The two dendrograms illustrate the hierarchical structure of the clusters resulting from the different distance measures. The ground truth is given as colored bar on the x-axis.

6.3.2. Clustering of Manutec r3 Arm Motion Plans

In this subsection, the more complex Manutec r3 robot arm model is considered, which has three actuated joints, such that a motion plan consists of a six-dimensional state and a three-dimensional control trajectory. The motion plans that describe optimal point-to-point movements from different start states to the same goal state (cf. Section 3.3.2) are computed as before using the numerical solver DIRCOL. The test set based on the Manutec r3 arm consists of 30 motion plans that are clustered manually into 9 different movements. There are three larger clusters of ten, six and five motions, respectively (given in Figures 6.12a to 6.12c). The other clusters contain one or two trajectories and are neglected in this evaluation.

A property of motion plans resulting from this problem is a low-amplitude zig-zag behavior in one dimension of the control trajectory (see Figure 6.11). This validates the use of a prominence filter (a salience threshold of 0.02 is sufficient) for the extrema to remove the

GT	Feature-based				DTW			
	C1	C2	C3	...	C1	C2	C3	...
C1	**10**				**10**			
C2		**6**				**6**		
C3			**3**	2	4		**1**	
...		3		...		1		...

Table 6.3.: The results of the clustering as confusion matrix for the Manutec robot arm test set comparing the feature-based distance measure and DTW. The rows give the three largest ground truth clusters and the columns the respective cluster assignments resulting from the two distance measures. The number of correctly classified motion plans is given on the diagonals. Only the three largest clusters are considered, all others are summarized in the last row and column.

many maxima and minima. The prominence filter is thus important to reduce the length of the feature sequence. Moreover, it can compensate for some noise in the input data. The feature class *root* provides no useful information for trajectories of this test set and is thus not used for the state or control trajectories.

Table 6.3 shows that the number of correctly clustered motion plans in the three largest clusters is the same for the feature-based approach and DTW. Using the presented approach, the large cluster 1 is identified correctly, but cluster 3 is incomplete and the algorithm adds some additional motion plans to the second cluster. In contrast, DTW is unable to distinguish between the first and the third cluster (three of four motion plans

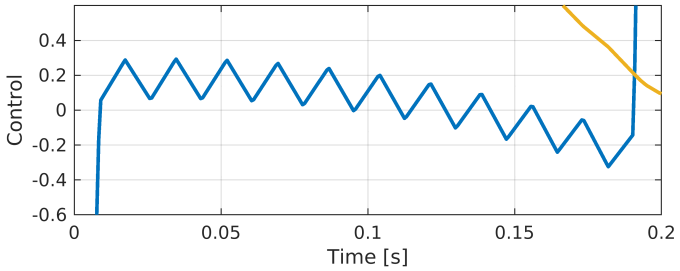
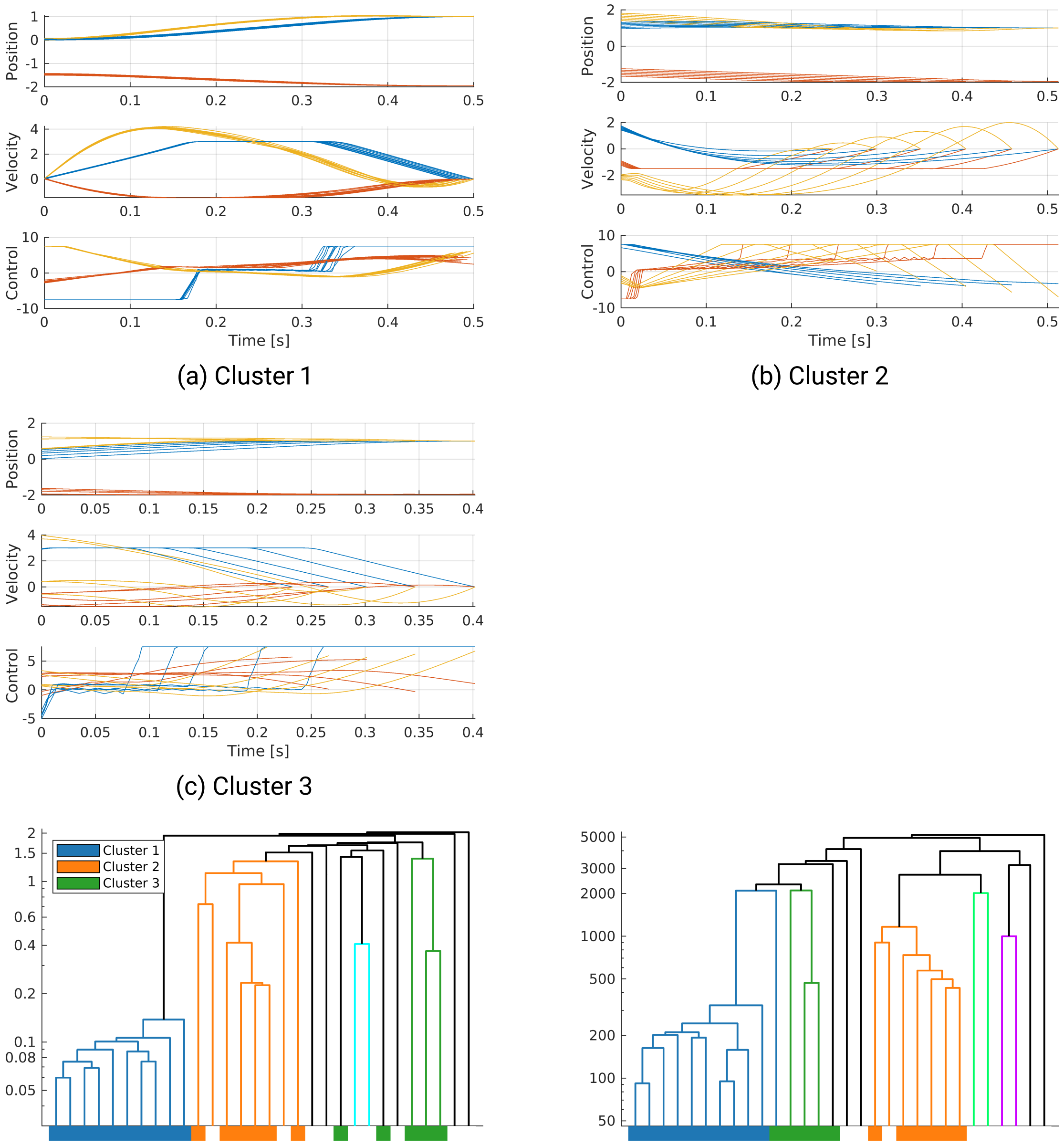


Figure 6.11.: Example: Enlarged view of a motion plan showing the zig-zag path of one dimension of the control trajectory around zero.



(a) Cluster 1

(b) Cluster 2

(c) Cluster 3

(d) Result with Feature-based

(e) Result with DTW

Figure 6.12.: Clustering result of the *Manutec* test set. Subfigs. 6.12a to 6.12c give the three largest ground truth clusters. They show the state and control trajectories for three joints over time. The two dendrograms illustrate the hierarchical structure of the clusters resulting from the different distance measures. The ground truth for the three largest clusters is given by the colors on the x-axis.

from the third cluster are incorrectly assigned to the first cluster). The second cluster includes one additional motion plan. Comparing the dendrograms (Figure 6.12), the diagram for DTW seems to be more clearly structured. The first cluster is presented contiguously, but a very low cutoff value is needed to separate it from other elements. In the dendrogram for the presented approach, this cluster is also clearly recognizable: It is separated from other motion plans, even for a very high cutoff value. Cluster 1 stands out visually from the other motion plans, it is clearly identified by the feature-based approach, whereas DTW has major problems with this cluster, as can be seen in the dendrogram.

To summarize, both methods have some difficulties with the selected data set, which is reflected in the dendrograms. In several cases, the motion plans have similar progressions in single trajectories, which makes clustering challenging even for human experts. Interestingly, the two approaches identify different clusters well.

6.3.3. Clustering of a Real-world Human Motion Dataset

To evaluate the performance of the presented method on real-world data, trajectories from the human locomotion dataset created by Reznick et al. [155] are clustered. They recorded data from ten participants walking, running, stair climbing and sitting down/standing up at various speeds and inclinations. The recorded data is post-processed (filtered and gaps in the data filled). In this chapter, the normalized dataset is used where the strides in the recorded trajectories are separated and the time is normalized to the interval $[0, 1]$ and interpolated to 150 data points (for details, see [155]). In this evaluation, the provided joint angle trajectories for the ankle, knee, hip, pelvis and foot progression are used; and from each of them only the x-component, which is the most expressive and thus provides the best results. Consequently, the state trajectories to be clustered have five dimensions. There is no control signal information, the joint limits are not known and roots have no meaning, such that only the state trajectory with maxima and minima as features is used. Extrema with prominence lower than 0.2 are removed from the feature sequence.

A cluster consists of the first stride of all ten participants in a single motion task. The distinctness of the different motions in the data set varies, such that the difficulty of the clustering task varies with the selection of motions. For example, both clustering methods fail to reliably differentiating between joint trajectories for walking with different inclinations; but this task is also difficult for humans. In the test set *Human Motion 1*, the following motion types, which are relatively easy to differentiate, are to be clustered: (1) running at $1.8\,\mathrm{m/s}$, (2) descending a 35° stair, (3) walking 10° downhill at $0.8\,\mathrm{m/s}$

GT	Feature-based (8)					DTW (13 clusters)					DTW (4 clusters)			
	C_1	C_2	C_3	C_4	...	C_1	C_2	C_3	C_4	...	C_1	C_2	C_3	C_4
C_1	**10**					**8**				2	**0**		10	
C_2		**9**			1		**6**			4		**10**		
C_3			**9**		1			**9**		1			**10**	
C_4				**8**	2				**7**	3	1			**9**

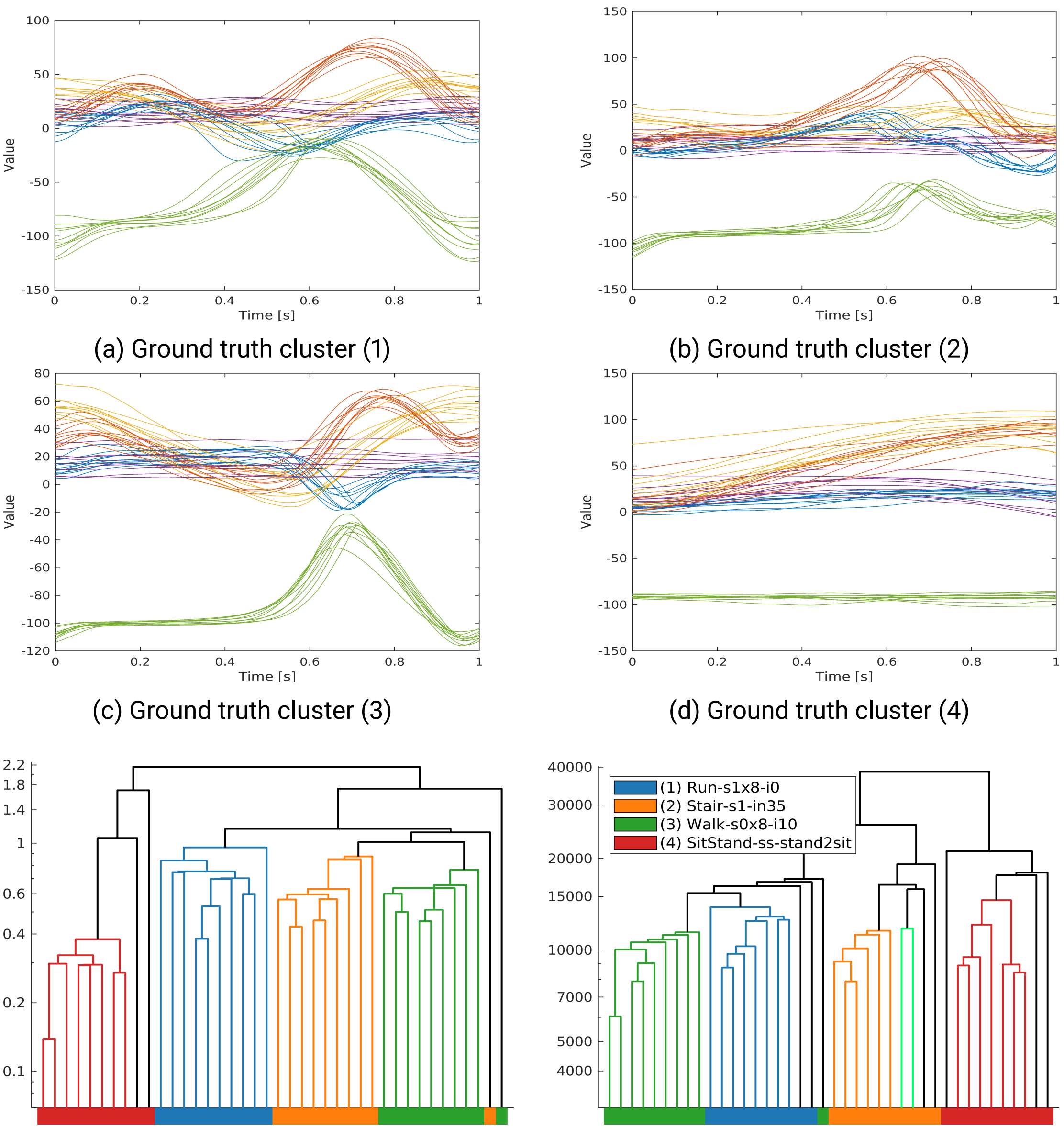
(a) Confusion matrices for dataset *Human Motion 1*

GT	Feature-bd. (13)					Feature-bd. (10)					DTW (13)					DTW (8)				
	C_1	C_2	C_3	C_4	...	C_1	C_2	C_3	C_4	...	C_1	C_2	C_3	C_4	...	C_1	C_2	C_3	C_4	...
C_1	**8**				2	**9**				1	**7**		2		1	**10**				
C_2		**10**					**10**					**8**			2	1	**8**			1
C_3	1		**4**		5	5	1	**1**		3			**8**		2	9		**1**		
C_4				**8**	2				**8**	2				**4**	6				**7**	3

(b) Confusion matrices for dataset *Human Motion 2*

Table 6.4.: The clusterings of the Human motion dataset [155] as confusion matrices using the feature-based approach and DTW. The rows give the ground truth clusters, the columns the respective cluster assignments resulting from the two distance measures. The number of clusters is given as number behind the method name; two separations into clusters are given if it is unclear which one is best. Additional clusters are summarized in "...". The number of correctly classified motion plans is given on the diagonals.

and (4) sitting down on a chair. The results of the clustering using DTW and the feature-based approach are presented in Figures 6.13e to 6.13f. By and large, all four clusters can be identified in the dendrograms. The feature-based approach has a problem to distinguish between the clusters (2) and (3), DTW to differentiate between (1) and (3). Both approaches report high distance values for some trajectories of a motion type and thus separate them from the others, resulting in more than four clusters. As before, the separation into the most favorable number of clusters for the feature-based approach and for DTW are reported in Table 6.4a, and additionally two alternative separations into 13 and 4 clusters that are equally well. It can be seen that the four ground truth motions are identified with much higher cutoff values using the feature-based approach, and also the number of misclassifications is substantially lower.



(a) Ground truth cluster (1)

(b) Ground truth cluster (2)

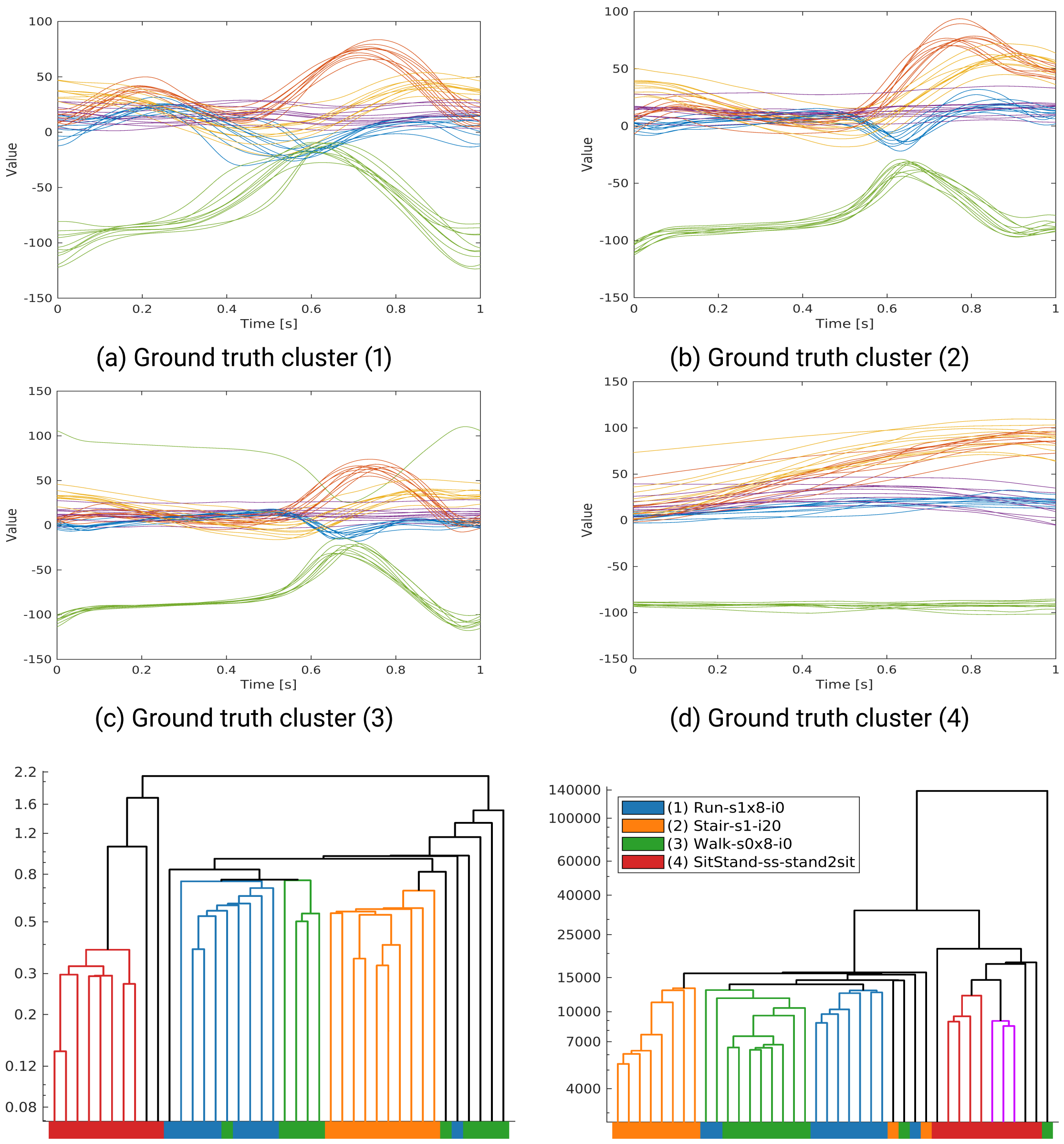
(c) Ground truth cluster (3)

(d) Ground truth cluster (4)

(e) *Human Motion 1*: Result with Feature-based

(f) *Human Motion 1*: Result with DTW

Figure 6.13.: Clustering result of the *Human Motion 1* test set using different distance measures. Subfigs. 6.13a to 6.13d give the four ground truth clusters. They show the x-component of the joint angles over time. The two dendrograms illustrate the hierarchical structure of the clusters resulting from the different distance measures. The ground truth is given as colored bar on the x-axis.



(a) Ground truth cluster (1)

(b) Ground truth cluster (2)

(c) Ground truth cluster (3)

(d) Ground truth cluster (4)

(e) *Human Motion 2*: Result with Feature-based

(f) *Human Motion 2*: Result with DTW

Figure 6.14.: Clustering result of the *Human Motion 2* test set using different distance measures. Subfigs. 6.14a to 6.14d give the four ground truth clusters. They show the x-component of the joint angles over time. The two dendrograms illustrate the hierarchical structure of the clusters resulting from the different distance measures. The ground truth is given as colored bar on the x-axis.

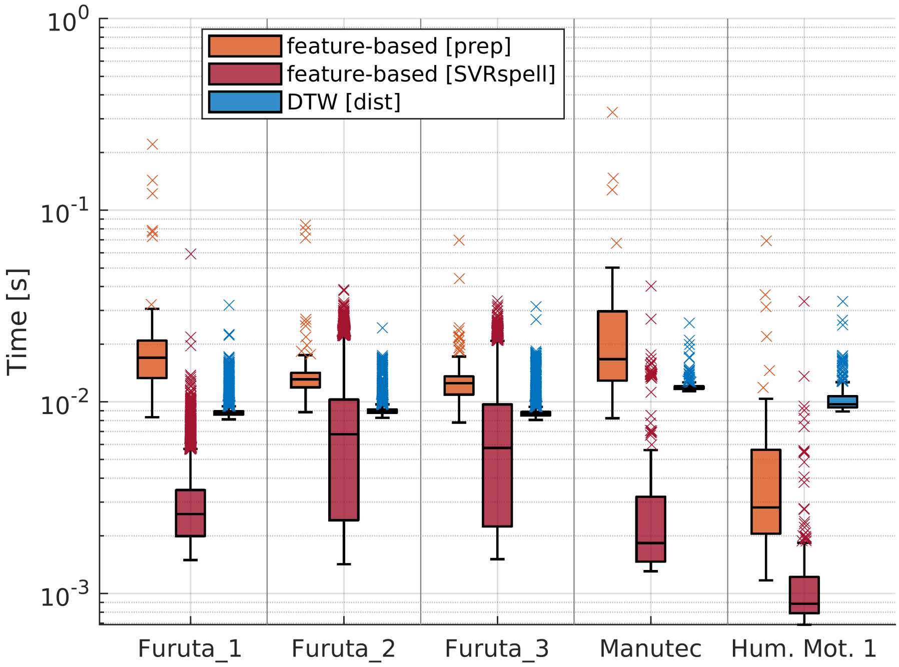


Figure 6.15.: Computation time for a single distance computation between two trajectories on the test sets considered here. The computation time of the feature-based approach is splitted into the precomputation step and the *SVRspell* distance computation.

The motion types of *Human Motion 2* are more difficult to differentiate: (1) running at $1.8\,\mathrm{m/s}$, (2) ascending a 20° stair, (3) walking at $0.8\,\mathrm{m/s}$ and (4) sitting down on a chair. The results are presented in Figures 6.14e to 6.14f. Here, the ground truth clusters are more difficult to distinguish in both dendrograms. The feature-based approach needs a separation into 13 clusters to distinguish four larger clusters, DTW also needs 13 clusters to distinguish between cluster (1) and (3). The results for different numbers of clusters are given in Table 6.4b. The performance of both approaches is comparable on this more difficult test set; both reveal difficulties in differentiating between cluster (1) and cluster (3) in particular.

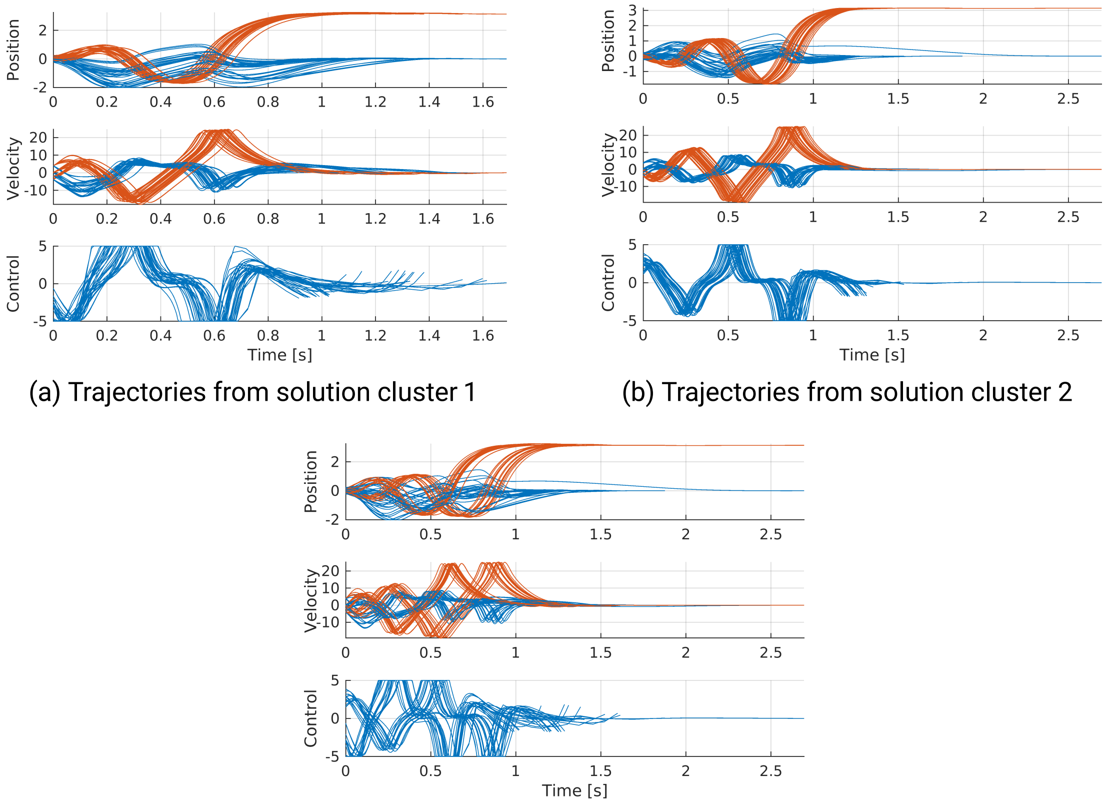
6.3.4. Evaluation of Efforts and Runtime

The runtime analysis of the feature-based distance computation can be divided into the feature extraction step and the *SVRspell*-based distance computation. The feature extraction is a preprocessing step that must be done only once for all trajectories, independent of the number of distance measurements required for the clustering. Its runtime depends on the number of feature classes and the algorithms employed to identify the features of a trajectory. The complexity of the distance computation (based on the SVRspell algorithm) is cubic [54] in the *number of features*. The number of features is typically much smaller than the number of time steps, on which the complexity of DTW depends ($\mathcal{O}(NM)$ with N, M the trajectories' number of time steps). The runtime of the *SVRspell*-based distance computation is important, as the number of calls to the distance computation is quadratic or even cubic in the number of trajectories for many clustering approaches and in particular for the classical agglomerative hierarchical clustering algorithm [127]. Figure 6.15 shows the time to extract the feature sequence (orange) as well as the times to compute the distance between two trajectories using DTW (blue) and feature-based approach (red). It must be noted that for this evaluation, the DTW and SVRspell implementations run as compiled code while the feature extraction is implemented in plain MATLAB. Thus, improved performance can be expected if the code for the preprocessing step is transferred to a compiled language. The distance measure based on the feature sequence representation thus has potential advantages regarding computation time.

6.4. Experiments on the Effect of Mixed Clusters on Learned Control Policies

It remains to be evaluated how mixing several clusters affects the approximated feedback control. A subset of a large set of solution trajectories from different start states that solve the Furuta pendulum problem (see Section 5.2.4) is subdivided into two solution clusters. Both clusters consist of 36 optimal trajectories, the distribution of start states in the joint space is approximately the same. Apart from these two sets of trajectories *Cluster 1* and *Cluster 2*, a third trajectory set *Cluster Mix* is constructed by combining 18 trajectories from each. The graphs of the three trajectory sets are presented in Figure 6.16.

Taking 130 samples from each trajectory, the training data for the feedback control approximation consists of 4680 samples for each set. Multiple GPs with different kernels are trained, all using FITC with 100 inducing points. The resulting feedback control policy



(a) Trajectories from solution cluster 1

(b) Trajectories from solution cluster 2

(c) Trajectories from both solution clusters

Figure 6.16.: The three sets of 36 trajectories, two from a single solution cluster and one combining trajectories from both clusters.

approximations are combined with a stabilizing PI controller (with the same parameters and switch conditions as in Section 5.4) and used on a real-world Furuta pendulum. Apart from evaluating if the pendulum can be balanced, the maximum arm deflection required by the PI controller to stabilize the system around the goal state, as introduced in Section 5.4, is again used as a non-binary performance criterion. The arm deflection values for the different controllers are reported in Table 6.5. It is noticeable that the error rate of the control trained with the data of the second cluster is significantly higher than that of the control trained with the data from the first cluster. The feedback controllers trained with the first cluster all successfully reach the goal state, but using data from the second cluster, only the GP controller with Gibbs kernel succeeds. As already observed, the training of

	ARD	Gibbs	Matérn-3/2	MLP	Sqexp
Cluster 1	0.6105	0.9296	1.3131	0.4295	0.6903
Cluster 2	NaN	0.6075	2.0739	2.0862	NaN
Mixed	2.0893	1.8070	NaN	1.6858	2.0862

Table 6.5.: Result for execution on real-world Furuta pendulum using different GP kernels on pure and mixed cluster data.

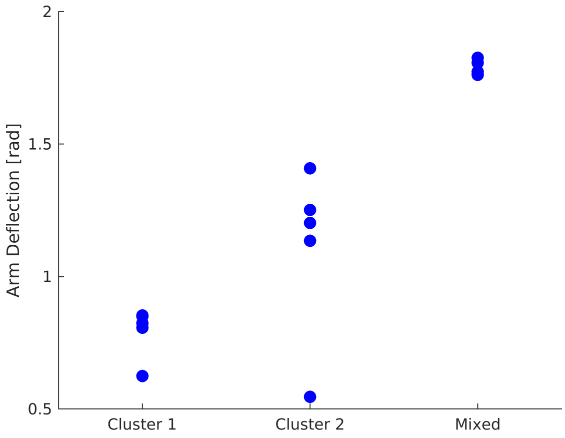


Figure 6.17.: Maximum arm deflections for repeated execution on the real-world Furuta pendulum using the Gibbs kernel on pure and mixed cluster data.

GPs fails in some cases, and some kernels are more susceptible to failure than others. Nevertheless, it is noteworthy that almost all kernels fail for the second cluster. One possible explanation is that the second solution cluster is more difficult to approximate than the first, since it contains an additional swing-up. The Gibbs kernel GP provides the most interesting results, as this controller successfully balances the pendulum for all three clusters. For the pure clusters *Cluster 1* and *Cluster 2*, the learned controllers provide very good results, as the lash required by the PI controller to balance the system is very small, with 0.93 for the first and 0.61 for the second cluster. However, learning a controller on a combination of trajectories from both clusters leads to severely degraded results: The PI controller generates a deflection of the pendulum arm of 1.81 rad to balance the system and almost violates the state constraint.

The experiment is repeated four times using the Gibbs kernel controller trained on cluster

1, cluster 2, and the mixed cluster to validate this result. Due to minor variations in the initial state, every execution is slightly different, even for the same controller. The results are presented in Figure 6.17. The data indicates that the very small arm deflection value for the controller trained on cluster 2, as reported in Table 6.5, is in fact an outlier and is typically larger than for the controller trained on cluster 1. However, all arm deflections required to stabilize the mixed cluster controller are notably larger than those for the pure cluster controllers. This shows clearly that mixing trajectories from different clusters to train a single feedback controller reduces the achievable performance. It is therefore desirable to differentiate between the different clusters provided by DIRCOL in the iterative extremal field approach and to carefully select, which trajectories are used in the iterative learning procedure.

6.5. Discussion and Conclusion

The number of features has the largest impact on the runtime complexity of the new distance measure, which effectively decouples the computation of the distance from the time length of a trajectory. This is an advantage if long trajectories can be reduced to a small number of features. The number of features per trajectory depends on the trajectory itself, the feature classes used and the parameterization (thresholds). It may be necessary to preprocess trajectories representing observation data to remove outliers and noise. A suitable salience threshold, as used in this chapter, is often sufficient to keep the feature sequence representations sufficiently small. A maximum length of the feature sequence can be enforced by sorting all features in a trajectory by its salience value and keeping only the features with the highest values. The dimensionality of the trajectories' co-domains affects the runtime of the distance measure only linearly, since the distances of the trajectories' dimensions are summed up (see Equation (6.8)).

An advantage of the feature-based distance measure is the flexibility in the selection of arbitrary many and arbitrary complex feature classes, which allows adaptation to specific problems and use cases. The parameters of the presented method are feature class dependent; typical parameters are, for example, salience thresholds. It must be noted that the distance measure needs a sufficient number of distinct features in the trajectory graph to show its potential.

In this chapter, a new approach is developed to compute the distance between trajectories representing motion plans that can be used for clustering. It is based on a method presented by Elzinga et al. [52] for measuring the distances between strings. To make this method applicable, an approach is presented to compress raw trajectories into sequences of user-defined high-level semantic features, such as extrema or roots, that characterize the general shape of the graph. These features are complemented with information about their temporal position and their salience in the trajectory. Elzinga's sub-sequence-based distance metric [54] is adopted to work with this feature sequence representation of trajectories augmented with time and salience information. The resulting distance measure for trajectories is used in agglomerative hierarchical clustering to identify trajectories with similarly shaped graphs.

The presented distance measure can be applied to all sorts of trajectories, independent of the application. In the field of human-robot interactions, further potential of the presented distance measure lies in the construction of hierarchical motion databases (as done, e.g., in [207, 95]), since the compressed feature-based representation of motions is memory efficient and allows fast comparisons.

Since the agglomerative hierarchical clustering approach needs to compute all pairwise distances of a set of trajectories, the size of trajectory sets that can be clustered is limited, since the required computational effort increases quadratically. For larger sets of trajectories, alternatives to agglomerative hierarchical clustering must be considered that do not require the computation of distances between all trajectory pairs.

Further work is needed to optimize the distance measure for online use of streamed trajectories. This would require reusing computations of Elzinga's distance measure when new elements are successively added to one of the feature sequences. The feature extraction must be performed on each update, which is possible with the current code. Further work is also needed to enable the distance measure to split a long recorded motion sequence into subsequences of already clustered motions.

Clustering using the novel feature-based distance measure has been compared with clustering based on the often used DTW algorithm [158, 128, 195]. The data on which the clustering is evaluated encompasses optimal trajectories provided by DIRCOL and real-world human locomotion data. The sets of motion plans generated by the optimal control solver describe movements of two robotic systems: the Furuta pendulum and the Manutec robot arm. Further, the real-world data test sets consist of joint motion data of humans performing different locomotion tasks that has been generated by Reznick et al. [155]. The results show a reliable performance that is at least on a par with the proven DTW, in some cases it even outperforms this commonly used distance measure. An advantage of the presented method is flexibility in the choice of feature classes, which allows adaptation on specific problems and use cases.

In the iterative extremal field approach, the data from several different optimal trajectories are used to train the near-optimal feedback controller. For some problems, such as the Furuta pendulum swing-up task described in Section 5.2.4, multiple locally optimal solutions with approximately the same cost but different characteristics exist. Trajectories from different start states with similar characteristic shapes are said to be in the same cluster. Experiments on the real-world Quanser Furuta pendulum have shown that using trajectories from different solution clusters to train the same feedback control policy can lead to a noticeable degradation in its performance. To avoid this, clustering can be used to reject locally optimal trajectories from all but one solution cluster.

The code implemented for this chapter (apart from Section 6.4) and the data used for evaluation are available on Github[1].

This chapter is based on the paper "Clustering of Motion Trajectories by a Distance Measure Based on Semantic Features", published in 2023 IEEE International Conference on Humanoid Robots (Humanoids) *[212]. It contains the following major additions: illustrative Figures 6.2, 6.3, 6.6, 6.7 and 6.11, extended Section 6.2.1, illustrations of the clusters in Figures 6.8 to 6.10 and 6.12 to 6.14, additional Section 6.4.*

[1] `https://github.com/cztuda/semantic-feature-clustering`

7. Conclusion

The optimal control of robotic systems offers significant performance advantages. Optimal feedback control policies for robotic systems allow to optimally solve a specific task not only on a precomputed trajectory from the current system state but in a whole region, which is necessary to deal with external perturbations or modeling errors. Unfortunately, the approximation of near-optimal feedback control policies for general high-dimensional, nonlinear robotic systems is hard. The extremal field approach pursued in this thesis allows to leverage information provided by numerical optimal control solvers. Data from a set of optimal trajectories is combined to learn a near-optimal approximation of the feedback control. The application of this approach to modern robotic systems is challenging, e.g., due to their high-dimensionality and nonlinearity of their dynamics. Iteratively adding training data from new optimal trajectories helps to restrict the coverage of the joint space to the part that is relevant for a specific task, as information from the current control approximation and previously computed trajectories can be used.

7.1. Contribution

In this thesis, several contributions to the field of near-optimal feedback control approximation for nonlinear dynamic robotic systems using the extremal field approach are made. They are reviewed in the following paragraphs.

Start State Selection Strategies

To reduce the effect of the curse of dimensionality, the extremal field approach needs to restrict the optimal trajectories that provide the training data to a subset of the joint space that is large enough such that the state of the robot system does not leave this subset in case of perturbations or modeling errors and, at the same time, small enough to be well covered by training data.

The presented approach selects the start states for the computation of optimal trajectories that provide the training data not all at once but iteratively. This has the advantage that in each iteration, information from the current feedback control approximation and previously computed trajectories can be used to focus on regions where training data is missing to ensure a well-balanced coverage of the relevant state space.

In this work, three novel start state selection methods are introduced to ensure that critical regions of the state space are well covered. They are based on the adjoint variable provided by the numerical optimal control solver based on direct collocation, on an approximation of the normed partial sensitivity and on an increase in the error along the trajectory when simulated with the current policy approximation.

The adjoint variable approach has been compared to the naive sampling approach that is used in many works, e.g., in [217, 120]. It has been demonstrated on the Manutec r3 robot arm that the policy trained using the adjoint-based start state selection method produces trajectories that get closer to the goal state in shorter time, and thus with a lower cost value.

The proposed selection methods are not intended to be used separately but combined to complement each other, together with random sampling around the nominal start state. The combination of the four different start state selection methods brings a substantial improvement of the results compared to those achieved with only the adjoint-based strategy. It has been demonstrated on the Manutec r3 robot arm, that, compared with local LHS sampling as used in [64], the combination of the four methods produce feedback control policy approximations with an overall higher success rate and shorter minimum distance to the goal state. This illustrates the advantage of a mixed selection strategy to cover the joint space with optimal trajectories.

The motivation to use near-optimal feedback control policies is their higher robustness to perturbations compared to open-loop control. The learned controller of the Manutec r3 arm shows decent performance when subjected to abrupt changes in the state or velocity during the execution of the control policy (intermittent disturbances) and perturbed system dynamics using different friction models (continuous nonlinear disturbances). This makes the presented approach applicable to problems where moderate perturbations can be expected.

To improve the convergence of the system state to the goal state, it has been proposed to pass control to a traditional PI controller that is tuned with an LQR approach to stabilize the system around the goal state. This removes the need for a large amount of training data around the goal, since the learned control policy only has to get the system state close to the goal state. Compared to the trajectory sampling around the goal state, the switch-over to the LQR controller significantly improves the final distance to the goal state. Using only a small number of short trajectories to keep the amount of training data small, a convergence to the goal state can not be achieved with the standard approach. However, the switch-over approach is able to stabilize the system at the goal state while using the same number of training samples. However, while the cost values of these trajectories are only slightly increased, the time to reach the goal state is significantly longer than before.

To analyze how the learned feedback control policy behaves in the presence of external perturbations or model inaccuracies, several simulations have been performed in MATLAB using state-of-the-art numerical solvers for ordinary differential equations.

Approximation Methods

Different methods for function approximation in the extremal field approach have been considered in this work. Several requirements on the approximation method to evaluate their suitability to model near-optimal feedback control policies have been formulated. Two widely used approximation methods, GPs and NNs, have been analyzed and compared based on these requirements on the Furuta pendulum and the Manutec r3 robot arm. This work contributes insights into the usability of two extensions to improve the suitability of NNs and GPs, into adequate hyperparameter selection, and into the achievable prediction accuracy and performance.

Monte-Carlo dropout can be used to get uncertainty estimates for the predictions of NNs. This has a low computational overhead, but complicates the training and decreases the prediction accuracy of the feedback control approximation. Global approximations using sparse GPs, which allow their use on large datasets, seem to have only a small impact on the accuracy.

For GPs, it turned out that MLP, Gibbs and Sqexp kernels are particularly suitable to approximate the highly nonlinear feedback control policy. Regarding fully connected feedforward NNs, the performed evaluations shows that a few (4 to 10) layers are sufficient to get a satisfactory accuracy. The number of nodes per layers seems to have a slightly larger impact on accuracy than the number of layers. The evaluation results indicate that it is not necessary to use more than a few hundred nodes per layer.

When large amounts of training data are available, NNs clearly outperform GPs, their accuracy is comparable on very small datasets with only a few thousand samples. Regarding prediction time, the use of NNs provides no advantage compared to GPs: Exact

GPs for small and sparse GPs approximations for large amounts of training data provide predictions faster than the tested NNs of different sizes.

In experiments on a real-world Furuta pendulum, NNs showed a slightly better performance than GPs with carefully selected kernel functions. However, when the system dynamics were substantially perturbed by weights on the pendulum arm, the GPs performed similar to the NNs or even outperformed them. This indicates that GPs can generalize better to unseen data than NNs, such that the use of GPs is favorable if large disturbances are expected.

Trajectory Distance Measure

For some problems, such as the Furuta pendulum swing-up task considered in this thesis, the optimal control solver provides multiple locally optimal solutions with approximately the same cost but different characteristics. Experiments on the Quanser Furuta pendulum have demonstrated that combining training data from different solution clusters to train a GP that represents a feedback control approximation has a negative impact on its performance on real systems. These results are consistent with the findings of Merkt et al. [117]. This insight motivates the proposal of a novel distance measure for trajectory clustering that creates a compressed representation of the trajectories to be clustered and uses a string kernel method by Elzinga [52] to compute the distances. The goal is to create an intuitively interpretable rule-based method (in contrast to black-box approaches based on NNs), where the focus is on salient characteristic features in the trajectory graphs instead of pointwise Euclidean distances between trajectories. Another advantage of the feature-based approach is that the feature-classes that determine the distance measure can be task-specifically designed. The feature-based distance measure has been compared with the widely used DTW approach on optimal trajectories computed with direct collocation for the Furuta pendulum and the Manutec r3 robot arm and on human locomotion data recorded using motion capturing by Reznick et al. [155]. Overall, the proposed distance measure provides slightly better clustering results than the widely used DTW approach. For some problems, the proposed distance measure is expected to show strong advantages in runtime, as the trajectory distance computation is decoupled from the trajectories' time length. The use of the distance measure for trajectory clustering is not restricted to the optimal trajectories used in the extremal field approach.

7.2. Directions for Further Work

The extremal field approach offers a way to utilize information from numerical optimal control solvers together with sophisticated state-of-the-art learning approaches and combine the advantages from both fields to approximate a near-optimal feedback control policy. Nevertheless, there also remain several challenges that need to be addressed in further work to improve the applicability of this approach. Two of them will be outlined in the following.

The training data for the extremal field approach is provided by optimal control solvers that compute the trajectories based on a detailed nonlinear physical dynamics model of the robot and its environment. Consequently, modeling errors and inaccuracies are thus present in every data point that is used to learn the control policy. Reinforcement learning approaches reduce model errors by repeated corrections during the various rollouts on the real system. In the extremal field approach, each correction of the dynamic model requires the recomputation or correction of all optimal trajectories computed so far. This is sufficient for nearest-neighbor approximations of the feedback control policy (e.g., Atkeson and colleagues [10]). However, if these optimal trajectories are used to train a parametric or non-parametric model of the feedback controller (as in this work, where a GP or a NN is trained), this training needs to be redone as every sample in the training set changes. Some existing approaches use parameterization of the dynamic model to explicitly learn different dynamics (e.g., Schierman et al. [161]), which obviously increases the dimensionality of the problem and with that the amount of the required training data. It may also be beneficial to adapt from the work done in the context of sim-to-real [125, 216] for reinforcement learning approaches, or to consider stochastic system dynamics as done in some RL approaches [144, 93, 102] and also for NMPC [118, 98]. In conclusion, the efficient handling of modeling errors in the extremal field approach is still an interesting field of research.

Another open field of research is the derivation of estimates or even guarantees on the suboptimality of the computed feedback control policy and its region of stability. This aspect is important to increase the reliability of the extremal field approach and enable its use in safety-critical applications. Such guarantees are difficult to formulate since the extremal field approach as presented in this thesis encompasses the numerical computation of locally optimal trajectories and the training of a control policy approximation. Numerical trajectory optimization methods may produce locally optimal results that are globally suboptimal. For GPs or NNs, whose performance depends on the available training data and the optimization routine used for training, it is notoriously difficult to specify error bounds. Khadke and Geyer [89] estimate suboptimality in their approach but cannot provide guarantees for some task-relevant part of the joint space.

In summary, the extremal field approach offers many advantages when used to approximate feedback control policies, and the results obtained so far look promising. This is reflected in the interest of the robotics community as well as, for example, the aerospace community in these methods. However, considering the open research questions outlined above, it is clear that there is still a long way to go to achieve efficient and reliable feedback control policies that enable complex tasks to be performed near-optimally by sophisticated robotic systems. This thesis is a step in that direction.

A. Appendix

A.1. The Gaussian Process Kernel Functions

The kernel functions used in this thesis are defined as implemented in the *GPmat* toolbox [3]. They can also be found in [154], but sometimes the notation is slightly different. For example, the *compound* kernel of GPmat is denoted by *kernel sum* in Rasmussen and Williams [154]. The scalar learnable, kernel-specific parameters are σ, σ_{SE}, σ_{B}, σ_{L}, σ_{N}, σ_w, σ_b and σ_ℓ. The learnable length-scale matrices are Σ_{SE} and Σ_{L}. The symbol $\mathbb{1}$ denotes a matrix where all entries are ones, and $\tilde{I}$ is a diagonal matrix where the i-th diagonal element is one if $x_i = x'_i$ and else zero.

The (anisotropic) squared exponential (SE) kernel function is defined as

$$k(x, x') = \sigma_{\text{SE}} \cdot \exp\left(-\frac{1}{2} \cdot (x - x')^T \Sigma_{\text{SE}} (x - x')\right), \tag{A.1}$$

the MLP kernel function is defined as

$$k(x, x') = \frac{2\sigma}{\pi} \sin^{-1}\left(\frac{\sigma_w x^T x' + \sigma_b}{\sqrt{\left(\sigma_w x^T x + \sigma_b + 1\right)\left(\sigma_w x'^T x' + \sigma_b + 1\right)}}\right) \tag{A.2}$$

and the Matérn-3/2 kernel function as

$$k(x, x') = \sigma \cdot \left(1 + \frac{\sqrt{3}\,(x - x')}{\sigma_\ell}\right) \exp\left(-\frac{\sqrt{3}\,(x - x')}{\sigma_\ell}\right). \tag{A.3}$$

The Gibbs kernel function for n-dimensional input vectors is given by

$$k(x, x') = \sigma \cdot \left(\frac{2\ell(x)\ell(x')}{\ell(x)^2 + \ell(x')^2}\right)^{\frac{n}{2}} \exp\left(\frac{-(x - x')^2}{\ell(x)^2 + \ell(x')^2}\right) \tag{A.4}$$

with $\ell(x) := \exp\left(k_{\text{MLP}}(x,x)\right)$.

The two compound kernels are the ARD kernel function, defined as

$$k(x,x') = \underbrace{\sigma_{\text{SE}} \cdot \exp\left(-\frac{\sigma_w}{2} \cdot (x-x')^T \Sigma_{\text{SE}} (x-x')\right)}_{\text{anisotropic squared exponential}} + \underbrace{\sigma_{\text{B}} \cdot \mathbb{1}}_{\text{bias}} + \underbrace{\sigma_{\text{L}} \cdot x^T \Sigma_{\text{L}} x'}_{\text{linear}} + \underbrace{\sigma_{\text{N}} \cdot \tilde{I}}_{\text{Gaussian noise}} \tag{A.5}$$

and the Sqexp kernel function, given by

$$k(x,x') = \underbrace{\sigma_{\text{SE}} \cdot \exp\left(-\frac{\sigma_w}{2} \cdot (x-x')^T (x-x')\right)}_{\text{squared exponential}} + \underbrace{\sigma_{\text{B}} \cdot \mathbb{1}}_{\text{bias}} + \underbrace{\sigma_{\text{N}} \cdot \tilde{I}}_{\text{Gaussian noise}} \tag{A.6}$$

A.2. Selection of Numerical Trajectory Optimization Software

Name	Ref.	Method	Lang.	License	Comment
PSOPT	[16]	direct local and orthog. collocation	C++	LGPL	solves using either pseudospectral or local approximations
Bocop	[27]	direct local collocation	C++	EPL	provides a user interface for efficient use
Horizon	[157]	direct collocation and multiple shooting	Python	LGPL	focus on robotic problems (e.g., parse URDF files)
DIRCOL	[174]	direct local collocation	Fortran	custom	for education and non-profit research purposes thesis
GPOPS II	[140]	direct orthog. collocation	MATLAB	comm.	commercial successor of the once freely available GPOPS
CGPOPS	[2]	direct orthog. collocation	C++	comm.	claimed to be faster than GPOPS II
Altro	[80]	modified iLQR with augmented Lagrangian	Julia	MIT Lic.	the only DDP-based method in this list
ICLOCS 2	[57]	direct local and orthog. collocation and multiple shooting	MATLAB	MIT Lic.	wide range of methods and solvers in one framework
ACADOS	[196]	direct multiple shooting	C	2-C BSD	successor of ACADO, focus is on real-time capabilities and model predictive control
DIDO	[156]	direct orthog. collocation	MATLAB	comm.	optimal control toolbox
OTIS	[138]	direct orthog. collocation	Fortran	restricted	restricted to users within the United States who are working for the Government
SOCS	[23]	direct transcription	?	comm.	developed at Boeing, seems to be no longer available
OptimTraj	[87]	direct local and orthog. collocation and multiple shooting	MATLAB	MIT Lic.	developed by Matthew Kelly while working on his PhD
SNCTRL	[68]	direct local collocation	Fortran	MIT Lic.	SNOPT interface to solve OCPs, no path constraints

Table A.1.: Selection of optimal control solver software, with focus on direct methods. For a more extensive "Survey of Numerical Methods for Solving Optimal Control Problems", see Putkaradze and Vakhtang [149]

Bibliography

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. “A review of uncertainty quantification in deep learning: Techniques, applications and challenges”. In: *Information Fusion* 76 (2021), pp. 243–297.

[2] Yunus M. Agamawi and Anil V. Rao. “CGPOPS: A C++ Software for Solving Multiple-Phase Optimal Control Problems Using Adaptive Gaussian Quadrature Collocation and Sparse Nonlinear Programming”. In: *ACM Transactions on Mathematical Software* 46.3 (2020), pp. 1–38.

[3] Neil Lawrence et al. *MATLAB GPmat Toolbox*. University of Sheffield. 2015. URL: `https://github.com/SheffieldML/GPmat`.

[4] Brian Armstrong-Hélouvry, Pierre E. Dupont, and Carlos C. De Wit. “A survey of models, analysis tools and compensation methods for the control of machines with friction”. In: *Automatica* 30.7 (1994), pp. 1083–1138.

[5] Karl Johan Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[6] Christopher G. Atkeson. “Nonparametric Model-Based Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. MIT Press, 1997, pp. 1008–1014.

[7] Christopher G. Atkeson. “Using Local Trajectory Optimizers To Speed Up Global Optimization In Dynamic Programming”. In: *Advances in Neural Information Processing Systems*. Morgan Kaufmann Publishers, 1993, pp. 663–670.

[8] Christopher G. Atkeson and Chenggang Liu. “Trajectory-Based Dynamic Programming”. In: *Modeling, Simulation and Optimization of Bipedal Walking*. Ed. by K. Mombaur and K. Berns. Springer, 2013, pp. 1–15.

[9] Christopher G. Atkeson and Jun Morimoto. “Nonparametric Representation of Policies and Value Functions: A Trajectory-Based Approach”. In: *Advances in Neural Information Processing Systems*. MIT Press, 2002, pp. 1643–1650.

[10] Christopher G. Atkeson and Benjamin J. Stephens. “Random Sampling of States in Dynamic Programming”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.4 (2008), pp. 924–929.

[11] Adrien Baranes and Pierre-Yves Oudeyer. “Active learning of inverse models with intrinsically motivated goal exploration in robots”. In: *Robotics and Autonomous Systems* 61.1 (2013), pp. 49–73.

[12] Fransiska Basoeki, Fabio Dalla Libera, Emanuele Menegatti, Enrico Pagello, and Hiroshi Ishiguro. “Clustering of Humanoid Robot Motions Executed in Response to Touch”. In: *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*. Springer, 2016, pp. 1063–1076.

[13] Guillaume Bastille-Rousseau and George Wittemyer. “Characterizing the landscape of movement to identify critical wildlife habitat and corridors”. In: *Conservation Biology* 35.1 (2021), pp. 346–359.

[14] Randal W. Beard, George N. Saridis, and John T. Wen. “Galerkin approximations of the generalized Hamilton-Jacobi-Bellman equation”. In: *Automatica* 33.12 (1997), pp. 2159–2177.

[15] Alex Beaudin and Hsiu-Chin Lin. *Learning Agile Paths from Optimal Control*. 2022. arXiv: 2212.00184 [cs.RO].

[16] Victor M. Becerra. “Solving complex optimal control problems at no cost with PSOPT”. In: *2010 IEEE International Symposium on Computer-Aided Control System Design*. IEEE, 2010, pp. 1391–1396.

[17] Scott C. Beeler, Hien T. Tran, and Harvey T. Banks. “Feedback Control Methodologies for Nonlinear Systems”. In: *Journal of Optimization Theory and Applications* 107 (2000), pp. 1–33.

[18] Richard E. Bellman. *Dynamic Programming*. Reprinted by Dover Publications, Inc., Mineola, New York (2003). Princeton University Press, 1957.

[19] Kristin P. Bennett and Emilio Parrado-Hernández. “The Interplay of Optimization and Machine Learning Research”. In: *Journal of Machine Learning Research* 7.46 (2006), pp. 1265–1281.

[20] David A. Benson, Geoffrey T. Huntington, Tom P. Thorvaldsen, and Anil V. Rao. "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method". In: *Journal of Guidance, Control, and Dynamics* 29.6 (2006), pp. 1435–1440.

[21] John T. Betts. "A Survey of Numerical Methods for Trajectory Optimization". In: *Journal of Guidance, Control, and Dynamics* 21.2 (1998), pp. 193–207.

[22] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming, Third Edition*. SIAM, 2020.

[23] John T. Betts and William P. Huffman. "Mesh refinement in direct transcription methods for optimal control". In: *Optimal Control Applications and Methods* 19.1 (1998), pp. 1–21.

[24] Jiang Bian, Dayong Tian, Yuanyan Tang, and Dacheng Tao. "Trajectory Data Classification: A Review". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.4 (2019), pp. 1–34.

[25] Francesco Biral, Enrico Bertolazzi, and Paolo Bosetti. "Notes on Numerical Methods for Solving Optimal Control Problems". In: *IEEJ Journal of Industry Applications* 5.2 (2016), pp. 154–166.

[26] Basilio Bona and Marina Indri. "Friction Compensation in Robotics: an Overview". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 4360–4367.

[27] Frédéric J. Bonnans, Pierre Martinon, Daphne Giorgi, Vincent Grélard, Benjamin Heymann, Stephan Maindrault, and Olivier Tissot. *Bocop – A collection of examples*. Tech. rep. INRIA, 2019.

[28] John V. Breakwell and Ho Yu-Chi. "On the conjugate point condition for the control problem". In: *International Journal of Engineering Science* 2.6 (1965), pp. 565–579.

[29] Michael H. Breitner. "Robust Optimal Onboard Reentry Guidance of a Space Shuttle: Dynamic Game Approach and Guidance Synthesis via Neural Networks". In: *Journal of Optimization Theory and Applications* 107.3 (2000), pp. 481–503.

[30] Arthur E. Bryson Jr. and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control*. CRC Press, 1975. 496 pp.

[31] Christof Büskens and Helmut Maurer. "Real-Time Control of an Industrial Robot using Nonlinear Programming Methods". In: *IFAC Proceedings Volumes* 30.3 (1997), pp. 203–208.

[32] Benjamin S. Cazzolato and Zebb Prime. “On the Dynamics of the Furuta Pendulum”. In: *Journal of Control Science and Engineering* 2011.1 (2011).

[33] Christian M. Chilan and Bruce A. Conway. “Optimal Nonlinear Control using Hamilton–Jacobi–Bellman Viscosity Solutions on Unstructured Grids”. In: *Journal of Guidance, Control, and Dynamics* 43.1 (2020), pp. 30–38.

[34] Christian M. Chilan, Bruce A. Conway, Brendan J. Bialy, and Sharon Stockbridge. “Optimal nonlinear feedback with feedforward control of high speed aerospace vehicles using a spatial statistical approach”. In: *AAS/AIAA Astrodynamics Specialist Conference, 2018*. Univelt Inc., 2018, pp. 997–1016.

[35] Ryan B. Christianson, Ryan M. Pollyea, and Robert B. Gramacy. “Traditional kriging versus modern Gaussian processes for large-scale mining data”. In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 16.5 (2023), pp. 488–506.

[36] Bruce A. Conway. “A Survey of Methods Available for the Numerical Optimization of Continuous Dynamic Systems”. In: *Journal of Optimization Theory and Applications* 152 (2012), pp. 271–306.

[37] Bruce A. Conway. “Evolutionary and Heuristic Methods Applied to Problems in Optimal Control”. In: *Variational Analysis and Aerospace Engineering: Mathematical Challenges for the Aerospace of the Future*. Ed. by A. Frediani, B. Mohammadi, O. Pironneau, and V. Cipolla. Springer, 2016, pp. 117–143.

[38] Lehel Csató and Manfred Opper. “Sparse On-Line Gaussian Processes”. In: *Neural Computation* 14.3 (2002), pp. 641–668.

[39] Bruno da Silva, Gianluca Baldassarre, George Konidaris, and Andrew G. Barto. “Learning parameterized motor skills on a humanoid robot”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5239–5244.

[40] Bruno Da Silva, George Konidaris, and Andrew G. Barto. “Active Learning of Parameterized Skills”. In: *Proceedings of the 31st International Conference on Machine Learning*. PMLR, 2014, pp. 1737–1745.

[41] Phil R. Dahl. *A Solid Friction Model*. Technical Report. El Segundo, CA: The Aerospace Corporation, 1968.

[42] George J. Davis and Max D. Morris. “Six factors which affect the condition number of matrices associated with kriging”. In: *Mathematical Geology* 29.5 (1997), pp. 669–683.

[43] Marc P. Deisenroth, Jan Peters, and Carl E. Rasmussen. “Approximate dynamic programming with Gaussian processes”. In: *2008 American Control Conference*. IEEE, 2008, pp. 4480–4485.

[44] Marc P. Deisenroth, Carl E. Rasmussen, and Jan Peters. “Gaussian process dynamic programming”. In: *Neurocomputing* 72.7 (2009), pp. 1508–1524.

[45] Moritz Diehl, Hans G. Bock, and Johannes P. Schlöder. “A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control”. In: *SIAM Journal on Control and Optimization* 43.5 (2005), pp. 1714–1736.

[46] Moritz Diehl, Hans G. Bock, Johannes P. Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. “Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations”. In: *Journal of Process Control* 12.4 (2002), pp. 577–585.

[47] Moritz Diehl, Rolf Findeisen, and Frank Allgöwer. “A Stabilizing Real-Time Implementation of Nonlinear Model Predictive Control”. In: *Real-Time PDE-Constrained Optimization*. Ed. by L. Biegler, D. Keyes, O. Ghattas, B. van Bloemen Waanders, and M. Heinkenschloss. SIAM, 2007, pp. 25–52.

[48] Moritz Diehl, Rolf Findeisen, Frank Allgöwer, Hans G. Bock, and Johannes P. Schlöder. “Nominal stability of real-time iteration scheme for nonlinear model predictive control”. In: *IEE Proceedings – Control Theory and Applications* 152.3 (2005), pp. 296–308.

[49] Kenji Doya. “Reinforcement Learning in Continuous Time and Space”. In: *Neural Computation* 12.1 (2000), pp. 219–245.

[50] Pierre E. Dupont. “Friction modeling in dynamic robot simulation”. In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 1370–1376.

[51] Nick J. Edwards and C. J. Goh. “Direct training method for a continuous-time nonlinear optimal feedback controller”. In: *Journal of Optimization Theory and Applications* 84.3 (1995), pp. 509–528.

[52] Cees H. Elzinga. “Sequence Similarity: A Nonaligning Technique”. In: *Sociological Methods & Research* 32.1 (2003), pp. 3–29.

[53] Cees H. Elzinga and Matthias Studer. “Spell Sequences, State Proximities, and Distance Metrics”. In: *Sociological Methods & Research* 44.1 (2015), pp. 3–47.

[54] Cees H. Elzinga and Hui Wang. “Versatile string kernels”. In: *Theoretical Computer Science* 495 (2013), pp. 50–65.

[55] Fariba Fahroo and Isaac M. Ross. “Direct Trajectory Optimization by a Chebyshev Pseudospectral Method”. In: *Journal of Guidance, Control, and Dynamics* 25.1 (2002), pp. 160–166.

[56] Ahmad Fakharian, Mohammad-Taghi Hamidi-Beheshti, and Ali Davari. “Solving the Hamilton–Jacobi–Bellman equation using Adomian decomposition method”. In: *International Journal of Computer Mathematics* 87.12 (2010), pp. 2769–2785.

[57] Paola Falugi, Eric Kerrigan, and Eugene Van Wyk. *Imperial College London Optimal Control Software User Guide (ICLOCS)*. Imperial College London. London, England, UK, 2010.

[58] Timm Faulwasser, Tobias Weber, Pablo Zometa, and Rolf Findeisen. “Implementation of Nonlinear Model Predictive Path-Following Control for an Industrial Robot”. In: *IEEE Transactions on Control Systems Technology* 25.4 (2016), pp. 1505–1511.

[59] Rolf Findeisen and Frank Allgöwer. “An introduction to Nonlinear Model Predictive Control”. In: *21st Benelux meeting on systems and control*. Technische Universiteit Eindhoven, 2002, pp. 119–141.

[60] Andre S. Furtado, Despina Kopanaki, Luis O. Alvares, and Vania Bogorny. “Multidimensional Similarity Measuring for Semantic Trajectories”. In: *Transactions in GIS* 20.2 (2016), pp. 280–298.

[61] Katsuhisa Furuta, Masaki Yamakita, and S. Kobayashi. “Swing-up Control of Inverted Pendulum Using Pseudo-State Feedback”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 206.4 (1992), pp. 263–269.

[62] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, 2016, pp. 1050–1059.

[63] Pradipto Ghosh and Bruce A. Conway. “Near-optimal feedback guidance for aeroassisted orbital transfer via spatial statistical prediction”. In: *24th AAS/AIAA Space Flight Mechanics Meeting, 2014*. Univelt Inc., 2014, pp. 2621–2640.

[64] Pradipto Ghosh and Bruce A. Conway. “Near-Optimal Feedback Strategies Synthesized Using a Spatial Statistical Approach”. In: *Journal of Guidance, Control, and Dynamics* 36.4 (2013), pp. 905–919.

[65] Pradipto Ghosh and Bruce A. Conway. “Spatial statistical point prediction guidance for heating-rate-limited aeroassisted orbital transfer”. In: *Acta Astronautica* 111 (2015), pp. 257–269.

[66] Philip E. Gill, Walter Murray, and Michael A. Saunders. "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization". In: *SIAM Review* 47.1 (2005), pp. 99–131.

[67] Philip E. Gill, Walter Murray, and Michael A. Saunders. *User's Guide for SNOPT 5.3: A FORTRAN Package for Large-Scale Nonlinear Programming*. 1997, p. 71.

[68] Philip E. Gill and Elizabeth Wong. *User's Guide for SNCTRL*. University of California. San Diego, CA, USA: Department of Mathematics, 2015.

[69] Torkel Glad. "Modeling of Dynamic Systems from First Principles". In: *Encyclopedia of Systems and Control*. Ed. by J. Baillieul and T. Samad. Springer, 2021, pp. 1286–1291.

[70] C. J. Goh and Kok L. Teo. "MISER: a FORTRAN program for solving optimal control problems". In: *Advances in Engineering Software (1978)* 10.2 (1988), pp. 90–99.

[71] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[72] Ruben Grandia, Farbod Farshidian, Alexey Dosovitskiy, René Ranftl, and Marco Hutter. "Frequency-Aware Model Predictive Control". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1517–1524.

[73] Ruben Grandia, Farbod Farshidian, René Ranftl, and Marco Hutter. "Feedback MPC for Torque-Controlled Legged Robots". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4730–4737.

[74] Sébastien Gros, Mario Zanon, Rien Quirynen, Alberto Bemporad, and Moritz Diehl. "From linear to nonlinear MPC: bridging the gap via the real-time iteration". In: *International Journal of Control* 93.1 (2020), pp. 62–80.

[75] John H. Halton. "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". In: *Numerische Mathematik* 2.1 (1960), pp. 84–90.

[76] Michael Hardt. "Multibody dynamical algorithms, numerical optimal control, with detailed studies in the control of jet engine compressors and biped walking". PhD thesis. San Diego: University of California, 1999.

[77] Heiko Hoffmann, Peter Pastor, Dae-Hyung Park, and Stefan Schaal. "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance". In: *2009 IEEE international Conference on Robotics and Automation*. IEEE, 2009, pp. 2587–2592.

[78] Matanya B. Horowitz. “Efficient Methods for Stochastic Optimal Control”. PhD thesis. Pasadena, California: California Institute of Technology, 2014.

[79] Taylor A. Howell, Chunjiang Fu, and Zachary Manchester. “Direct Policy Optimization Using Deterministic Sampling and Collocation”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5324–5331.

[80] Taylor A. Howell, Brian E. Jackson, and Zachary Manchester. “ALTRO: A Fast Solver for Constrained Trajectory Optimization”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7674–7679.

[81] David H. Jacobson and David Q. Mayne. *Differential Dynamic Programming*. Ed. by R. Bellman. Modern Analytic and Computational Methods in Science and Mathematics 24. Elsevier, 1970.

[82] Hossein Jafari, Saber Ghasempour, and Dumitru Baleanu. “On comparison between iterative methods for solving nonlinear optimal control problems”. In: *Journal of Vibration and Control* 22.9 (2016), pp. 2281–2287.

[83] Matthew R. Jardin and Arthur E. Bryson Jr. “Methods for Computing Minimum-Time Paths in Strong Winds”. In: *Journal of Guidance, Control, and Dynamics* 35.1 (2012), pp. 165–171.

[84] Matthew R. Jardin and Arthur E. Bryson Jr. “Neighboring Optimal Aircraft Guidance in Winds”. In: *Journal of Guidance, Control, and Dynamics* 24.4 (2001), pp. 710–715.

[85] Eugenia Kalnay. *Atmospheric modeling, data assimilation and predictability*. Cambridge University Press, 2003.

[86] Henry J. Kelley. “Guidance theory and extremal fields”. In: *IRE Transactions on Automatic Control* 7.5 (1962), pp. 75–82.

[87] Matthew Kelly. “An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation”. In: *SIAM Review* 59.4 (2017), pp. 849–904.

[88] Alex Kendall and Yarin Gal. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?” In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, pp. 5574–5584.

[89] Ashwin Khadke and Hartmut Geyer. “Policy Decomposition: Approximate Optimal Control with Suboptimality Estimates”. In: *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2021, pp. 185–192.

[90] Ashwin Khadke and Hartmut Geyer. "Sparsity Inducing System Representations for Policy Decompositions". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 6824–6829.

[91] Beomjoon Kim, Albert Kim, Hongkai Dai, Leslie Kaelbling, and Tomas Lozano-Perez. "Generalizing Over Uncertain Dynamics for Online Trajectory Generation". In: *Robotics Research*. Ed. by A. Bicchi and W. Burgard. Vol. 2. Springer, 2018, pp. 39–55.

[92] Andrew Kirmse and Jonathan de Ferranti. "Calculating the prominence and isolation of every mountain in the world". In: *Progress in Physical Geography* 41.6 (2017), pp. 788–802.

[93] Jens Kober, James A. Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[94] Dimitris Kouzoupis, Gianluca Frison, Andrea Zanelli, and Moritz Diehl. "Recent Advances in Quadratic Programming Algorithms for Nonlinear Model Predictive Control". In: *Vietnam Journal of Mathematics* 46.4 (2018), pp. 863–882.

[95] Dana Kulić, Christian Ott, Dongheui Lee, Junichi Ishikawa, and Yoshihiko Nakamura. "Incremental learning of full body motion primitives and their sequencing through human motion observation". In: *The International Journal of Robotics Research* 31.3 (2012), pp. 330–345.

[96] Malte Kuss and Carl E. Rasmussen. "Gaussian Processes in Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. MIT Press, 2003, pp. 751–758.

[97] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, pp. 6402–6413.

[98] Daniel Landgraf, Andreas Völz, Felix Berkel, Kevin Schmidt, Thomas Specker, and Knut Graichen. "Probabilistic prediction methods for nonlinear systems with application to stochastic model predictive control". In: *Annual Reviews in Control* 56 (2023), p. 100905.

[99] Trung Le, Khanh Nguyen, Vu Nguyen, Tu Dinh Nguyen, and Dinh Phung. "GoGP: Fast Online Regression with Gaussian Processes". In: *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 257–266.

[100] Seong-Whan Lee. “Automatic gesture recognition for intelligent human-robot interaction”. In: *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*. IEEE, 2006, pp. 645–650.

[101] Teguh S. Lembono, Antonio Paolillo, Emmanuel Pignat, and Sylvain Calinon. “Memory of Motion for Warm-Starting Trajectory Optimization”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2594–2601.

[102] Sergey Levine and Vladlen Koltun. “Guided Policy Search”. In: *Proceedings of the 30th International Conference on Machine Learning*. PMLR, 2013, pp. 1–9.

[103] Sergey Levine and Vladlen Koltun. “Learning Complex Neural Network Policies with Trajectory Optimization”. In: *Proceedings of the 31st International Conference on Machine Learning*. PMLR, 2014, pp. 829–837.

[104] Frank L. Lewis, Draguna Vrabie, and Vassilis L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[105] Weiwei Li and Emanuel Todorov. “Iterative linear quadratic regulator design for nonlinear biological movement systems”. In: *First International Conference on Informatics in Control, Automation and Robotics*. SciTePress, 2004, pp. 222–229.

[106] Chenggang Liu and Christopher G. Atkeson. “Standing balance control using a trajectory library”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 3031–3036.

[107] Chenggang Liu, Christopher G. Atkeson, and Jianbo Su. “Biped walking control using a trajectory library”. In: *Robotica* 31.2 (2013), pp. 311–322.

[108] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. “When Gaussian Process Meets Big Data: A Review of Scalable GPs”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.11 (2020), pp. 4405–4423.

[109] Marcos Llobera. “Building Past Landscape Perception With GIS: Understanding Topographic Prominence”. In: *Journal of Archaeological Science* 28.9 (2001), pp. 1005–1014.

[110] Ruikun Luo, Rafi Hayne, and Dmitry Berenson. “Unsupervised early prediction of human reaching for human–robot collaboration in shared workspaces”. In: *Autonomous Robots* 42.3 (2018), pp. 631–648.

[111] Devira A. Maharani, Hanif Fakhrurroja, Riyanto Machbub, and Carmadi Machbub. “Hand gesture recognition using K-means clustering and support vector machine”. In: *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE, 2018, pp. 1–6.

[112] Zachary Manchester and Scott Kuindersma. “Robust direct trajectory optimization using approximate invariant funnels”. In: *Autonomous Robots* 43 (2019), pp. 375–387.

[113] Nicolas Mansard, Andrea DelPrete, Mathieu Geisert, Steve Tonneau, and Olivier Stasse. “Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2986–2993.

[114] André L. Marchildon and David W. Zingg. “A Non-intrusive Solution to the Ill-Conditioning Problem of the Gradient-Enhanced Gaussian Covariance Matrix for Gaussian Processes”. In: *Journal of Scientific Computing* 95.3 (2023), p. 65.

[115] Didier Marin and Olivier Sigaud. “Reaching optimally over the workspace: A machine learning approach”. In: *2012 4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, 2012, pp. 1128–1133.

[116] David Mayne. “A Second-order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-time Systems”. In: *International Journal of Control* 3.1 (1966), pp. 85–95.

[117] Wolfgang X. Merkt, Vladimir Ivan, Traiko Dinev, Ioannis Havoutis, and Sethu Vijayakumar. “Memory Clustering Using Persistent Homology for Multimodality- and Discontinuity-Sensitive Learning of Optimal Control Warm-Starts”. In: *IEEE Transactions on Robotics* 37.5 (2021), pp. 1649–1660.

[118] Ali Mesbah, Stefan Streif, Rolf Findeisen, and Richard D. Braatz. “Stochastic nonlinear model predictive control with probabilistic constraints”. In: *2014 American Control Conference*. IEEE, 2014, pp. 2413–2419.

[119] Katja Mombaur, Jean-Paul Laumond, and Anh Truong. “An Inverse Optimal Control Approach to Human Motion Modeling”. In: *Robotics Research. Springer Tracts in Advanced Robotics*. Springer, 2011, pp. 451–468.

[120] Igor Mordatch and Emo Todorov. “Combining the benefits of function approximation and trajectory optimization”. In: *Robotics: Science and Systems*. 2014.

[121] Jun Morimoto and Christopher G. Atkeson. “Minimax Differential Dynamic Programming: An Application to Robust Biped Walking”. In: *Advances in Neural Information Processing Systems*. MIT Press, 2002, pp. 1563–1570.

[122] David D. Morrison, James D. Riley, and John F. Zancanaro. “Multiple shooting method for two-point boundary value problems”. In: *Communications of the ACM* 5.12 (1962), pp. 613–614.

[123] Fabio Muratore. *SimuRLacra - A Framework for Reinforcement Learning from Randomized Simulations*. `https://github.com/famura/SimuRLacra`. 2020.

[124] Fabio Muratore, Christian Eilers, Michael Gienger, and Jan Peters. "Data-Efficient Domain Randomization With Bayesian Optimization". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 911–918.

[125] Fabio Muratore, Fabio Ramos, Greg Turk, Wenhao Yu, Michael Gienger, and Jan Peters. "Robot Learning From Randomized Simulations: A Review". In: *Frontiers in Robotics and AI* 9 (2022).

[126] Fionn Murtagh and Pedro Contreras. "Algorithms for hierarchical clustering: an overview". In: *WIREs Data Mining and Knowledge Discovery* 2.1 (2012), pp. 86–97.

[127] Fionn Murtagh and Pedro Contreras. "Algorithms for hierarchical clustering: an overview, II". In: *WIREs Data Mining and Knowledge Discovery* 7.6 (2017), e1219.

[128] Cory Myers, Lawrence R. Rabiner, and Andrew E. Rosenberg. "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.6 (1980), pp. 623–635.

[129] Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. "Local Gaussian Process Regression for Real Time Online Model Learning". In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2008, pp. 1193–1200.

[130] Duy Nguyen-Tuong, Matthias Seeger, and Jan Peters. "Model Learning with Local Gaussian Process Regression". In: *Advanced Robotics* 23.15 (2009), pp. 2015–2034.

[131] Hassan S. Nik, Sohrab Effati, and Mohammad Shirazian. "An approximate-analytical solution for the Hamilton–Jacobi–Bellman equation via homotopy perturbation method". In: *Applied Mathematical Modelling* 36.11 (2012), pp. 5614–5623.

[132] Shogo Okada, Yoichi Kobayashi, Satoshi Ishibashi, and Toyoaki Nishida. "Incremental learning of gestures for human–robot interaction". In: *AI & society* 25 (2010), pp. 155–168.

[133] Xavier Olive, Luis Basora, Benoit Viry, and Richard Alligier. "Deep Trajectory Clustering with Autoencoders". In: *ICRAT 2020, 9th International Conference for Research in Air Transportation*. 2020.

[134] Henrik Olsson, Karl J. Åström, Carlos C. De Wit, Magnus Gäfvert, and Pablo Lischinsky. "Friction Models and Friction Compensation". In: *European Journal of Control* 4.3 (1998), pp. 176–195.

[135] Nicolai Ommer, Alexander Stumpf, and Oskar von Stryk. “Real-time Online Adaptive Feedforward Velocity Control for Unmanned Ground Vehicles”. In: *Robot World Cup*. Springer, 2017, pp. 3–16.

[136] Martin Otter and Stefan Türk. *The DFVLR Models 1 and 2 of the Manutec r3 Robot*. Tech. rep. DFVLR-Mitt. 88-13, Institut für Dynamik und der Flugsysteme, Oberpfaffenhofen, Germany, 1988.

[137] Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady Andrienko, Natalia Andrienko, Vania Bogorny, Maria L. Damiani, Aris Gkoulalas-Divanis, Jose Macedo, Nikos Pelekis, Yannis Theodoridis, and Zhixian Yan. “Semantic trajectories modeling and analysis”. In: *ACM Computing Surveys (CSUR)* 45.4 (2013), pp. 1–32.

[138] Stephen Paris, John Riehl, and Waldy Sjauw. “Enhanced Procedures for Direct Trajectory Optimization Using Nonlinear Programming and Implicit Integration”. In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. AIAA, 2006.

[139] Chandeok Park and Panagiotis Tsiotras. “Sub-optimal feedback control using a successive wavelet-Galerkin algorithm”. In: *Proceedings of the 2003 American Control Conference*. IEEE, 2003, pp. 1926–1931.

[140] Michael A. Patterson and Anil V. Rao. “GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using Hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming”. In: *ACM Transactions on Mathematical Software* 41.1 (2014), pp. 1–37.

[141] Ettore Pennestrı̀, Valerio Rossi, Pietro Salvini, and Pier P. Valentini. “Review and comparison of dry friction force models”. In: *Nonlinear Dynamics* 83 (2016), pp. 1785–1801.

[142] Hans J. Pesch. “Numerical computation of neighboring optimum feedback control schemes in real-time”. In: *Applied Mathematics and Optimization* 5.1 (1979), pp. 231–252.

[143] Hans J. Pesch, Ingrid Gabler, S. Miesbach, and Michael H. Breitner. “Synthesis of Optimal Strategies for Differential Games by Neural Networks”. In: *New Trends in Dynamic Games and Applications*. Birkhäuser Boston, 1995, pp. 111–141.

[144] Jan Peters and Stefan Schaal. “Policy Gradient Methods for Robotics”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 2219–2225.

[145] Tomasz Piatkowski. “Dahl and LuGre dynamic friction models—The analysis of selected properties”. In: *Mechanism and Machine Theory* 73 (2014), pp. 91–100.

[146] Stylianos Piperakis, Stavros Timotheatos, and Panos Trahanias. “Unsupervised Gait Phase Estimation for Humanoid Robot Walking”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 270–276.

[147] Athanasios S Polydoros and Lazaros Nalpantidis. “Survey of Model-Based Reinforcement Learning: Applications on Robotics”. In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173.

[148] Mauro Pontani and Bruce A. Conway. “Particle Swarm Optimization Applied to Space Trajectories”. In: *Journal of Guidance, Control, and Dynamics* 33.5 (2010), pp. 1429–1441.

[149] Vakhtang Putkaradze and Stuart Rogers. “Constraint Control of Nonholonomic Mechanical Systems”. In: *Journal of Nonlinear Science* 28.1 (2018), pp. 193–234.

[150] Quanser. *QUBE Product Page*. 2023. URL: `https://www.quanser.com/products/qube-servo-2/` (visited on 12/26/2023).

[151] Jeffrey F. Queißer, René F. Reinhart, and Jochen J. Steil. “Incremental bootstrapping of parameterized motor skills”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp. 223–229.

[152] Jeffrey F. Queißer and Jochen J. Steil. “Bootstrapping of Parameterized Skills Through Hybrid Optimization in Task and Policy Spaces”. In: *Frontiers in Robotics and AI* 5 (2018).

[153] Joaquin Quiñonero-Candela and Carl E. Rasmussen. “A Unifying View of Sparse Approximate Gaussian Process Regression”. In: *Journal of Machine Learning Research* 6 (2005), pp. 1939–1959.

[154] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[155] Emma Reznick, Kyle R. Embry, Ross Neuman, Edgar Bolívar-Nieto, Nicholas P. Fey, and Robert D. Gregg. “Lower-limb kinematics and kinetics during continuously varying human locomotion”. In: *Scientific Data* 8.1 (2021), p. 282.

[156] Isaac M. Ross. *Enhancements to the DIDO Optimal Control Toolbox*. 2020. arXiv: `2004.13112`.

[157] Francesco Ruscelli, Arturo Laurenzi, Nikos G. Tsagarakis, and Enrico Mingo Hoffman. “Horizon: A Trajectory Optimization Framework for Robotic Systems”. In: *Frontiers in Robotics and AI* 9 (2022).

[158] Hiroaki Sakoe and S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49.

[159] Stefan Schaal. “Learning from Demonstration”. In: *Advances in Neural Information Processing Systems*. MIT Press, 1997, pp. 1040–1046.

[160] Stefan Schaal and Christopher G. Atkeson. “Learning Control in Robotics”. In: *IEEE Robotics & Automation Magazine* 17.2 (2010), pp. 20–29.

[161] John D. Schierman, David G. Ward, Jason R. Hull, Neha Gandhi, Michael Oppenheimer, and David B. Doman. “Integrated Adaptive Guidance and Control for Re-Entry Vehicles with Flight Test Results”. In: *Journal of Guidance, Control, and Dynamics* 27.6 (2004), pp. 975–988.

[162] Falko Schmid, Kai-Florian Richter, and Patrick Laube. “Semantic Trajectory Compression”. In: *Advances in Spatial and Temporal Databases*. 2009, pp. 411–416.

[163] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. 2nd ed. Springer Handbooks. Springer, 2016.

[164] Bharat Singh, Rajesh Kumar, and Vinay Pratap Singh. “Reinforcement learning in robotic applications: a comprehensive survey”. In: *Artificial Intelligence Review* 55.2 (2022), pp. 945–990.

[165] Sandeep K. Singh and John L. Junkins. “Stochastic learning and extremal-field map based autonomous guidance of low-thrust spacecraft”. In: *Scientific Reports* 12.1 (2022), p. 17774.

[166] Nicki Skafte, Martin Jørgensen, and Søren Hauberg. “Reliable training and estimation of variance networks”. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019, pp. 6326–6336.

[167] Edward Snelson and Zoubin Ghahramani. “Sparse Gaussian Processes using Pseudo-inputs”. In: *Advances in Neural Information Processing Systems*. MIT Press, 2005, pp. 1257–1264.

[168] Yunlong Song and Davide Scaramuzza. “Learning High-Level Policies for Model Predictive Control”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 7629–7636.

[169] Yunlong Song and Davide Scaramuzza. “Policy Search for Model Predictive Control With Application to Agile Drone Flight”. In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2114–2130.

[170] Jason L. Speyer and David H. Jacobson. *Primer on Optimal Control Theory*. SIAM, 2010.

[171] Maximilian Stelzer and Oskar von Stryk. "Efficient forward dynamics simulation and optimization of human body dynamics". In: *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 86.10 (2006), pp. 828–840.

[172] Martin Stolle and Christopher G. Atkeson. "Policies based on trajectory libraries". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE, 2006, pp. 3344–3349.

[173] Oskar von Stryk. "Numerical Solution of Optimal Control Problems by Direct Collocation". In: *Optimal Control: Calculus of Variations, Optimal Control Theory and Numerical Methods*. Ed. by R. Bulirsch, A. Miele, J. Stoer, and K. Well. Vol. 111. ISNM International Series of Numerical Mathematics. Birkhäuser Basel, 1993, pp. 129–143.

[174] Oskar von Stryk and Roland Bulirsch. "Direct and indirect methods for trajectory optimization". In: *Annals of Operations Research* 37.1 (1992), pp. 357–373.

[175] Oskar von Stryk and Maximilian Schlemmer. "Optimal Control of the Industrial Robot Manutec r3". In: *Computational Optimal Control*. Birkhäuser Basel, 1994, pp. 367–382.

[176] Matthias Studer and Gilbert Ritschard. "What Matters in Differences Between Life Trajectories: A Comparative Review of Sequence Dissimilarity Measures". In: *Journal of the Royal Statistical Society Series A: Statistics in Society* 179.2 (2015), pp. 481–511.

[177] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. "A survey of trajectory distance measures and performance evaluation". In: *The VLDB Journal* 29.1 (2020), pp. 3–32.

[178] Linfeng Su, Jinbo Wang, and Hongbo Chen. "A Real-Time and Optimal Hypersonic Entry Guidance Method Using Inverse Reinforcement Learning". In: *Aerospace* 10.11 (2023), p. 948.

[179] Cynthia Sung, Dan Feldman, and Daniela Rus. "Trajectory clustering for motion prediction". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 1547–1552.

[180] Richard S. Sutton, Andrew G. Barto, and Ronald J. Williams. "Reinforcement learning is direct adaptive optimal control". In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 19–22.

[181] Yaguang Tao, Alan Both, Rodrigo I. Silveira, Kevin Buchin, Stef Sijben, Ross S. Purves, Patrick Laube, Dongliang Peng, Kevin Toohey, and Matt Duckham. “A comparative analysis of trajectory similarity measures”. In: *GIScience & Remote Sensing* 58.5 (2021), pp. 643–669.

[182] Yuval Tassa, Tom Erez, and William Smart. “Receding horizon differential dynamic programming”. In: *Advances in neural information processing systems*. Curran Associates, Inc., 2007, pp. 1465–1472.

[183] Yuval Tassa, Tom Erez, and Emanuel Todorov. “Synthesis and stabilization of complex behaviors through online trajectory optimization”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[184] Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W. Roberts. “LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification”. In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052.

[185] Inc. The MathWorks. *Topographic Prominence*. 2023. URL: `https://de.mathworks.com/help/signal/ug/prominence.html` (visited on 01/05/2024).

[186] Michalis Titsias. “Variational Learning of Inducing Variables in Sparse Gaussian Processes”. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. PMLR, 2009, pp. 567–574.

[187] Emanuel Todorov. “Efficient computation of optimal actions”. In: *Proceedings of the National Academy of Sciences* 106.28 (2009), pp. 11478–11483.

[188] Emanuel Todorov and Weiwei Li. “A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems”. In: *Proceedings of the 2005, American Control Conference*. IEEE, 2005, pp. 300–306.

[189] Emanuel Todorov and Yuval Tassa. “Iterative local dynamic programming”. In: *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 2009, pp. 90–95.

[190] Panagiotis Tsiotras and Ricardo Sanz Diaz. “Real-Time Near-Optimal Feedback Control of Aggressive Vehicle Maneuvers”. In: *Optimization and Optimal Control in Automotive Systems systems*. Ed. by H. Waschl, I. Kolmanovsky, M. Steinbuch, and L. del Re. Springer, 2014, pp. 109–129.

[191] Srinivas R. Vadali and Rajnish Sharma. “Optimal Finite-Time Feedback Controllers for Nonlinear Systems with Terminal Constraints”. In: *Journal of Guidance, Control, and Dynamics* 29.4 (2006), pp. 921–928.

[192] Marica Vagni, Noemi Giordano, Gabriella Balestra, and Samanta Rosati. “Comparison of different similarity measures in hierarchical clustering”. In: *2021 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. 2021, pp. 1–6.

[193] Matias Valdenegro-Toro and Daniel S. Mori. “A Deeper Look into Aleatoric and Epistemic Uncertainty Disentanglement”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2022, pp. 1508–1516.

[194] Shrihari Vasudevan, Fabio Ramos, Eric Nettleton, and Hugh Durrant-Whyte. “Gaussian process modeling of large-scale terrain”. In: *Journal of Field Robotics* 26.10 (2009), pp. 812–840.

[195] Neil Vaughan and Bogdan Gabrys. “Comparing and Combining Time Series Trajectories Using Dynamic Time Warping”. In: *Procedia Computer Science* 96 (2016), pp. 465–474.

[196] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. “acados – a modular open-source framework for fast embedded optimal control”. In: *Mathematical Programming Computation* 14.1 (2022), pp. 147–183.

[197] Sethu Vijayakumar, Aaron D’Souza, and Stefan Schaal. “Incremental Online Learning in High Dimensions”. In: *Neural Computation* 17.12 (2005), pp. 2602–2634.

[198] Sethu Vijayakumar and Stefan Schaal. “Locally Weighted Projection Regression: Incremental Real Time Learning in High Dimensional Space”. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann Publishers, 2000, pp. 1079–1086.

[199] Andreas Wächter and Lorenz T. Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical Programming* 106.1 (2006), pp. 25–57.

[200] Zhixun Wen, Haiqing Pei, Hai Liu, and Zhufeng Yue. “A Sequential Kriging reliability analysis method with characteristics of adaptive sampling regions and parallelizability”. In: *Reliability Engineering & System Safety* 153 (2016), pp. 170–179.

[201] Brian Wilcox and Michael C. Yip. “SOLAR-GP: Sparse Online Locally Adaptive Regression Using Gaussian Processes for Bayesian Robot Model Learning and Control”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2832–2839.

[202] Christopher K. I. Williams. “Computation with Infinite Neural Networks”. In: *Neural Computation* 10.5 (1998), pp. 1203–1216.

[203] Sheran Wiratunga. “Training Gaussian Process Regression Models Using Optimized Trajectories”. Master Thesis. University of Waterloo, 2014.

[204] Sebastian Wolf, Giorgio Grioli, Oliver Eiberger, Werner Friedl, Markus Grebenstein, Hannes Höppner, Etienne Burdet, Darwin G. Caldwell, Raffaella Carloni, Manuel G. Catalano, Dirk Lefeber, Stefano Stramigioli, Nikos Tsagarakis, Michaël Van Damme, Ronald Van Ham, Bram Vanderborght, Ludo C. Visser, Antonio Bicchi, and Alin Albu-Schäffer. “Variable Stiffness Actuators: Review on Design and Components”. In: *IEEE/ASME Transactions on Mechatronics* 21.5 (2016), pp. 2418–2430.

[205] Waldemar Wroblewski. “Implementation of a model predictive control algorithm for a 6DOF Manipulator – simulation results”. In: *Proceedings of the Fourth International Workshop on Robot Motion and Control (IEEE Cat. No. 04EX891)*. IEEE, 2004, pp. 209–212.

[206] Xinyang Wu, Mohamed El-Shamouty, Christof Nitsche, and Marco F. Huber. “Uncertainty-Guided Active Reinforcement Learning with Bayesian Neural Networks”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5751–5757.

[207] Katsu Yamane, Marcel Revfi, and Tamim Asfour. “Synthesizing object receiving motions of humanoid robots with human motion database”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1629–1636.

[208] Di Yao, Chao Zhang, Zhihua Zhu, Jianhui Huang, and Jingping Bi. “Trajectory clustering via deep representation learning”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3880–3887.

[209] Gwonjin Yi and Junghoon Jee. “Search Space Reduction In Motion Matching by Trajectory Clustering”. In: *SIGGRAPH Asia 2019 Posters*. ACM, 2019.

[210] Andrea Zanelli, Greg Horn, Gianluca Frison, and Moritz Diehl. “Nonlinear Model Predictive Control of a Human-sized Quadrotor”. In: *2018 European Control Conference (ECC)*. IEEE, 2018, pp. 1542–1547.

[211] Christoph Zelch, Jan Peters, and Oskar von Stryk. “Approximate Policy Representation: A Comparison of GPs vs Deep Neural Networks”. Submitted. 2024.

[212] Christoph Zelch, Jan Peters, and Oskar von Stryk. “Clustering of Motion Trajectories by a Distance Measure Based on Semantic Features”. In: *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*. IEEE, 2023, pp. 1–8.

[213] Christoph Zelch, Jan Peters, and Oskar von Stryk. “Learning Control Policies from Optimal Trajectories”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2529–2535.

[214] Christoph Zelch, Jan Peters, and Oskar von Stryk. “Start State Selection for Control Policy Learning from Optimal Trajectories”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3247–3253.

[215] Jie Zhao, Moritz Diehl, Richard Longman, Hans G. Bock, and Johannes P. Schloeder. “Nonlinear Model Predictive Control of Robots Using Real-time Optimization”. In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. AIAA, 2004.

[216] Wenshuai Zhao, Jorge P. Queralta, and Tomi Westerlund. “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[217] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. “Value function approximation and model predictive control”. In: *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 100–107.

Own Publications

Christoph Zelch, Jan Peters, and Oskar von Stryk. “Clustering of Motion Trajectories by a Distance Measure Based on Semantic Features”. In: *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*. IEEE, 2023, pp. 1–8.

Christoph Zelch, Jan Peters, and Oskar von Stryk. “Learning Control Policies from Optimal Trajectories”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2529–2535.

Christoph Zelch, Jan Peters, and Oskar von Stryk. “Start State Selection for Control Policy Learning from Optimal Trajectories”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3247–3253.

Use of Tools Based on Artificial Intelligence

The AI-based online translation service DeepL[1] has been used for writing this thesis to translate single words, phrases, or full sentences, to find synonyms, and to check phrases and sentences for grammatical correctness.

The DeepL Write assistant[2] and Grammarly[3] have been utilized to identify spelling and grammatical errors in the text and enhance formulations.

The AI Photo Enhancer[4] has been used to enhance Figures 5.1, 5.4 and 3.6a, i.e., to increase the resolution, reduce the image noise, improve the color and saturation and sharpen the images.

[1] https://www.deepl.com/translator
[2] https://www.deepl.com/write
[3] https://app.grammarly.com/
[4] https://www.artguru.ai/photo-enhancer/

Wissenschaftlicher Werdegang

2011	Allgemeine Hochschulreife
2011 – 2017	Studium der Mathematik an der Technischen Universität Darmstadt
2017	Erlangung des akademischen Grades Master of Science (M.Sc.) der Mathematik
2017 – 2024	Wissenschaftlicher Mitarbeiter und Doktorand am Fachbereich Informatik der Technischen Universität Darmstadt
2024	Disputation zur Erlangung des akademischen Grades eines Doktor-Ingenieurs

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 7. Februar 2024

C. Zelch